UFR 919 Informatique – Master Informatique

Spécialité STL – UE ILP

TME6 — Variations autour du JSR233

Christian Queinnec Benjamin Canou

1 JSR 223

La galaxie Java a introduit avec Java 5 une interface permettant d'embarquer un langage de script dans une JVM. Cette interface est spécifiée par le document JSR 223 en `http://jcp.org/aboutJava/communityprocess/final/jsr223/index.html` et l'environnement de base de Java 7 comporte d'ailleurs un évaluateur Javascript interne (à base de Rhino).

Les grandes lignes de cette spécification peuvent être lues dans le tutoriel `http://www.javaworld.com/javaworld/jw-04-2006/jw-0424-scripting.html`

Le langage de script peut ainsi utiliser les objets Java (ce que sait faire ILP3). On peut ainsi écrire certaines parties d'une application en ILP3 plutôt qu'en Java. La plupart des langages de scripts ayant une implantation en Java (Jython, Rhino, JRuby, Groovy, SISC, etc.) procurent cette interface.

Objectif : Permettre l'utilisation d'ILP3 comme langage de script au sein de Java conformément à la spécification JSR 223.

Buts :

— Être familiarisé avec la forme des spécifications JSR.
— Avoir étudié la compilation d'ILP vers JavaScript.
— Avoir identifié les interactions entre un langage de script et son langage hôte, en particulier le partage d'environnement.

2 Étapes

Votre production, sera stockée dans le paquetage `fr.upmc.ilp.ilp3tme6`.
Le travail sera décomposé en plusieurs étapes.

1. couplage de niveau 0 : lancer depuis Java l'interprétation d'un fichier ILP3 et récupérer la valeur.

2. couplage de niveau 1 : lancer depuis Java l'interprétation d'un programme ILP3 utilisant un environnement de variables globales créé par Java.
3. couplage de niveau 2 : lancer depuis Java la « compilation » d'un programme ILP3 en un AST puis lancer, plusieurs fois depuis Java, l'exécution de ce fichier compilé tout en fournissant un environnement de variables globales créé par Java.
4. couplage de niveau 3 : lancer depuis Java l'interprétation d'un programme ILP3 utilisant un environnement de variables globales liées à des fonctions de Java (des méthodes statiques comme `Math.sin`).

2.1 Interprétation d'un fichier

Ce travail peut s'incarner en seulement deux classes : `ScriptEngine` et `Process`.

La classe `ScriptEngine` est la machine permettant depuis Java d'évaluer (interpréter, compiler ou exécuter) des scripts sous forme de chaîne de caractères ou de flux de caractères (interface `Reader`).

Pour implanter `ScriptEngine`, on pourra s'aider de la classe `AbstractScriptEngine` (n'hésitez-pas au passage à parcourir les autres classes présentes dans le paquetage `javax.script`). Votre classe `ScriptEngine` utilisera votre classe `Process`, qui pourra étendre la classe `Process` existante, et servira d'intermédiaire entre la mécanique de `ScriptEngine` et celle d'ILP3.

Travail à réaliser : Le premier travail est de créer un `ScriptEngine` minimal qui sait interpréter un script contenu dans une chaîne de caractères ou un fichier. Écrivez ensuite un programme de test pour vérifier que l'interprétation des programmes en passant par votre `ScriptEngine` ILP1, ILP2 et ILP3 fournis fonctionne bien.

2.2 Environnement

Il est possible de lancer un script contenant des variables libres et de spécifier au `ScriptEngine`, en Java, la valeur de ces variables libres. Par exemple, le programme ILP3 : `(* 3 x)` doit retourner 12 si, en Java, on dit que `x` vaut 4.

De la même façon, une affectation globale dans un script ILP3 doit pouvoir être visible depuis Java. Par exemple, le programme ILP3 `(set! y 4)` doit permettre à Java d'extraire du contexte la valeur 4 pour la variable `y`.

Travail à réaliser : Améliorez votre solution pour qu'elle ait ce comportement. Il existe plusieurs possibilités, l'une d'entre elles est d'implanter une classe (par exemple du nom de `CommonPlusScriptContextAdapter`) permettant d'unifier les environnements d'ILP et du `ScriptEngine` (c'est-à-dire répondant aux interfaces `ICommon` et `ScriptContext`. Écrivez un programme de test reprenant l'exemple du paragraphe précédent.

2.3 Isolation

Une division par zéro en ILP3 provoque une exception. Il convient alors que `ScriptEngine` la récupére et la convertisse en une `ScriptException`.

Travail à réaliser : Améliorez votre `ScriptEngine` pour isoler de la sorte les exceptions d'ILP3.

2.4 Compilation

Il ne s'agira pas là de compilation vers C qui est un peu compliquée à mettre en œvre mais d'une compilation vers Javascript. À cet effet vous est donné le compilateur ILP vers Javascript (sujet d'un examen passé) dans `fr.upmc.ilp.ilp4.jsgen`. Rhino est un évaluateur Javascript inclus d'office dans Java 7 sous la forme d'un `ScriptEngine` conforme à la JSR 223. En compilant ILP vers Javascript puis en demandant à Rhino de le compiler en code-octet java vous aurez une compilation d'ILP.

Travail à réaliser : En suivant ce procédé, faites en sorte que votre `ScriptEngine` implante l'interface `Compilable`. Là encore, lancer l'exécution d'un programme compilé doit permettre de spécifier la valeur des variables libres ainsi que récupérer la valeur finale de ces variables libres.

2.5 Usage de méthode statique comme fonction

Cette question nécessite d'utiliser les aspects réflexifs de Java. On veut ajouter au `ScriptEngine` une méthode `wrapJavaPrimitive` permettant de rendre une méthode statique Java appelable depuis ILP3.

Par exemple, `se.wrapPrimitive("sinus",java.lang.Math.class,"sin");` permet de dire que `Math.sin` est une fonction primitive prédéfinie connue en ILP3 sous le nom de `sinus`.

Travail à réaliser : Faites en sorte que votre classe `ScriptEngine` implante l'interface `ScriptEngineWithWrapper` suivante :

```
1  package fr.upmc.ilp.ilp3tme6
2
3  import javax.script.ScriptException;
4
5  public interface ScriptEngineWithWrapper
6    extends javax.script.ScriptEngine {
7    public abstract void
8      wrapPrimitive(String ilpname, Class<?> cname, String mname)
9      throws ScriptException;
10 }
```