

# Interopérabilité entre OCaml et Java

Béatrice Carré

15 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.

- **Interopérabilité efficace :**

Accès simple à l'autre langage (ici, manipuler un objet Java comme un objet OCaml).

- **Projet :**

Adapter deux travaux déjà développés (*O'Jacaré* et *OCaml-Java*) pour tirer parti des avantages de chacun.

# Comparaison des deux mondes

Dans ce projet, 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	✓	✗
surcharge	✓	✗
héritage multiple	pour les interfaces	✓

## Definition

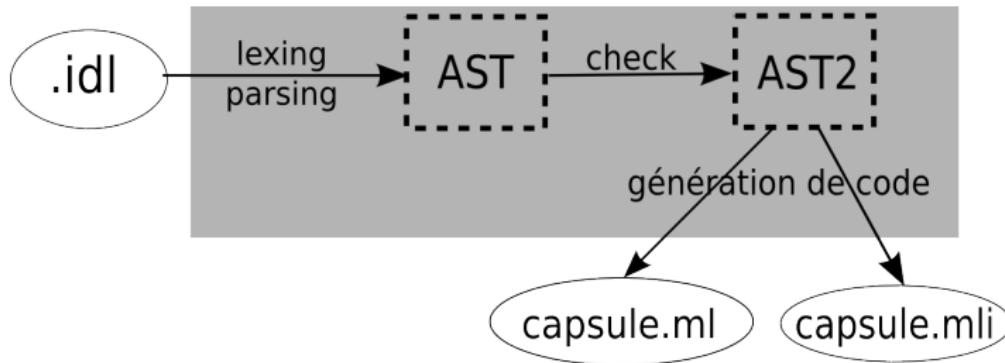
Une **interface** définit la frontière de communication entre deux entités.

Pour l'interopérabilité, il est nécessaire de définir une interface qui adapte les deux modèles.

## 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.



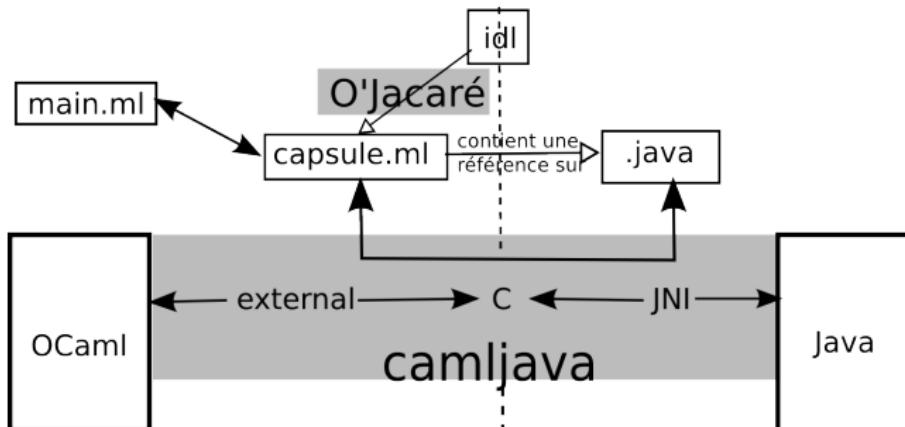
# O'Jacaré : schéma global (2)

## 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
- Conversion des types de base



# O'Jacaré exemple : 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 exemple : IDL et utilisation

## IDL : point.idl

```
package mypack;

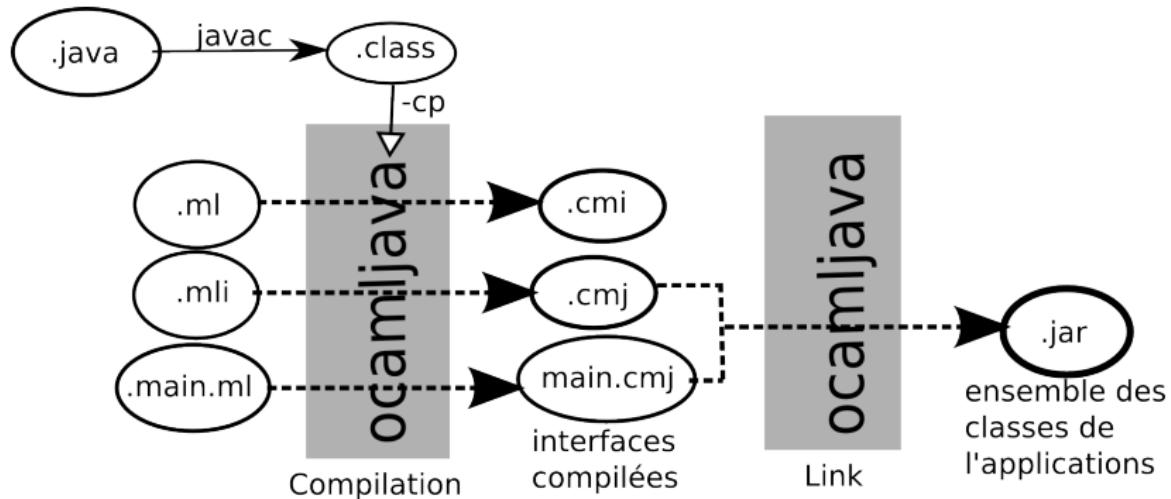
class Point {
    int x;
    int y;
    [name default_point] <init> ();
    [name point] <init> (int ,int );
    void moveto(int ,int );
    string toString();
    boolean eq(Point);
}
```

## Utilisation : main.ml

```
open Point

let p = new default_point () in
let p2 = new point 1 1 in
p#moveto 4 3;
p#toString ();
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
- Typage statique

# OCaml-Java : l'accès au monde Java

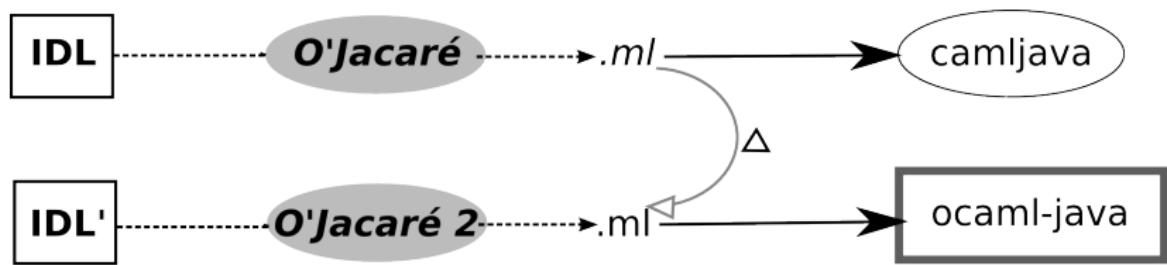
Accès à du code Java grâce à des méthodes définies dans le **module Java** OCaml pour OCaml-Java.

Exemple d'utilisation :

```
and x = Int32.of_int 1
and y = Int32.of_int 2 in
let (p : mypack'Point java_instance) =
  Java.make "mypack.Point()" ()
let (p2 : mypack'Point java_instance) =
  Java.make "mypack.Point(int,int)" x y
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 runtimes	1 runtime	1 runtime



# Adaptation d'Ojacaré : O'Jacaré 2

- ① Garder la même structure de capsule sauf l'introspection des types dynamiques, qui se fait statiquement par OCaml-Java
- ② Adapter au module *Java* de OCaml-Java
- ③ Redéfinir la conversion des types

<i>TYPE IDL</i>	<i>type Java (java_type)</i>	<i>type OCaml manipulés par OCaml-Java (oj_type t)</i>	<i>type OCaml (ml_type t)</i>
<i>int</i>	int	int32	int
<i>long</i>	long	int64	int
<i>string</i>	java.lang.String	java'lang'String	string
<i>pack/Obj</i>	pack.Obj	pack'Obj	jObj

# Comparaison de génération

## La génération du constructeur de Point par

- O'Jacaré

```
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;;
```

- O'Jacaré 2

```
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;;
```

## Travail :

- S'approprier les outils et lire les articles associés
- Comprendre le système de types de OCaml-Java
- Ecrire O'Jacaré 2

## O'Jacaré 2 :

- Accès simple à du code Java ou à l'API
- Accès utilisateur transparent grâce aux classes encapsulantes
- 1 seul runtime -> gestion mémoire plus sûre
- Code généré simplifié ( $\sim$  5 fois moins de lignes)

## Travaux futurs :

- Conversion des types tableaux
- Gérer un mécanisme de rappel (la redéfinition d'une méthode Java pas une méthode OCaml)

# Bibliographie

-  CHAILLOUX E., MANOURY P., PAGANO B., *Développement d'applications avec Objective Caml*, O'Reilly , 2000
-  HENRY G., *O'Jacaré* <http://www.pps.univ-paris-diderot.fr/~henry/ojacare/>
-  CLERC X., *OCaml-Java 2.0*  
<http://ocamljava.x9c.fr/preview/>
-  CHAILLOUX E., HENRY G., *O'Jacaré, une interface objet entre Objective Caml et Java*, 2004
-  CLERC X., *OCaml-Java : Typing Java Accesses from OCaml Programs*, Trends in Functional Programming, Lecture Notes in Computer Science Volume 7829, 2013
-  Leroy X., *The camljava project*,
-  CLERC X., *OCaml-java : module Java lien*

# Module Java de OCaml

<i>types OCaml-Java</i>	<i>descriptions et exemples</i>
java_instance	référence sur une instance Java
java_constructor	signature d'un constructeur "java.lang.Object()"
java_method	signature d'une méthode "java.lang.String.lastIndexOf(String) :int"
java_field_get	signature d'un attribut "mypack.Point.x :int"
java_field_set	signature d'un attribut "mypack.Point.x :int"
java_type	classe, interface ou type Array "java.lang.String"

```
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
```

# génération O'Jacaré 2 : la classe Point

```
type top = java'lang'Object java_instance;;
exception Null_object of string;;
type _jni_jPoint = mypack'Point java_instance;;
class type jPoint =
  object
    method _get_jni_jPoint : _jni_jPoint
    method set_x : int -> unit
    method get_x : unit -> int
    method set_y : int -> unit
    method get_y : unit -> int
    method moveto : int -> int -> unit
    method toString : unit -> string
    method eq : jPoint -> bool
  end;;
class _capsule_jPoint (jni_ref : _jni_jPoint) =
  let _ =
    if Java.is_null jni_ref then raise (Null_object "mypack/Point") else ()
  in
    object (self)
      method eq =
        fun (_p0 : jPoint) ->
          let _p0 = _p0#_get_jni_jPoint
          in Java.call "mypack.Point.eq(mypack.Point):boolean" jni_ref
              _p0
      method toString =
        fun () ->
          JavaString.to_string
            (Java.call "mypack.Point.toString():java.lang.String"
              jni_ref)
```

# génération O'Jacaré 2 : la classe Point (2)

```
method moveto =
  fun _p0 _p1 ->
    let _p1 = Int32.of_int _p1 in
    let _p0 = Int32.of_int _p0
    in Java.call "mypack.Point.moveto(int,int):void" jni_ref _p0
       _p1
method set_y =
  fun _p ->
    let _p = Int32.of_int _p
    in Java.set "mypack.Point.y:int" jni_ref _p
method get_y =
  fun () -> Int32.to_int (Java.get "mypack.Point.y:int" jni_ref)
method set_x =
  fun _p ->
    let _p = Int32.of_int _p
    in Java.set "mypack.Point.x:int" jni_ref _p
method get_x =
  fun () -> Int32.to_int (Java.get "mypack.Point.x:int" jni_ref)
method _get_jni_jPoint = jni_ref
end;;
let jPoint_of_top (o : top) : jPoint =
  new _capsule_jPoint (Java.cast "mypack.Point" o);;
let _instance_of_jPoint (o : top) = Java.instanceof "mypack.Point" o;;
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;;
class default_point () =
  let java_obj = Java.make "mypack.Point()" ()
  in object (self) inherit _capsule_jPoint java_obj end;;
```