

Interopérabilité entre OCaml et Java

Béatrice Carré

8 mai 2014

Caractéristiques d'une interopérabilité efficace :

- accès à l'autre monde simple ou implicite
- bonne gestion mémoire et des exceptions
- gestion des caractéristiques de chaque monde

L'interopérabilité se fait sur le modèle objet de chacun

<i>caractéristiques</i>	<i>Java</i>	<i>OCaml</i>
accès champs	selon la visibilité	via appels de méthode
var./méth. statiques	✓	fonct./décl. globales
typage dynamique	✓	pas de downcast
héritage \equiv sous-typage ?	✓	×
surcharge	✓	×
héritage multiple	pour les interfaces	✓
modules paramétrés	×	✓

\Rightarrow Il faut réduire les possibilités d'un outil à l'intersection des deux mondes

O'Jacaré : schéma global

Pour un accès à des classes Java :

O'Jacaré génère leurs classes encapsulantes à partir d'un fichier dans lequel sont décrites les classes qu'on veut manipuler.

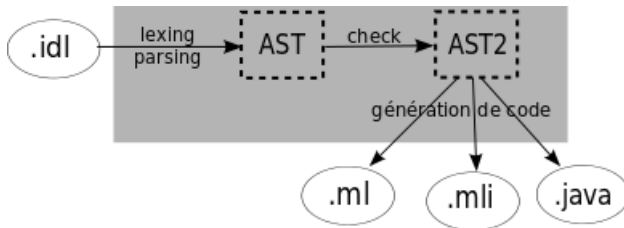
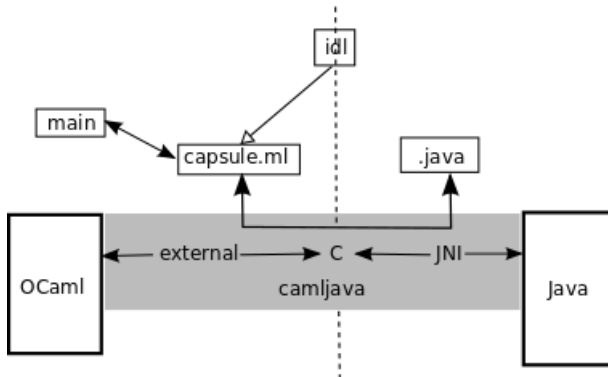


Figure: La génération de code d'O'Jacaré

O'Jacaré : schéma global

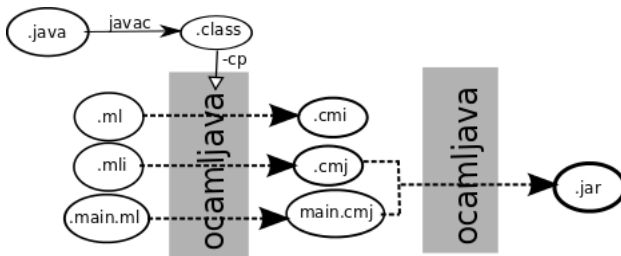
- La capsule générée permet à l'utilisateur de faire des appels transparents à des méthodes Java.
- Camljava gère la communication entre les deux mondes :
 - Recherche des classes par nom et des méthodes par signature
 - Conversion des types de base est assurée



O'Jacaré : exemple d'idl

```
package mypack;
class Point {
    int x;
    int y;
    [name default_point] <init> ();
    [name point] <init> (int ,int );
    void moveto(int ,int );
    string toString();
    boolean eq(Point);
}
interface Colored {
    string getColor();
    void setColor(string);
}
[callback] class ColoredPoint extends Point implements Colored {
    [name default_colored_point] <init> ();
    [name colored_point] <init> (int ,int ,string );
    [name eq_colored_point] boolean eq(ColoredPoint);
}
```

OCaml-Java : schéma global




Compilation vers du bytecode Java

⇒ un seul runtime Pas de problème de gestion mémoire, de communications .

OCaml-Java : l'accès au monde Java

```
make : 'a java_constructor -> 'a
call : 'a java_method -> 'a
get : 'a java_field_get -> 'a
set : 'a java_field_set -> 'a
is_null : 'a java_instance -> bool
instanceof : 'a java_type -> 'b java_instance -> bool
cast : 'a java_type -> 'b java_instance -> 'a
proxy : 'a java_proxy -> 'a
```

Accès à l'API Java et à du code utilisateur grâce à ce module.

 exempleOCaml-Java.png

Fusion des deux approches

<i>O'Jacaré+CamlJava</i>	<i>OCaml-Java</i>	<i>O'Jacaré+OCaml-Java</i>
appels transparents	via module Java	appels transparents
2 runtime	1 runtime	1 runtime
Pas d'allocation		
vérifications accès dans la capsule (avant accès Java)	vérifications accès par le compilateur (côté Java)	vérifications accès par le compilateur (côté Java)

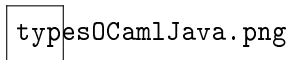


Figure: Les types dans OCamlJava

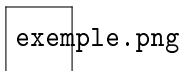


Figure: La génération du constructeur de Point

Ce nouvel outil a apporté des nouvelles possibilités d'un côté, avec une simplicité d'utilisation :

- Accès simple à l'API Java
- Accès utilisateur transparent grâce aux classes encapsulantes
- 1 seul runtime -> Gestion mémoire simplifiée et sûre
- Code généré simplifié (~ 5 fois moins)
-

<i>caractéristiques</i>	<i>O'Jacaré + OCaml-Java</i>
accès champs	selon la visibilité + via appels de méthode
var./méth. statiques	fonctions/décl. globales
héritage \equiv sous-typage ?	×
surcharge	×
héritage multiple	✓