# DSL : Simulation Sensor Language

Berlioz Alison - Fezaï Ahmed - Faguet Guillaume

27 février 2018

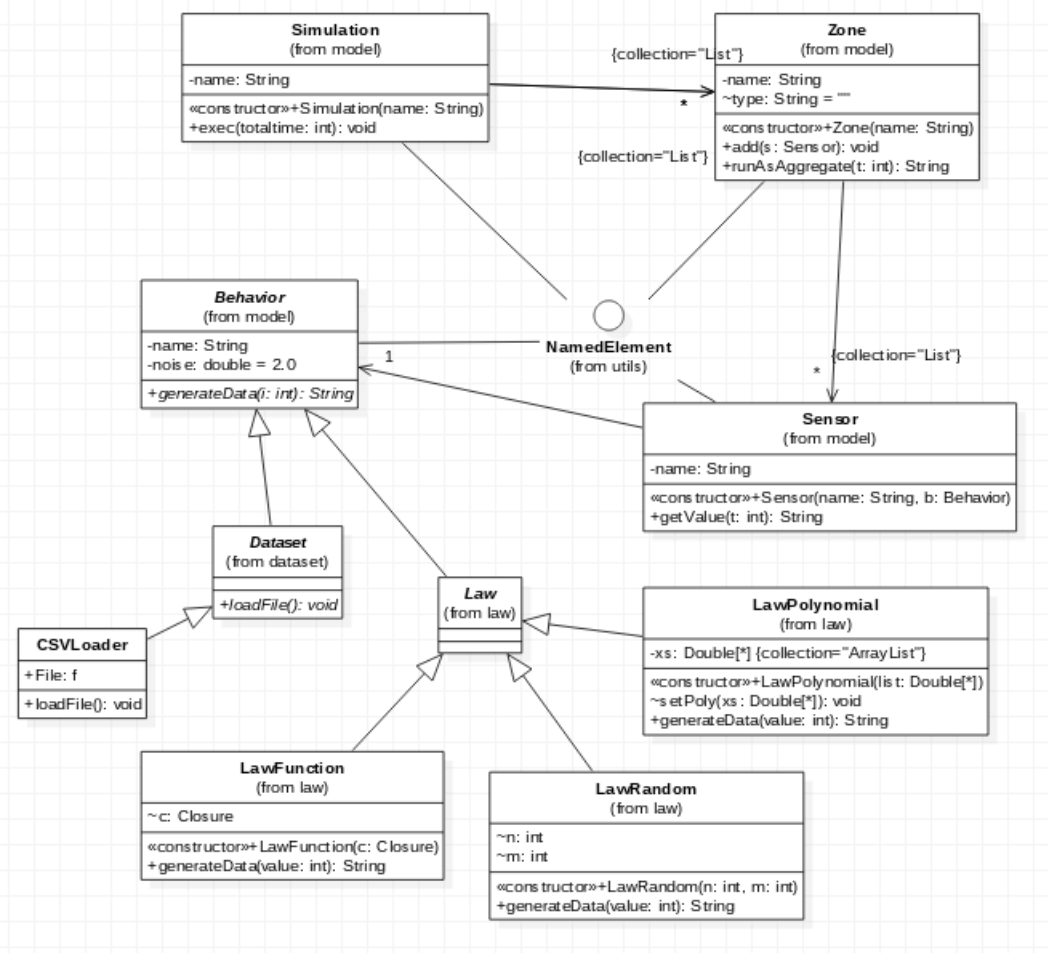
# Introduction

L'objectif de ce projet est la réalisation d'un DSL dans un contexte d'une simulation d'un environnement de capteur. Le langage doit permettre la définition des éléments composant la simulation soit les lois de simulation, les zones de capteurs et les capteurs eux-mêmes. Les lois doivent être variées et il doit être possible de visualiser les données de nos capteurs. Le choix de l'extension s'est porté sur les Composites Sensors et le langage sur Groovy.

# Modèle et Syntaxe Abstraite

## Diagramme de Classe

Notre langage suit le modèle suivant :




L'objectif suivi par ce modèle est d'approcher le réel afin de rendre la compréhension du langage, par un utilisateur peu compétent en programmation, la plus simple possible.

### Description du Modèle

À chaque exécution d'un script, un objet simulation est également créé. Il s'agit du composant principal qui se composera par la suite de différentes Zones et Sensors. Chaque sensor possède un Behavior qui caractérise les données qu'il génèrera. Les Behaviors se caractérisent en deux catégories, les Law qui reposent sur les retours de valeurs générées durant la simulation et les Datasets qui permettent le replay de valeurs déjà enregistrées.

### Analyse des choix d'architecture

Dans le monde réel, un utilisateur va chercher à faire une simulation qui se compose de capteur avec un comportement et placés dans des lieux. Le modèle, vu ici, rassemble tous les objets concrets et abstraits que manipulent des utilisateurs de sensor, ce qui favorise la prise en main du langage. La différenciation des behaviors en law et dataset a, actuellement, peu d'intérêt pratique. L'objectif premier était de faciliter l'implémentation de nouvelles fonctionnalités toujours en étant proche du réel. Ils en étaient de même pour les zones dans la première partie du projet mais l'extension à changer ce constat. Les zones sont des rassemblements de Sensors et représentent des lieux géographiques. Leur présence dans le modèle permet le fonctionnement des composites Sensors. En effet, on peut définir une méthode d'agrégation des sensors pour une zone. Une remarque importante également, chaque composant possède également un nom qui va permettre d'identifier les objets que définira l'utilisateur.

## Syntaxe Concrète et fonctionnalitées

Dans cette partie, nous allons décrire la syntaxe des différentes fonctionnalités implémentées. Le but de la syntaxe concrète coïncide avec la facilité d'assimilation de la syntaxe abstraite. En effet, ici on va chercher à produire des petites phrases qui coulent de source quand estimant qu'elles vont produire.On commence toujours par définir le nombre d'itérations de notre simulation.

```
duration 10
```

Chaque itération correspond à une heure dans la base de données. Ce n'est pas très pertinent comme unité de temps mais cela permet de visualiser facilement des résultats.Trois types de lois ont été définis dans notre langage. Chaque loi a besoin de champs différents en fonction de son type. C'est pourquoi nous avons décidé d'utiliser une classe abstraite Law (qui se spécifie en trois types) pour permettre une implémentation plus claire des fonctionnalités.

### Loi aléatoire

La première, la plus simple, est la lois aléatoire qui renvoie un nombre aléatoire entre deux entiers.

```
lawRandom "random" between 10 and 20
```

Cette loi est un peu passe-partout et permet de simuler, grossièrement, une grande variété de capteur. Un capteur de température qui varie entre 20 et 25 degrés par exemple.

### Loi Polynomiale

La seconde loi est la loi polynomiale servant à manipuler des polynômes. Et se comporte de la façon suivante :

```
lawPolynomial "poly" value ([1.0d,2.0d,3.0d])
```

La liste de double ici représente les coefficients a,b,c (etc...) tels que $y = a+bx+cx^2$. Cette méthode permet de définir n'importe quel polynôme, par contre on suppose que les utilisateurs ne vont pas chercher à former un polynôme comme "ax puissance 1000" par exemple. Cela nécessiterait l'écrire d'une liste de 1000 caractères ce qui n'est pas pratique. D'un autre côté on voit mal l'intérêt d'une telle définition.

### Loi Fonction

```
lawFunction "fonction" , {
    t ->
    if (t < 10) return 1
    if (t > 10 && t < 20) return 2
    if (t > 20) return 2
}
```

La dernière loi implémentée permet la définition de fonction via les closures de groovy. On considère ici que l'utilisateur sait comment écrire correctement une fonction. Dans le cas contraire, une erreur survient. Les closures de groovy peuvent être directement utilisés dans les langages et sont considérées comme un type de données au même titre que les int ou les doubles. La Closure nous permet ici d'avoir un degré de liberté importante quant à la définition possible de fonction.

### Zone et Sensor

On peut définir des zones de la façon suivante :

```
zone "parking"
```

Les sensors quant à eux ce définissent comme suit :

```
sensor "try" using "poly" on "classroom_0+106"
sensor "rand" using "random" on "roof_E"
sensor "test" using "poly" on "parking"
sensor "test2" using "poly2" on "parking"

lot 3 using "random" on "roof_E"
```

Le sensor est relié en une zone à la création mais il est possible de ne pas définir la zone auparavant par soucis de clarté. On peut également définir tout un paquet de sensors qui se comportent de façon similaire avec le mot clé "lot".

### Composite Sensor

Ce sont les zones qui vont nous servir à simuler des composites sensors. On signale simplement quelle zone on veut composer et de quelle façon avec la syntaxe suivante :

```
aggregate "parking" using "min"
```

On considère que l'utilisateur sait ce qu'il fait et donc qu'il est en mesure de définir la meilleure façon de composer ses sensors. Par contre, on lui propose seulement quelques opérations d'agrégation et si le projet devait se poursuivre l'amélioration de cette fonctionnalité serait de rigueur. Donc pour le moment on a choisi les opérations "max", "min", "sum" et "mean".

### Replay de CSV

On utilise la syntaxe suivante pour le replay d'un capteur via un csv.

```
applicatelaw "test" from "result/Simulation.csv" to "result/output.csv"
```

On prend simplement le chemin relatif du fichier de données avec le nom du capteur que l'on veut replay. Concrètement, le replay de csv ne se comporte pas encore comme un Behavior car il ne se relie pas à un sensor. Les valeurs sont bien extraites par contre.

### Choix du mode d'écriture

Il est très vite apparue que pour tester la validité des fonctionnalités, plusieurs façons de visualiser les résultats de simulation étaient requises. On peu donc choisir si l'on veut un retour dans le terminal directement, dans un fichier csv qui peut nous servir à un éventuel replay ou dans la base de données pour pouvoir observer les courbes avec graphana.

```
writeOn "csv"
//writeOn "influxDB"
//writeOn "terminal"
```

### Lancement de la Simulation

Pour finir, le mot clé "go" suivi du nom que l'on veut donner à notre simulation permet de lancer concrètement la simulation une fois complète.

```
go "Simulation"
```

Cette fonctionnalité permet notamment de bien identifier la dernière ligne du script pour l'utilisateur.

## Analyse Critique

Groovy est un langage très intéressant en ce qui concerne la création de dsl. En effet, sa flexibilité importante en matière de syntaxe ainsi que les divers outils qu'il propose en font un candidat de choix. Surtout que pour une implémentation de dsl interne, on se retrouve avec un pouvoir de définition proche d'un langage externe. Les outils que proposent Groovy sont la Closure, utiliser dans notre cas pour définir des fonctions complexes avec facilité, et les méthodes qui peuvent être définies directement dans le script. Dans notre cas on n'a pas utilisé la méthode car trop proche du milieu du programmeur et pas assez de l'utilisateur.
Groovy propose également un Binding qui permet de sauvegarder tous nos objets secondaires grâce à leur nom, ce qui est très pratique pour sauvegarder des objets via leur nom et ce quel que soit leur type. Par ailleurs, le banding fonctionne très bien dans le contexte SSL qui cherche souvent à définir des zones ou sensors nommées. Au niveau de l'implémentation, il nous suffit alors de récupérer les noms des objets à manipuler depuis le banding et le tour est joué.

## Pour conclure : DSL et SSL

Concrètement, la mise en place d'un dsl dans le cadre de la simulation de Sensor a pour objectif de proposer à un utilisateur, confirmer dans le domaine des simulations de capteur mais peu compétent en programmation, la possibilité de concevoir facilement sa propre simulation de capteur. Dans le but de limiter la charge cognitive due à l'apprentissage du DSL, le langage cherche à être le plus intuitif possible relatif au domaine. De plus, le langage produit est court et concis, peut être facilement étendu et est simple à maintenir.