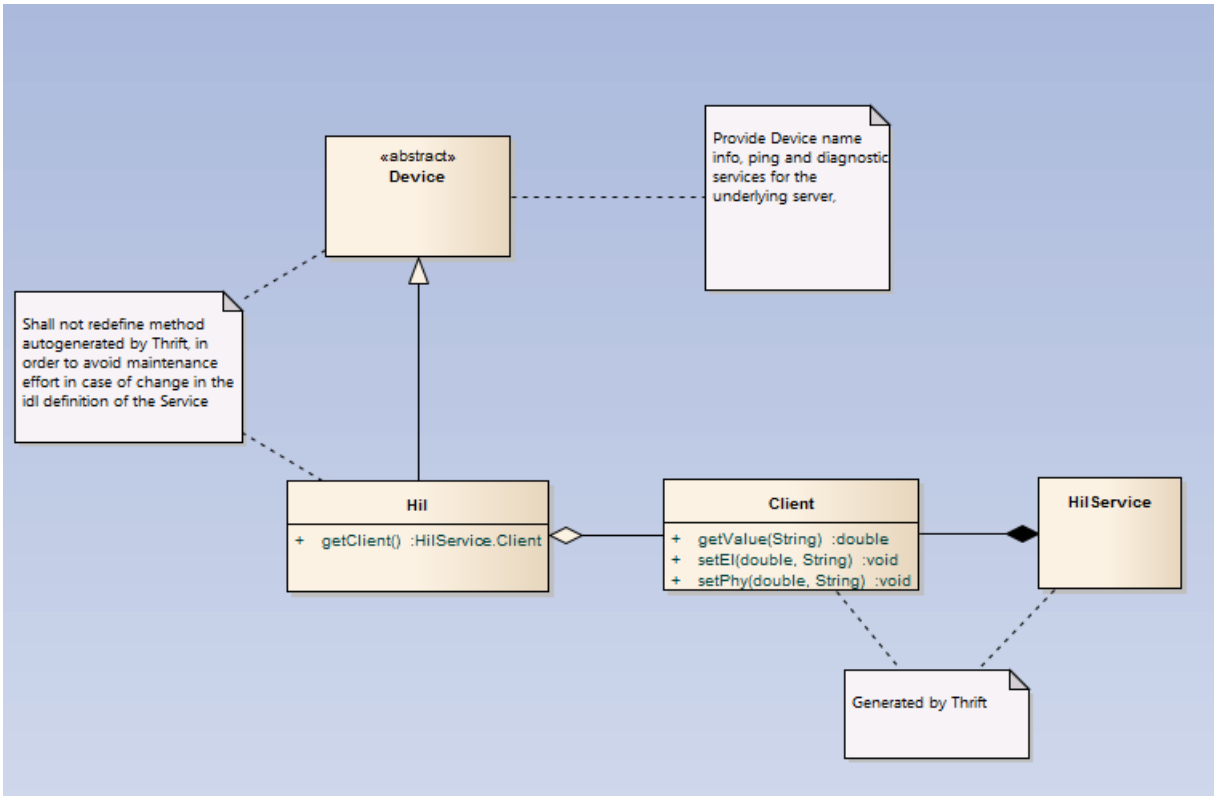
GreenT : plateforme de tests automatisés

Les différents modèles que vous verrez dans ce document ont été réfléchis lors de réunion auxquels j'étais présent, nous étions donc 2 ou 3 à penser ces modèles.

Les devices

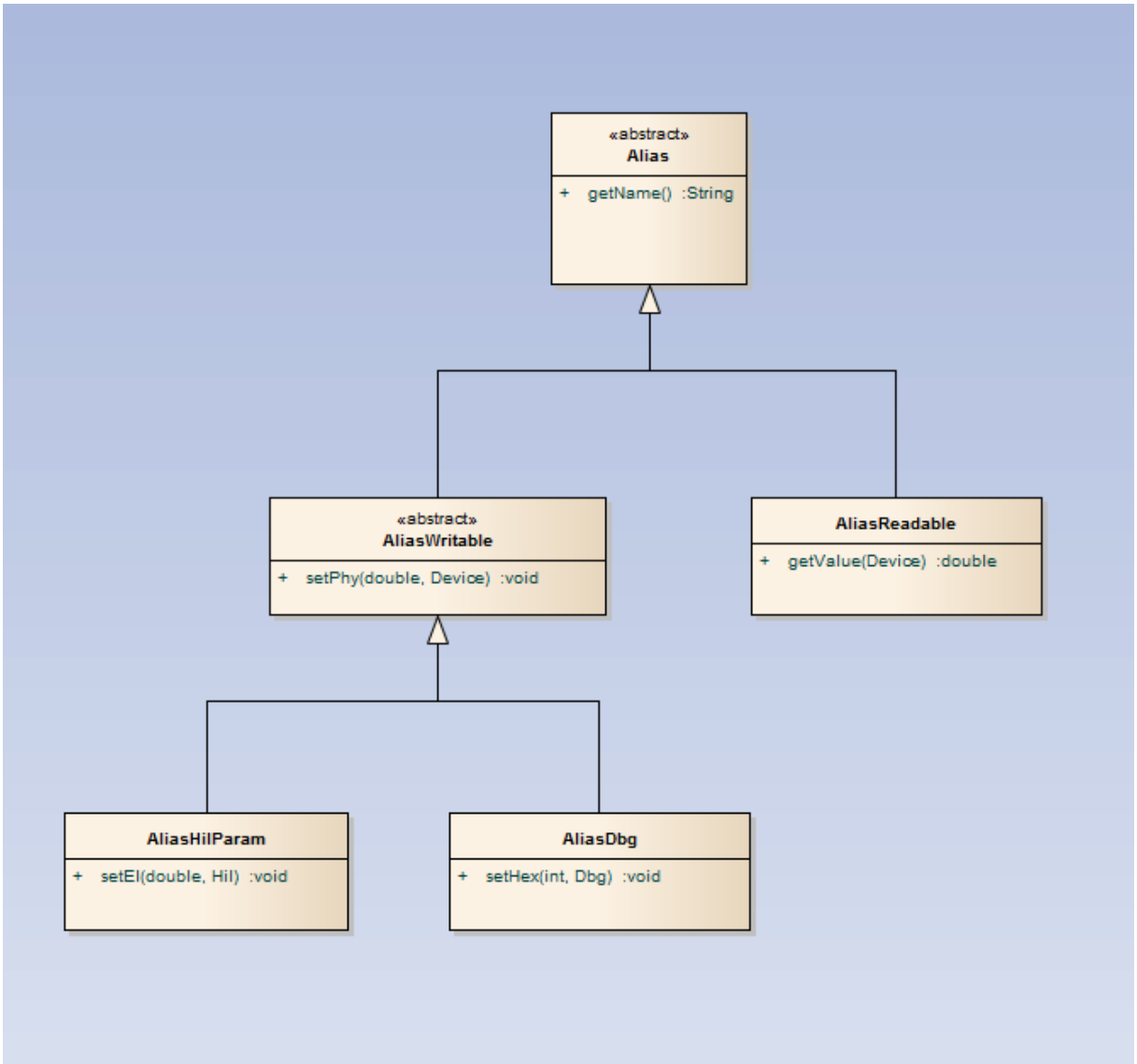


Un device peut être un Hil, un Débuggeur ou tout autre appareil distant. Notre bibliothèque communique avec un device à l'aide de Thrift, qui permet de faire communiquer du Java et du python de la même manière que le service RMI.

En cas de nouveau device, un fichier Thrift devra être déclaré afin de générer un nouveau client et son service associé.

setEl corresond à un set électrique, qui permet de mettre une valeur électrique, setPhy quant à lui est pour une valeur physique.

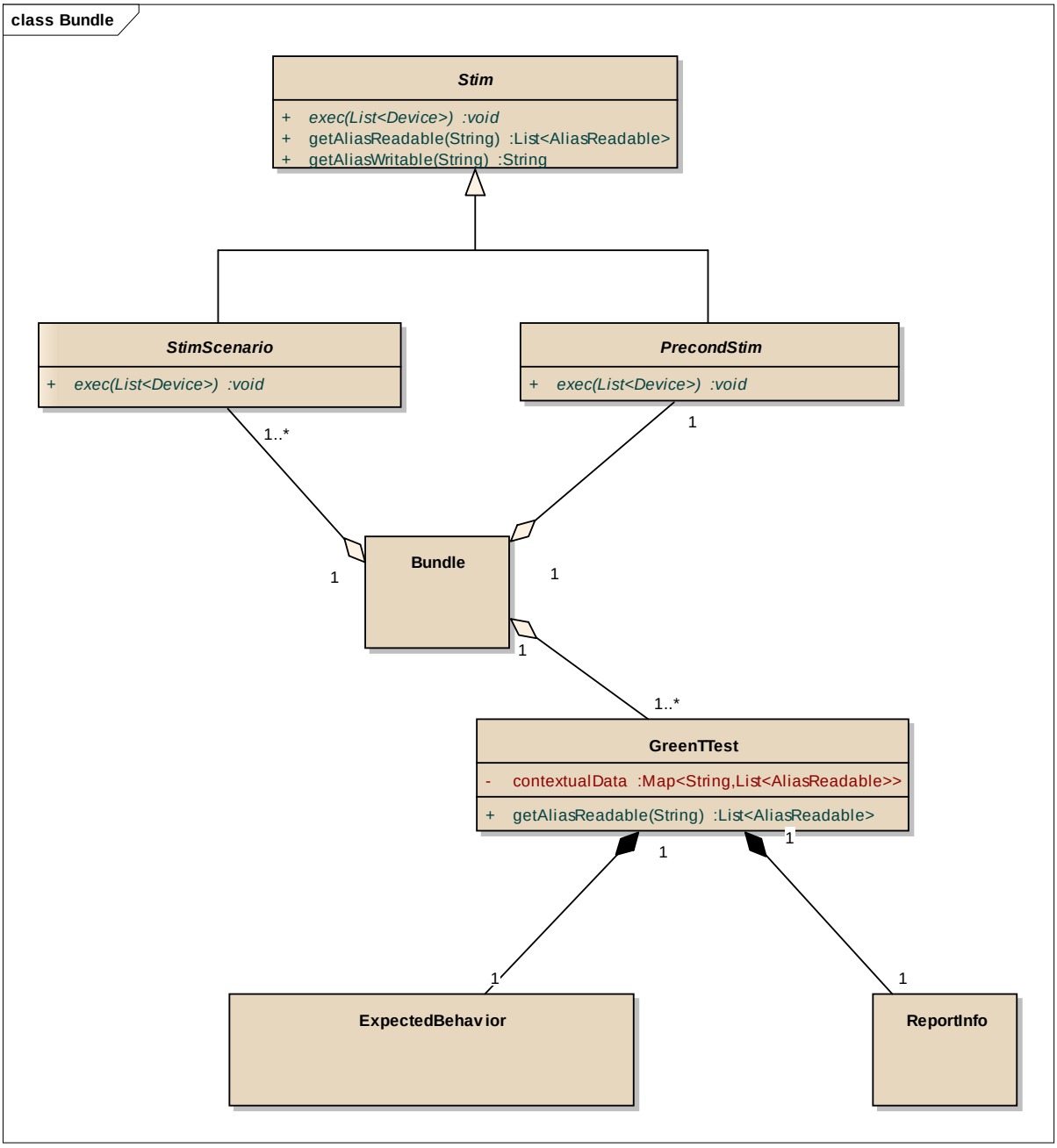
Gestion des alias



Un alias permet d'accéder à un élément présent sur le Device passé en paramètre, il permet de faire la relation entre une adresse dans la base de données et un nom : ici, le modèle contient un Hil et un Dbg, d'autres Alias pourront apparaître si ceux-ci existe dans le modèle présenté précédemment.

En effet, en fonction des devices, nous n'avons pas les même type de « set » (set hexadécimal pour un débuggeur alors qu'un hil a un set électrique par exemple)

Exécution des tests



Un precondstim est un scénario de précondition qui sera lancé au début d'un bundle : démarrer le contrôleur, mettre des valeurs à certaines variables, ….
StimScenario sont des scénarios permettant de stimuler le device avant de lancer le test proprement dit.
Une expectedBehavior est le test proprement dit : une expression permettant de dire si un test est vert, jaune ou rouge.
Un bundle est un package de tests pouvant être exécutés simultanément sur le même workbench : même préconditions, même scénarios de stimulations, seul les expectedBehavior sont différents.

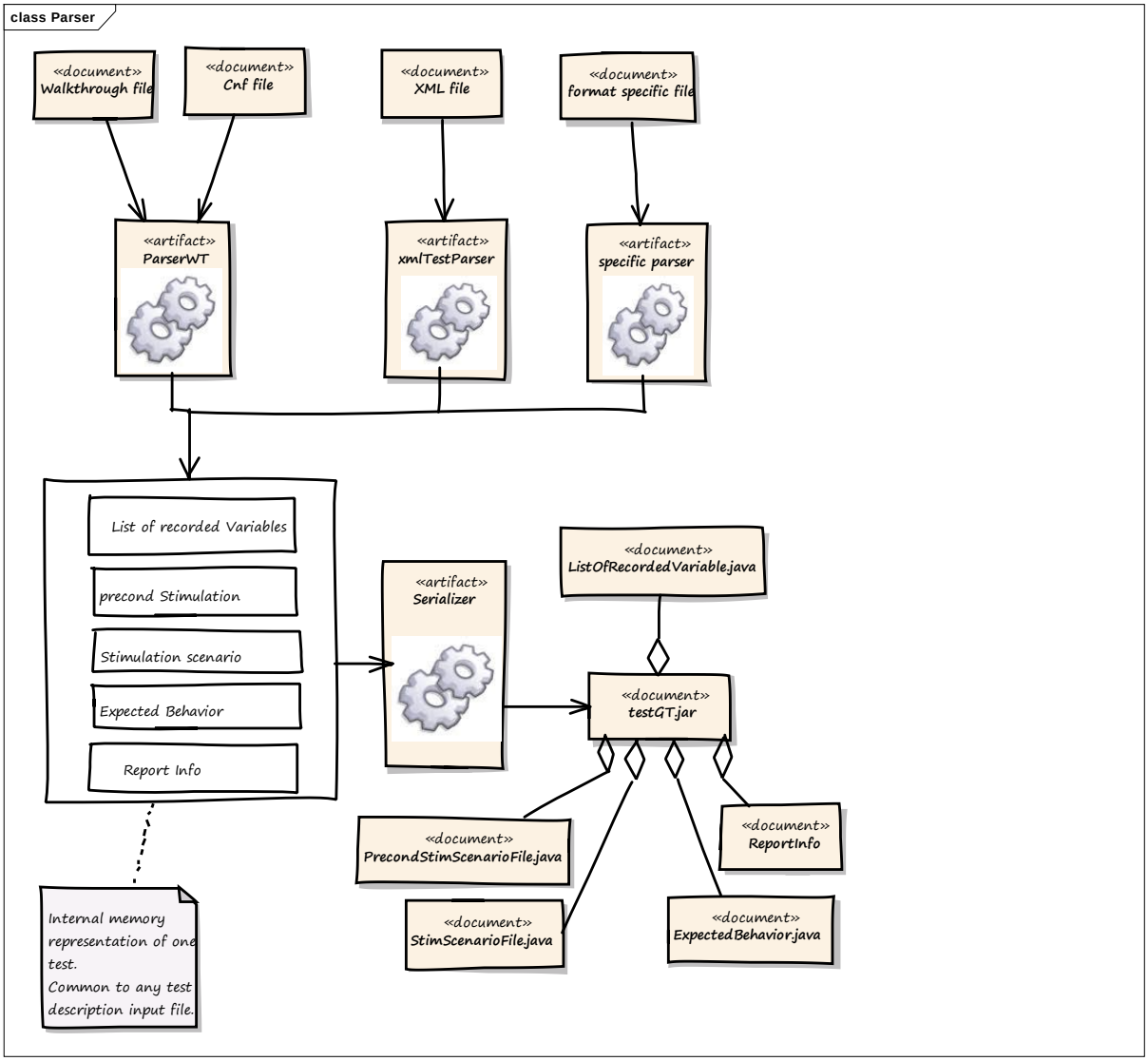
Je dois générer 4 types de fichier : un ensemble de classes héritant de StimScenario, une classe héritant de PreCondStim une classe héritant de GreenTTest et un ExpectedBehavior.

Un scénario de stimulation et un precondstim contienne chacun une méthode exec(Device). Celles-ci contiennent l'exécution du scénario et de la précondition de tests, elles seront lancés par le Bundle.

Le GreenTTest quant-à lui contient une expectedBehavior : expression permettant de donner la couleur du test (vert, jaune, rouge), le fichier contient également des informations sur le test (nom de la variable de test, description, sévérité, commentaires, responsable du test, …)

Parser

Ci-dessous, le fonctionnement général du parser. Actuellement, nous ne nous occupons que de la partie ParserWT correspondant au fichier Walkthrough : un fichier excel.



Le fichier excel contient un certain nombre de colonnes, dont une colonne precondstim, stimscenario, exepectedbehiavor.

Le fichier cnf quant-à lui contient les adresses de certains alias utilisés, et le contenu de certaines actions.

Voici un exemple de chaîne pouvant été parsé, chaines présente dans une cellule stim scenario ou precondstim :

```
SUB_SCENARIO_VS_0_50_0 : {
        HIL_VS = 0;
        CHECK(HIL_VS_OUT, 0,  TOLRES(1)) ;
        HIL_VS = 0;
        CHECK(HIL_VS_OUT, 50,  TOLPER(1)) ;
```

```
        HIL_VS = 0;
        CHECK(HIL_VS_OUT,  0, TOLRES(1));
}
```

Nous avons créé notre propre grammaire, en utilisant Antlr. Antlr s'occupe de générer toute la partie permettant d'effectuer le parcours de l'arbre syntaxique. Il ne reste plus qu'à interpréter les mots clés.

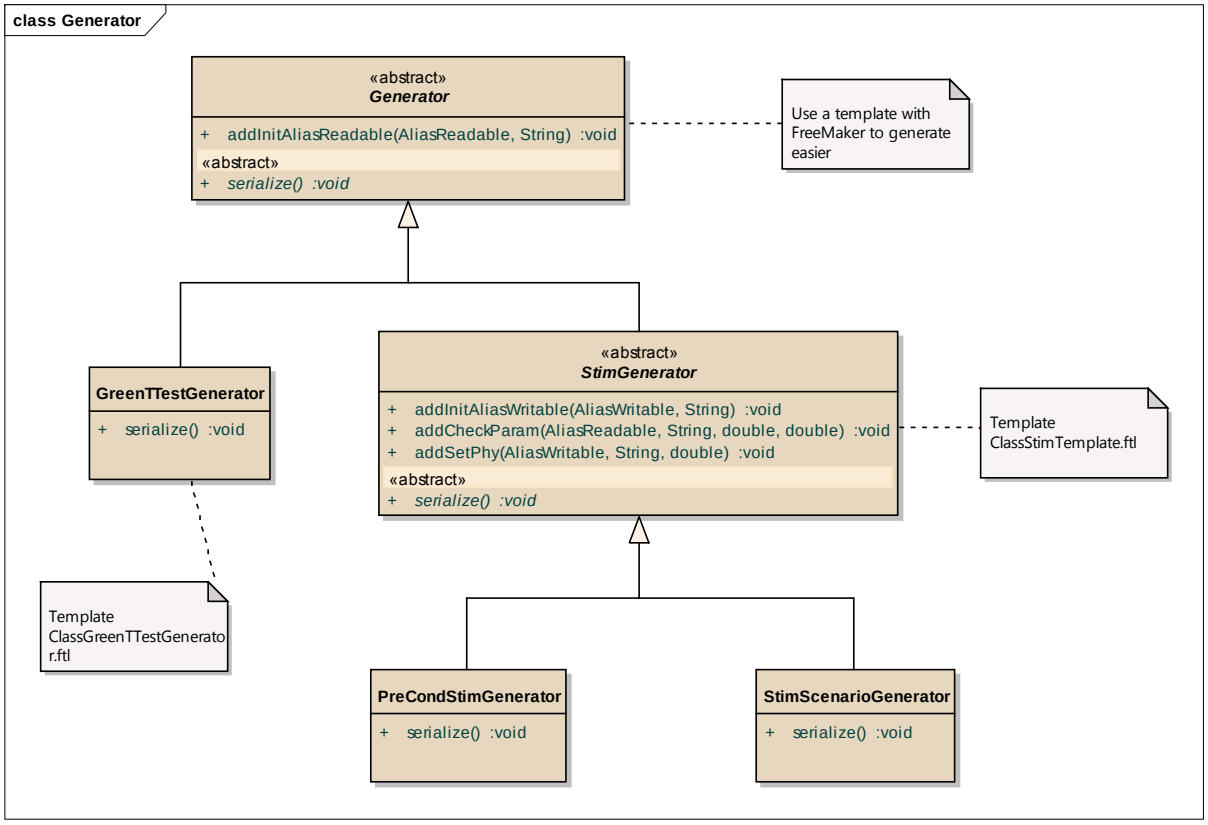
HIL_VS=0 signifie que nous allons mettre à 0 l'alias HIL_VS, il est possible de remplacer 0 par une expression tel que HIL_VS = (HIL_VS_OUT*2)^2 par exemple.

CHECK(HIL_VS_OUT,0,TOLRES(1)) ; permet de vérifier que HIL_VS_OUT est bien à 0.

Afin d'assurer le plus de souplesse possible, et que le développement des fichiers de tests soit simples et rapides pour l'utilisateur, le parser renvoie des erreurs en cas de problème de syntaxe, mais surtout la mauvaise utilisation des alias (Alias inconnu, AliasWritable demandant une lecture, AliasReadable demandant une écriture).
Cela permet d'éviter d'attendre la phase d'exécution pour que le problème soit remonté.

Générateur

Une fois le fichier parsé, il faut générer les classes correspondantes à une ligne du fichier excel : il est donc nécessaire de parser chaque ligne du fichier excel, et de générer les classes correspondantes, une fois ces classes générés, il faut les compiler et les mettre dans un fichier .jar. Un manager s'occupera d'assembler les différents tests en Bundle si c'est possible.



Afin de générer le plus simplement les fichiers, j'utilise un moteur de template : FreeMaker.

Actuellement, je suis en train de générer la partie d'un GreenTTest correspondant à une ExceptedBehavior.

Exemple de fichier généré

Joint à ce document, vous trouverez des exemples de fichiers générés (GreenTTest, StimScenario).