

Interopérabilité entre OCaml et Java

Béatrice Carré

14 mai 2014

Definition

L'interopérabilité entre deux langages est la capacité d'un programme écrit dans un certain langage d'utiliser un programme dans un autre langage.

- Intérêt :
Tirer parti des spécificités de chaque langage
- Caractéristiques d'une interopérabilité efficace :
 - accès simple ou implicite à l'autre langage
 - gestion mémoire et des exceptions
 - bonne gestion des caractéristiques différentes

Comparaison des deux mondes

L'interopérabilité se fait sur le modèle objet de chacun, dont voici les caractéristiques qui diffèrent :

<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	✓	✗
surcharge	✓	✗
héritage multiple	pour les interfaces	✓

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

O'Jacaré : schéma global

Definition

Un **IDL** (Langage de définition d'Interface) définit une interface entre les deux langages, qui va permettre de les faire communiquer.

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

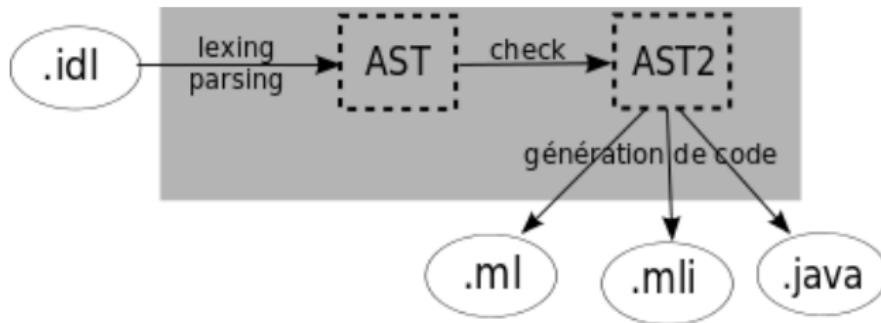


Figure: La génération de code d'O'Jacaré

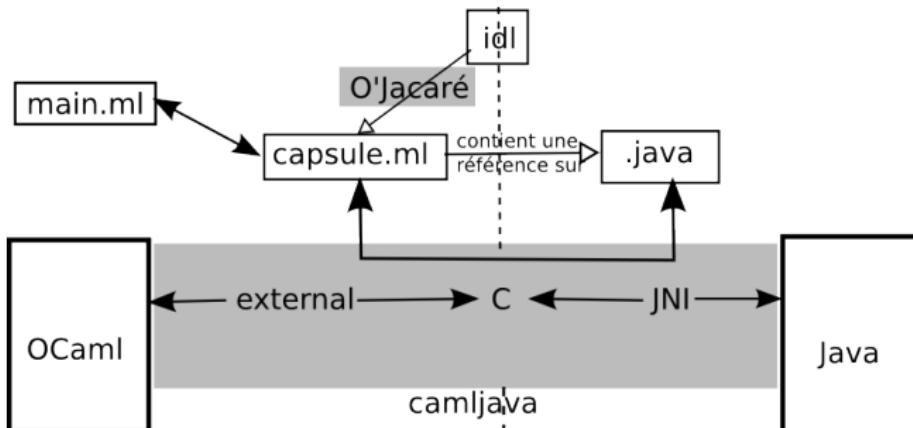
O'Jacaré : schéma global (2)

Definition

La **classe encapsulante** générée à partir de l'IDL contient une référence sur un objet Java et permet à l'utilisateur de faire les appels sur celui-ci.

La bibliothèque camljava gère la communication OCaml-Java :

- Recherche des classes par nom et des méthodes par signature
- La conversion des types de base est assurée



La classe Point (Java)

```
package mypack;
public class Point {
    int x;
    int y;
    public Point() {
        this.x = 0;
        this.y = 0;
    }
    public Point(int x,int y) {
        this.x = x;
        this.y = y;
    }
    public void moveTo(int x,int y){
        this.x = x;
        this.y = y;
    }
    public String toString() {
        return "("+x+","+y+ ")";
    }
    public double distance() {
        return Math.sqrt (this.x*this.x+this.y*this.y);
    }
    public boolean eq(Point p) {
        return this.x == p.x && this.y == p.y;
    }
}
```

O'Jacaré : définition de l'IDL

IDL : point.idl

```
package mypack;

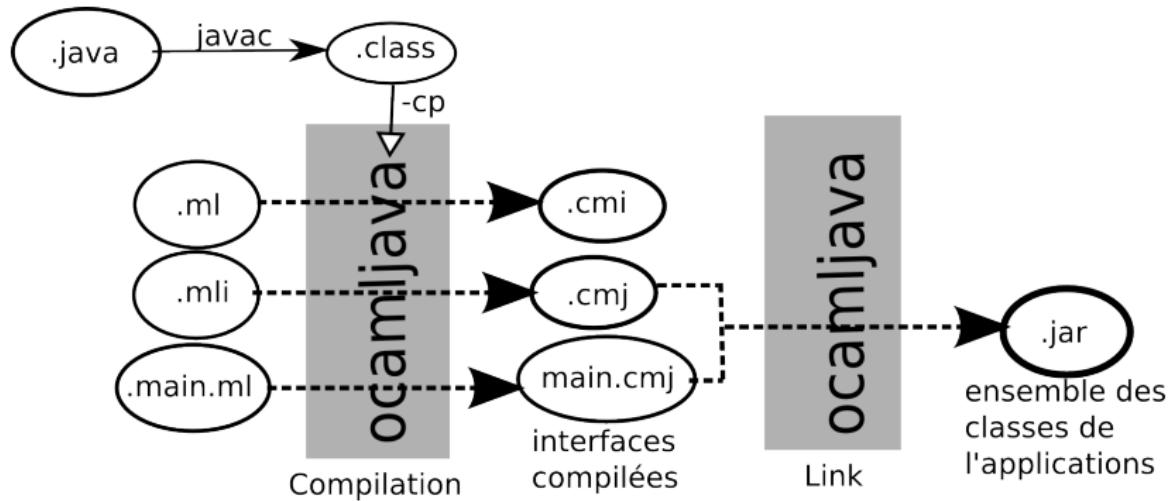
class Point {
    int x;
    int y;
    [name default_point] <init> ();
    [name point] <init> (int, int);
    void moveto(int, int);
    string toString();
    boolean eq(Point);
}
```

Utilisation : main.ml

```
open Point

let p = new default_point () in
let p2 = new point 1 1 in
p#moveto 4 3;
p#display ();
print_string (if (p#eq p2) then "true" else "false")
```

OCaml-Java : schéma global



Compilation vers du bytecode Java
⇒ un seul runtime

- Pas de problème de gestion mémoire,
- ni de communication dynamique

OCaml-Java : l'accès au monde Java

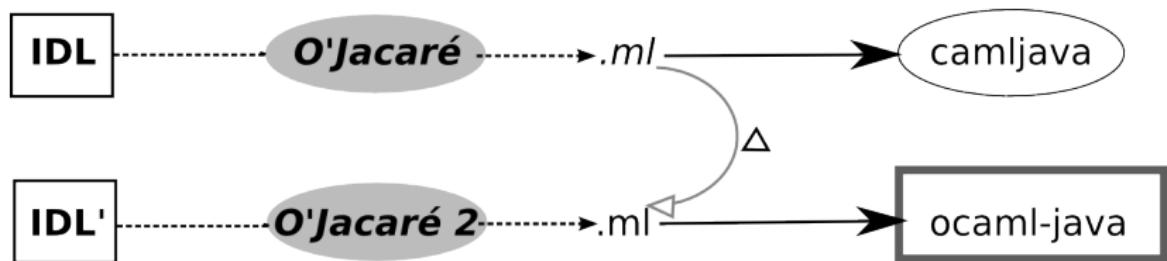
```
make : 'a java_constructor -> 'a
call : 'a java_method -> 'a
get : 'a java_field_get -> 'a
set : 'a java_field_set -> 'a
is_null : 'a java_instance -> bool
instanceof : 'a java_type -> 'b java_instance -> bool
cast : 'a java_type -> 'b java_instance -> 'a
```

Accès aux bibliothèques Java et à du code utilisateur grâce à ce module.

```
let color = JavaString.of_string "bleu"
and x = Int32.of_int 1
and y = Int32.of_int 2 in
let p = Java.make "mypack.ColoredPoint(int,int,java.lang.String)" s y
  color
in
  Java.call "mypack.Point.eq(mypack.Point):boolean" p p2
```

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



Adaptation d'Objacaré

TYPE IDL	type Java (java_type)	type OCaml pour OCaml-Java (oj_type t)	type OCaml (ml_type t)
<i>void</i>	void	unit	unit
<i>boolean</i>	boolean	bool	bool
<i>byte</i>	byte	int	int
<i>char</i>	char	int	char
<i>double</i>	double	float	float
<i>float</i>	float	float	float
<i>int</i>	int	int32	int
<i>long</i>	long	int64	int
<i>short</i>	short	int	int
<i>string</i>	java.lang.String	java'lang'String	string
<i>pack/Obj</i>	pack.Obj	pack'Obj	jObj

Figure: Les types dans OCamlJava

Comparaison de génération

```
let _init_point =
  let clazz = Jni.find_class "mypack/Point" in
  let id =
    try Jni.get_methodID clazz "<init>" "(II)V"
    with
    | _ ->
      failwith
        "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
         (int,int)\"."
  in
    fun (java_obj : _jni_jPoint) _p0 _p1 ->
      let _p1 = _p1 in
      let _p0 = _p0
      in
        Jni.call_nonvirtual_void_method java_obj clazz id
        [| Jni.Camlint _p0; Jni.Camlint _p1 |];
class point _p0 _p1 =
  let java_obj = _alloc_jPoint ()
  in let _ = _init_point java_obj _p0 _p1
  in object (self) inherit _capsule_jPoint java_obj end;;
class point _p0 _p1 =
  let _p1 = Int32.of_int _p1
  in let _p0 = Int32.of_int _p0
  in let java_obj = Java.make "mypack.Point(int,int)" _p0 _p1
  in object (self) inherit _capsule_jPoint java_obj end;;
```

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)
-