

# L'interopérabilité entre OCaml et Java

Béatrice Carré  
beatrice.carre@etu.upmc.fr

Encadrants : Emmanuel Chailloux, Xavier Clerc et Grégoire Henry

9 mai 2014

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>1 L'interopérabilité entre OCaml et Java</b>	<b>4</b>
1.1 O'Jacaré, un générateur de code d'interface . . . . .	4
1.1.1 Principe global . . . . .	4
1.1.2 Génération de code . . . . .	4
1.1.3 Compilation avec la bibliothèque <code>camljava</code> . . . . .	5
1.2 <i>OCaml-Java</i> : compilation de code OCaml vers du bytecode Java . . . . .	6
1.2.1 Principe global . . . . .	6
1.2.2 Barrière d'abstraction : manipuler du Java . . . . .	7
1.3 Le travail à effectuer pour profiter des deux approches . . . . .	8
<b>2 Portage d'O'Jacaré pour <i>OCaml-Java</i> : <i>O'Jacaré 2</i></b>	<b>9</b>
2.1 Étude de la génération d'O'Jacaré . . . . .	9
2.1.1 La définition de l'IDL . . . . .	9
2.1.2 Analyse lexicale, syntaxique et sémantique . . . . .	9
2.1.3 génération de Java pour le callback . . . . .	9
2.1.4 Génération des classes encapsulante . . . . .	9
2.2 Génération de code pour Ocaml-Java . . . . .	10
2.2.1 Les types dans <i>OCaml-Java</i> . . . . .	10
2.2.2 schémas de compilation d'O'Jacaré 2 . . . . .	11
2.3 Comparaison d'O'Jacaré avec O'Jacaré 2 . . . . .	14
<b>3 Application</b>	<b>15</b>
<b>Conclusion</b>	<b>16</b>
<b>Bibliographie, références</b>	<b>17</b>
<b>4 Annexe</b>	<b>18</b>
4.1 grammaire de l'IDL d'O'Jacaré . . . . .	18
4.2 Génération de la classe Point par O'Jacaré . . . . .	19
4.3 Génération de la classe Point par O'Jacaré 2 . . . . .	22
4.4 Exemple d'application : l'affichage d'une base de donnée . . . . .	24

## Introduction

Il est utile de réutiliser dans un certain langage du code écrit dans un autre, sans avoir à le réécrire. Il arrive souvent de vouloir utiliser l'expressivité et l'élégance du langage Ocaml autant que le style objet de Java et la diversité de son API.

C'est pourquoi l'interopérabilité est un problème intéressant. Mais elle engendre beaucoup de questions sur la gestion de plusieurs éléments : la cohérence des types d'un langage à l'autre, la copie ou le partage des valeurs d'un monde à l'autre, le passage des exceptions, la gestion automatique de la mémoire (GC<sup>1</sup>), et des caractéristiques de programmation qui ne sont pas forcément gérées par les deux langages.

OCaml et Java comportent des différences entre leurs modèles objet, comme le montre le tableau ci-dessous, il donc est nécessaire de réduire l'étude à leur l'intersection.

<i>caractéristiques</i>	<i>Java</i>	<i>OCaml</i>
accès champs	selon la visibilité	via appels de méthode
variables/méthodes statiques	✓	fonctions/décl. globales
typage dynamique	✓	×
surcharge	✓	×
héritage multiple	seulement pour les interfaces	✓

Deux travaux ont déjà été réalisés pour l'interopérabilité entre OCaml et Java à travers leur modèle objet respectif :

- *O'Jacaré*[2] conserve les runtimes des deux langages (GC, Exceptions, ...) et les fait communiquer par l'interface `camljava`[8], avec l'aide de classes encapsulantes générées par O'Jacaré.
- *OCaml-java 2.0*[3] utilise un seul runtime, en compilant code le OCaml en bytecode Java. La manipulation des classes Java se fait à l'aide de nouveaux types introduits.

L'idée est de profiter des deux approches : d'une part, d'un accès simple à des classes définies, en générant grâce à O'Jacaré le code nécessaire à cet accès et profiter d'autre part de l'accès direct à toute l'API Java en ne gardant qu'un seul runtime, la JRE, grâce à *OCaml-Java*

Après l'étude des deux outils, le projet consiste à engendrer pour *ocaml-java* les fichiers d'encapsulation d'O'Jacaré. Ce portage est réalisé en OCaml étant donné qu'il reprend ce qui a déjà été développé pour O'Jacaré. Cette adaptation ne gère pas les appels de Java vers OCaml (*callback*<sup>2</sup>).

Dans ce rapport, nous décrivons le schéma global d'O'Jacaré, et d'Ocaml-Java pour en faire ressortir les avantages d'un portage d'O'Jacaré (O'Jacaré 2) pour *OCaml-Java*. Nous détaillons par la suite les modifications apportées à la génération d'interfaces, adaptée pour une encapsulation utilisable par le compilateur d'*OCaml-Java*. Pour finir, un exemple d'application sera présenté.

---

1. Garbage Collector

2. attribut représentant le sens d'appel de Java vers OCaml

# 1 L'interopérabilité entre OCaml et Java

## 1.1 O'Jacaré, un générateur de code d'interface

### 1.1.1 Principe global

O'Jacaré génère le code nécessaire à l'encapsulation des classes définies dans un IDL<sup>3</sup>, pour permettre aux interfaces avec C de chacun des deux langages de communiquer.

Lorsqu'on parle d'une classe encapsulante (capsule) d'une classe Java, on parle d'une classe OCaml qui porte une référence sur l'objet Java en question, et qui est chargée de faire les opérations sur celui-ci.

L'appel à des classes et méthodes Java est alors possible en appelant les méthodes de la capsule générée, qui va gérer l'appel aux classes Java par le biais de l'interface `camljava`. Le code Java généré pour le callback va permettre avec le même principe, les appels dans l'autre sens.

`camljava` est une interface bas-niveau basée sur les interfaces de chaque langage avec C : la JNI<sup>4</sup> et *External*.

La génération de code se fait en plusieurs passes :

- analyse lexicale et analyse syntaxique de l'IDL donnant un AST.
- vérification des types de l'AST, donnant un nouvel arbre CAST<sup>5</sup>.
- la génération des fichiers Java nécessaires pour un appel callback.
- la génération à partir du CAST des classes encapsulantes dans un fichier `.ml`
- la génération à partir du CAST du module `.mli` adapté

Les deux dernières étapes seront présentées plus en profondeur dans la section 2.1. Un schéma décrit ces étapes dans la figure 2.

### 1.1.2 Génération de code

La génération de code se fait à partir d'un IDL, dont la grammaire (BNF<sup>6</sup>) est définie en annexe dans la section 4.1. Dans cet IDL, nous pouvons définir des déclarations qui sont à l'intersection de ce qui est accessible dans les modèles objet de chaque langage.

Le but de cet IDL est de définir la signature des classes Java déjà définies, que nous voulons manipuler du côté OCaml. Ces classes encapsulantes servant à faire le lien entre les classes/interfaces de chaque côté, il n'est donc nécessaire de définir dans l'IDL que les méthodes que nous voulons appeler depuis OCaml.

Voici un exemple de déclarations dans un IDL, si on veut utiliser la classe `Point` définie à gauche. Il n'est nécessaire de définir dans l'IDL uniquement les méthodes ou attributs que l'on veut manipuler depuis OCaml.

---

3. Langage de définition d'interface

4. Java Native Interface

5. Pour *Checked AST*

6. Backus-Naur Form

Point.java

```

1 package mypack;
2 public class Point {
3     int x;
4     int y;
5     public Point() {
6         this.x = 0;
7         this.y = 0;
8     }
9     public Point(int x, int y) {
10        this.x = x;
11        this.y = y;
12    }
13    public void moveto(int x, int y) {
14        this.x = x;
15        this.y = y;
16    }
17    public String toString() {
18        return "(" + x + ", " + y + ")";
19    }
20    public double distance() {
21        return Math.sqrt
22            (this.x*this.x+this.y*this.y);
23    }
24    public boolean eq(Point p) {
25        return this.x == p.x
26            && this.y == p.y;
27    }
28 }

```

point.idl

```

1 package mypack;
2 class Point {
3     int x;
4     int y;
5     [name default_point] <init> ();
6     [name point] <init> (int , int);
7     void moveto(int , int);
8     string toString();
9     boolean eq(Point);
10 }

```

Pour une déclaration de classe ou interface, la génération de code donne :

- Un type abstrait correspondant au type Java
- Un type classe t
- Une classe encapsulante C de type t
- 1 à n classes Ci, sous-classes de C (une par constructeur), 0 si c'est une interface
- Une fonction instanceof pour ce type
- Une fonction de cast pour ce type

Vous trouverez en annexe le code du fichier généré par ces déclarations.

Les fichiers générés sont destinés à être compilés avec l'aide la bibliothèque `camljava`.

### 1.1.3 Compilation avec la bibliothèque `camljava`

`camljava` gère l'interfacage entre OCaml et Java via C.

Un exemple d'utilisation de cette bibliothèque est :

```

1 let clazz = Jni.find_class "mypack/Point" in
2 let id = Jni.get_methodID clazz "<init>" "(II)V" in
3 let java_obj = Jni.alloc_object clazz : Jni.obj in
4 Jni.call_nonvirtual_void_method java_obj clazz id
5 [| Jni.Camlint 1; Jni.Camlint 2 |];;

```

Les références sur les objets Java correspondent à un type abstrait en OCaml, dans lequel des opérations donnent accès à des méthodes ou à des champs de cet objet Java.

L'exécution se fait dans les 2 runtimes, qui peuvent alors communiquer.

La gestion des exceptions est faite par encapsulation aussi, et la gestion de la mémoire se fait par une mise en racine de l'objet dans la mémoire l'autre monde avant de le passer en référence.

Mais cette gestion avec deux Garbage Collector et deux ensembles de racines pour la mémoire reste incertaine, dans la mesure où du point de vue mémoire tout objet Java alloué en O'Caml est une racine du GC de Java. Quand O'Caml ne l'utilise plus, la racine Java est supprimée mais l'objet peut être conservé par le GC de Java s'il est référencé par un autre objet encore vivant. On retrouve donc les problèmes inhérents aux compteurs de références pour les structures circulaires.

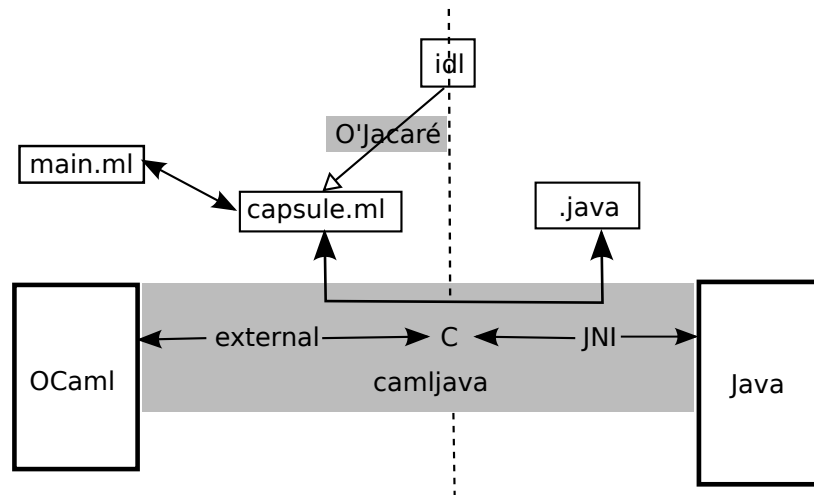


FIGURE 1 – La communication grâce à `camljava`

## 1.2 *OCaml-Java* : compilation de code OCaml vers du bytecode Java

### 1.2.1 Principe global

*OCaml-Java* est un compilateur, générant du code octet Java (`.jar`) à partir d'un programme OCaml. Ce processus s'effectue en deux phases :

1. La compilation vers du code intermédiaire
2. La résolution dynamique pour produire un exécutable pour la JVM<sup>7</sup>

Il est naturellement possible d'utiliser des bibliothèques Java en utilisant la barrière d'abstraction d'*OCaml-Java*.

---

7. Java Virtual Machine

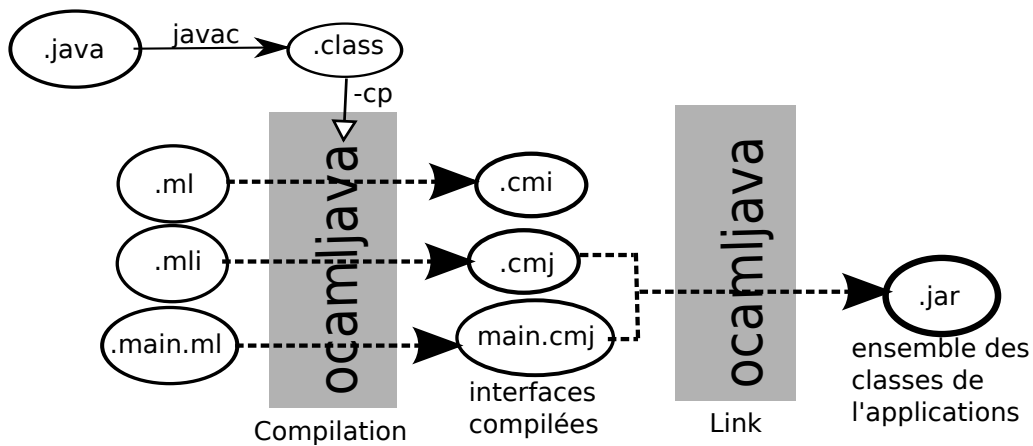


FIGURE 2 – Schéma global du compilateur d'*OCaml-Java*

### 1.2.2 Barrière d'abstraction : manipuler du Java

Description des types manipulés par *OCaml-Java* permettant un accès au monde de Java depuis celui d'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"

et les méthodes du module Java[9] pour OCaml

```

1 make : 'a java_constructor -> 'a
2 call : 'a java_method -> 'a
3 get : 'a java_field_get -> 'a
4 set : 'a java_field_set -> 'a
5 is_null : 'a java_instance -> bool
6 instanceof : 'a java_type -> 'b java_instance -> bool
7 cast : 'a java_type -> 'b java_instance -> 'a
8 proxy : 'a java_proxy -> 'a

```

Une exception est aussi définie pour permettre d'attraper les exceptions du côté OCaml :

```

1 exception Java_exception of java 'lang' Throwable java_instance

```

Voici un exemple d'utilisation du module Java [9] d'OCaml, voué à être compilé avec *OCaml-Java* :

```

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

```

Ce compilateur apporte une interopérabilité sûre par le fait qu'il amène à une exécution sur un seul runtime et qu'il encadre et vérifie les accès à Java. Mais sa syntaxe est assez verbeuse, donc son utilisation moyennement accessible.

### 1.3 Le travail à effectuer pour profiter des deux approches

O'Jacaré construit les classes encapsulantes de classes java définies par l'utilisateur, et permet ainsi l'accès aux méthodes (d'instance ou de classe) Java en OCaml en passant par l'interface de bas niveau `camljava`.

*OCaml-Java* permet l'accès simple à toute l'API Java depuis OCaml ou toute autre classe définie, de manière sûre.

La question à se poser est maintenant est : comment modifier la génération d'O'Jacaré pour obtenir les classes d'encapsulation adaptées pour *OCaml-Java*. Sur le schéma ci-dessous, cette modification est représentée par  $\Delta$ .

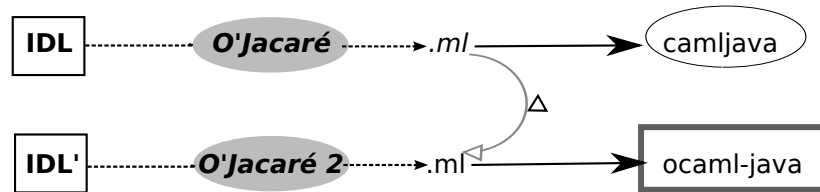


FIGURE 3 – Le travail à effectuer

La génération de code va être basée sur le même principe de classes encapsulantes pour garder l'avantage d'un appel largement simplifié vers Java, mais en utilisant désormais *OCaml-Java* comme outil de communication avec Java.



## 2 Portage d'O'Jacaré pour *OCaml-Java* : *O'Jacaré 2*

### 2.1 Étude de la génération d'O'Jacaré

#### 2.1.1 La définition de l'IDL

Les deux modèles objets étant différents, l'interface entre OCaml et Java doit être réduite à l'intersection de ces modèles pour définir l'IDL.

La syntaxe du langage d'interface est donné en annexe, en utilisant la notation BNF.

O'Jacaré 2 utilisera ce même IDL, en retirant de sa BNF l'attribut *callback*, l'autre sens de communication n'étant pas géré dans O'Jacaré 2.

#### 2.1.2 Analyse lexicale, syntaxique et sémantique

La première phase est celle d'analyse lexicale et syntaxique, séparant l'IDL en lexèmes et construisant un AST<sup>8</sup>, structurant les déclarations de l'IDL.

Vient ensuite la phase d'analyse sémantique, analysant l'AST obtenu par la phase précédente, vérifiant si l'IDL est correct sémantiquement, restructurant chaque classe ou interface définie dans l'IDL en construisant un nouvel AST (CAST) qui va être manipulé dans les passes de génération de code.

Dans O'Jacaré 2, cette phase est différente uniquement dans le fait qu'on n'accepte plus l'attribut *callback* dans l'IDL.

#### 2.1.3 génération de Java pour le callback

O'Jacaré permet la génération de classes Java qui encapsulent une classe OCaml, et permet ainsi un appel dans l'autre sens (de Java vers OCaml), grâce à un attribut "callback" ajouté devant une classe définie dans l'IDL.

Mais O'Jacaré 2 ne gérant pas ce sens de communication, nous ne nous intéresserons pas à cette génération de code.

#### 2.1.4 Génération des classes encapsulante

Cette phase est la plus largement modifiée pour notre adaptation.

À partir de la structure du fichier généré par O'Jacaré, nous avons d'abord étudié l'utilité et les éléments à modifier pour l'adapter pour *OCaml-Java* :

---

8. *Abstract syntax tree* ou arbre syntaxique abstrait

<i>élément</i>	<i>descriptions supplémentaires</i>	<i>modifications à faire pour OCaml-Java</i>
Type abstrait jni	référence sur l'objet Java	de type java_instance
Class type t	type objet respectant le type de la classe Java	similaires
Cast JNI (up et down)	utile pour le cast utilisateur	inutile
Fonction d'allocation	alloue la référence Java du côté OCaml	inutile : <i>OCaml-Java</i> n'utilise que le runtime Java
Capsule	vérifications avant l'appel vers Java	inutile : les vérifications se font lors de la compilation
Capsule	arguments des méthodes	conversion des types
Downcast utilisateur	cast de top vers t	simplifié grâce à la fonction Java.cast
Instance_of utilisateur	teste si un objet est une instance de la classe Java	simplifié grâce à la fonction Java.instanceof
Fonction d'initialisation	fonctions intermédiaire pour l'initialisation	inutiles
Classe de construction	crée une instance de la capsule	simplifiée car pas d'allocation
fonctions/méthodes statiques	transformation en fonctions/méthodes globales	similaires

Nous savons que *OCaml-Java* s'assure statiquement de l'existence des classes et méthodes appelées, nous pouvons être sûrs que beaucoup de fonctions seront inutiles pour celui-ci.

Après cela, il a fallu partir des fichiers générés d'O'Jacaré, et faire une ébauche à la main (en supprimant l'inutile et en adaptant les appels vers Java).

Puis à partir de l'ébauche écrite, nous avons fait ressortir les schémas de compilation correspondants à chaque type de déclaration de l'IDL.

## 2.2 Génération de code pour Ocaml-Java

### 2.2.1 Les types dans *OCaml-Java*

Pour préparer ces schémas de compilation, nous avons écrit le tableau représentant les équivalents en OCaml des types Java manipulés par *OCaml-Java*. À partir de celui-ci, la phase suivante était de définir les conversions à faire pour l'adaptation sous forme de schémas de compilation.

La troisième colonne représente les types manipulés par le programme OCaml écrit par l'utilisateur d'O'Jacaré2, la troisième ce qu'attend le compilateur *OCaml-Java*, tout cela en fonction du type correspondant au type Java. Le problème est donc de convertir du deuxième *oj\_type* au *ml\_type* type pour la manipulation côté OCaml et du *ml\_type* au *oj\_type* lors d'un appel à une fonction du module Java (un appel, un constructeur ou autre).

<i>TYPE IDL</i>	<i>type Java</i> (java_type)	<i>type OCaml pour OCaml-Java</i> (oj_type t)	<i>type OCaml</i> (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 java_instance	string
<i>pack/Obj</i>	pack.Obj	pack'Obj java_instance	jObj

FIGURE 4 – Tableau associant pour chaque type de l'IDL les fonctions de conversion utiles aux schémas de compilation manipulant ceux-ci, comme explicité ci-dessus.

Tableau associant pour chaque type de l'IDL les fonctions de conversion utiles aux schémas de compilation manipulant ceux-ci, comme explicité ci-dessus.

<i>TYPE IDL</i>	<i>to_oj_Type ARGi</i>	<i>to_ml_type ARGi</i>	<i>fcast</i>
<i>int</i>	Int32.of_int	Int32.to_int	
<i>long</i>	Int64.of_int	Int64.to_int	
<i>string</i>	JavaString.of_string	JavaString.to_string	
<i>pack/Obj</i>	_pi#_get_jni_jObj	(new _capsule_jObj ... : jObj)	(_pi : jObj)

## 2.2.2 schémas de compilation d'O'Jacaré 2

Nous considérons un environnement contenant les variables suivantes, initialisées à leur valeur par défaut :

- $\rho$  = "" : le nom du **package** où trouver les classes définies.
- $\Lambda$  = "" : le **nom de la classe** courant.
- $\gamma$  = false : si la déclaration est une **interface**.
- $\delta$  = "" : la classe dont **hérite** la classe courante.
- $\Delta$  = [] : les interfaces qu'**implemente** la classe courante.

### class

Le type "top" manipulé est le type d'une instance de Ocaml-Java de la classe Objet Java :

```
1 type top = java'lang'Object java_instance;;
```

Une exception a été définie, qui permettra de vérifier dynamiquement si l'objet Java passé en référence est nul ou non, en attrapant une exception Java du côté OCaml :

```
1 exception Null_object of string
```

**class**

```

[[class CLASS extends E implements I1, I2...{
    < static > attr1; < static > attr2; ...;
    < static > m1; < static > m2; ...;
    init1; init2; ...;
}] $\rho, \Lambda, \gamma, \delta, \Delta \longrightarrow$ 

(** type 'a java_instance*)
"type _jni_jCLASS = PACK'CLASS java_instance;; "

(** classe encapsulante *)
" class type jCLASS =
  object inherit E
    inherits jI1
    inherits jI2  ... "

  eval_classe[[attr1; attr2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 
  eval_classe[[m1; m2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 

  " method _get_jni_jCLASS : _jni_jCLASS
  end"

(* capsule wrapper *)
" class _capsule_jCLASS =
  fun (jni_ref : _jni_jCLASS) ->
    let _ =
      if Java.is_null jni_ref
      then raise (Null_object \ "PACK/POINT\ ")
      else ()
    in
      object (self) "

  eval_capsule[[attr1; attr2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 
  eval_capsule[[m1; m2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 

  "method _get_jni_jI1 = (jni_ref :> _jni_jI1)
    method _get_jni_jI2 = (jni_ref :> _jni_jI2) ...
    method _get_jni_jE = (jni_ref :> _jni_jE)
    method _get_jni_jCLASS = jni_ref
  end"

(* downcast utilisateur *)
"let jCLASS_of_top (o : top) : jCLASS =
  new _capsule_jCLASS (Java.cast \ "PACK.CLASS\ " o)
(* instance_of *)
let _instance_of_jCLASS (o : top) =
  Java.instanceof \ "PACK.CLASS\ " o;;

eval[[init1; init2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 
eval[[static m1; static m2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 
eval[[static attr1; static attr2; ...]] $\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...$ 

```

### methodes

$eval\_classe \llbracket RTYPE \text{ METH } (TARG1, TARG2, \dots) \rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2\ldots} \longrightarrow$

```
" method METH : "(ml_type TARG1) -> (ml_type TARG2) -> ... -> (ml_type RTYPE)
```

$eval\_capsule \llbracket RTYPE \text{ METH } (TARG1, TARG2, \dots) \rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2\ldots} \longrightarrow$

```
"      method METH =
        fun "(fcast TARG0) (fcast TARG1) ..." ->
          let _p1 = "(to_obj_type TARG1)" _p1 in
          let _p0 = "(to_obj_type TARG0)" _p0
          in "
            (to_ml_type RTYPE)
            "Java.call \"PACK.CLASS.METH(\"(javaType TARG1),(javaType TARG2
              ),...):(javaType RTYPE)\" \" jni_ref _p0 _p1 ..."
```

### attributs

$eval\_classe \llbracket TYPE \text{ ATTR}; \rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2\ldots} \longrightarrow$

```
"method set_ATTR : "(ml_type TYPE)" -> unit
method get_ATTR : unit -> "(ml_type TYPE)
```

$eval\_classe \llbracket TYPE \text{ ATTR}; \rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2\ldots} \longrightarrow$

```
"method set_ATTR =
  fun "(fcast TYPE)" ->
    let _p = "(to_obj_type TYPE)" _p
    in Java.set \"PACK.CLASS.ATTR:TYPE\" \" jni_ref _p
method get_ATTR =
  fun () ->
    \"(to_ml_type TYPE)\" (Java.get \"PACK.CLASS.ATTR:TYPE\" \" jni_ref)\"
```

### inits

$eval \llbracket [name \text{ INIT}] <init> (TARG0, TARG1, \dots) \rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2\ldots} \longrightarrow$

```
"class INIT _p0 _p1 ... =
  let _p1 = "(to_obj_type TARG1)" in
  let _p0 = "(to_obj_type TARG2)" in
  let java_obj = Java.make \"PACK.CLASS(\"(javaType
    TARG0),(javaType TARG1),...)\": _p0 _p1
  in
  object (self)
    inherit _capsule_jCLASS java_obj
  end;;"
```

### méthodes statiques

$eval\llbracket staticR\!TYPE\ METH\ (TARG1, TARG2, ..)\rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=false, \delta=E, \Delta=I1, I2...}$

```
"let PACK_CLASS__ml =
  fun "(fcast TARG0) (fcast TARG1) ..." ->
    let _p1 = "(to_oj_type TARG1)" _p1 in
    let _p0 = "(to_oj_type TARG0)" _p0
    in "
      (to_ml_type RTYPE)
      "Java.call \"PACK.CLASS.METH(\"(javaType TARG1),(javaType TARG2
        ),...):(javaType RTYPE)\" \" _p0 _p1 ..."
```

### attributs statiques

$eval\llbracket static\!TYPE\ ATTR;\rrbracket_{\rho=PACK, \Lambda=CLASS, \gamma=, \delta=E, \Delta=I1, I2...} \longrightarrow$

```
"let PACK_CLASS__set_ATTR =
  fun "(fcast TYPE)" ->
    let _p = "(to_oj_type TYPE)" _p
    in Java.set \"PACK.CLASS.ATTR:TYPE\" () _p
let PACK_CLASS__get_ATTR =
  fun () ->
    "(to_ml_type TYPE)" (Java.get \"PACK.CLASS.ATTR:TYPE\" () )"
```

## 2.3 Comparaison d'O'Jacaré avec O'Jacaré 2

Ici nous présentons un exemple de morceaux de code généré

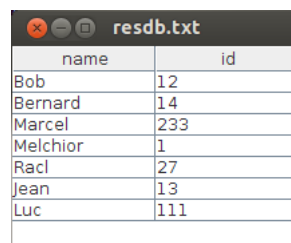
```
1 let _init_point =
2   let clazz = Jni.find_class "mypack/Point" in
3   let id =
4     try Jni.get_methodID clazz "<init>" "(II)V"
5     with
6       | - ->
7         failwith
8           "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
              (int,int)\"."
9   in
10    fun (java_obj : _jni_jPoint) _p0 _p1 ->
11      let _p1 = _p1 in
12      let _p0 = _p0
13      in
14        Jni.call_nonvirtual_void_method java_obj clazz id
15        [| Jni.Camlint _p0; Jni.Camlint _p1 |];;
16 class point _p0 _p1 =
17   let java_obj = _alloc_jPoint ()
18   in let _ = _init_point java_obj _p0 _p1
19   in object (self) inherit _capsule_jPoint java_obj end;;
```

```
1 class point _p0 _p1 =
2   let _p1 = Int32.of_int _p1
3   in let _p0 = Int32.of_int _p0
4   in let java_obj = Java.make "mypack.Point(int,int)" _p0 _p1
5   in object (self) inherit _capsule_jPoint java_obj end;;
```

### 3 Application

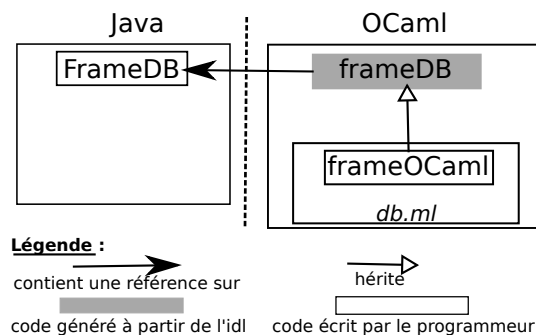
En développant ce projet, une application concrète m'est venue à l'esprit : un mini-projet de gestion de base de données implémenté en OCaml au cours de mes études manquait d'une interface graphique. Or, la bibliothèque graphique *swing* offre un composant permettant d'afficher un tableau (JTable), très simple d'utilisation.

J'ai donc d'un côté repris ce projet en ajoutant une fonction créant un JTable en lui donnant les données de la base de données créée, et implémenté une classe Java JTable construisant le composant en question. En quelques lignes de code, j'ai ajouté une interface graphique Java à mon programme OCaml de gestion de base de données.



name	id
Bob	12
Bernard	14
Marcel	233
Melchior	1
Racl	27
Jean	13
Luc	111

La structure de ce programme est la suivante :



Le détail du code est en annexe dans la section 4.4

## Conclusion

Le but de ce projet était de créer un outil simple d'utilisation et efficace pour l'interopérabilité entre OCaml et Java. *O'Jacaré* et *OCaml-Java* respectent chacune des ces deux idées : nous avons repris la simplicité d'utilisation d'*O'Jacaré* et l'efficacité de compilation d'*OCaml-Java*.

Nous avons adapté *O'Jacaré* afin qu'il prenne pour cible *OCaml-Java*. *O'Jacaré 2* respecte le contrat dans la mesure où l'utilisateur n'a qu'à définir un IDL simple, pour pouvoir utiliser du code Java dans son programme OCaml. De plus, la compilation se fait de manière sûre grâce à un outil qui assure des appels cohérents de manière statique, et en n'utilisant qu'un seul environnement d'exécution.

L'interopérabilité a tout de même des limites du fait que deux modèles de programmations différents ne peuvent communiquer que sur ce qu'ils ont en commun, mais il est certain que de futures applications profiteront de cette complémentarité entre les mondes Java et OCaml.

Enfin, en ajoutant à *O'Jacaré 2* la possibilité d'assurer la communication de Java vers OCaml, nous pourrions dire qu'il sera complet. Dans cette continuité, un stage à INRIA sera effectué afin de parfaire cette communication de Java vers OCaml.



## Bibliographie, références

### Références

- [1] CHAILLOUX E., MANOURY P., PAGANO B., *Développement d'applications avec Objective Caml*, O'Reilly , 2000, (<http://www.oreilly.fr/catalogue/ocaml.html>)
- [2] HENRY G., *O'Jacaré* <http://www.pps.univ-paris-diderot.fr/~henry/ojacare/>
- [3] CLERC X., *OCaml-Java 2.0* <http://ocamljava.x9c.fr/preview/>
- [4] CHAILLOUX E., HENRY G., *O'Jacaré, une interface objet entre Objective Caml et Java*, 2004
- [5] CLERC X., *OCaml-Java : Typing Java Accesses from OCaml Programs*, Trends in Functional Programming, Lecture Notes in Computer Science Volume 7829, 2013, lien
- [6] CLERC X., *OCaml-Java : OCaml on the JVM*, Trends in Functional Programming, 2012, lien
- [7] CLERC X., *OCaml-Java : OCaml-Java : from OCaml sources to Java bytecodes* , Trends in Functional Programming, 2012, <http://www.lexifi.com/ml2012/full19.pdf>
- [8] Leroy X., *The camljava project*, (<http://forge.ocamlcore.org/projects/camljava/>)
- [9] CLERC X., *OCaml-java : module Java* <http://ocamljava.x9c.fr/preview/javalib/index.html>
- [10] CamlP4 <http://pauillac.inria.fr/camlp4/>

## 4 Annexe

### 4.1 grammaire de l'IDL d'O'Jacaré

*Les symboles < et > encadrent des règles optionnelles, les terminaux sont en bleu, et les non-terminaux sont en italique.*

```
file ::= package <package>*
      | decl <decl>*

package ::= package qname ; decl <decl>*

decl ::= class
      | interface

class ::= <[attributes]> <abstract> class name
      < extends qname >
      < implements qname <, qname>* >
      { <class_elt ;>* }

class_elt ::= <[ attributes ]> <static> <final> type name
            | <[ attributes ]> <static> <abstract> type name (<args>)
            | [ attributes ] <init> (<args>)

interface ::= <[ attributes ]> interface name
            < extends qname <, qname>* >
            { <interface_elt ;>* }

interface_elt ::=
    <[ attributes ]> type name
    | <[ attributes ]> type name (<args>)

args ::= arg <, arg>*
arg ::= <[ attributes ]> type <name>

attributes ::= attribute <, attribute>*
attribute ::= name ident
            | callback
            | array

type ::= basetype
      | object
      | basetype [ ]

basetype ::= void
          | boolean
          | byte
          | char
          | short
          | int
          | long
          | float
          | double
          | string

object ::= qname
qname ::= name <.name>*
name ::= ident
```

## 4.2 Génération de la classe Point par O'Jacaré

```
1 type _jni_jPoint = Jni.obj;;
2 class type jPoint =
3   object
4     inherit JniHierarchy.top
5     method _get_jni_jPoint : _jni_jPoint
6     method set_x : int -> unit
7     method get_x : unit -> int
8     method set_y : int -> unit
9     method get_y : unit -> int
10    method moveto : int -> int -> unit
11    method toString : unit -> string
12    method eq : jPoint -> bool
13  end;;
14 let __jni_obj_of_jni_jPoint (java_obj : _jni_jPoint) =
15   (Obj.magic : _jni_jPoint -> Jni.obj) java_obj;;
16 let __jni_jPoint_of_jni_obj =
17   let clazz =
18     try Jni.find_class "mypack/Point"
19     with | _ -> failwith "Class not found : mypack.Point."
20   in
21   fun (java_obj : Jni.obj) ->
22     if not (Jni.is_instance_of java_obj clazz)
23     then failwith "'cast error' : jPoint (mypack/Point)"
24     else (Obj.magic java_obj : _jni_jPoint);;
25 let _alloc_jPoint =
26   let clazz = Jni.find_class "mypack/Point"
27   in fun () -> (Jni.alloc_object clazz : _jni_jPoint);;
28
29 class _capsule_jPoint =
30   let clazz = Jni.find_class "mypack/Point"
31   in
32     let __mid_eq =
33       try Jni.get_methodID clazz "eq" "(Lmypack/Point;)Z"
34       with
35         | _ -> failwith
36           "Unknown method from IDL in class \"mypack.Point\" : \"boolean
37             eq(mypack.Point)\"."
38     in
39       let __mid_toString =
40         try Jni.get_methodID clazz "toString" "()Ljava/lang/String;"
41         with
42           | _ -> failwith
43             "Unknown method from IDL in class \"mypack.Point\" : \"string
44               toString()\"."
45       in
46         let __mid_moveto =
47           try Jni.get_methodID clazz "moveto" "(II)V"
48           with
49             | _ -> failwith
50               "Unknown method from IDL in class \"mypack.Point\" : \"void
51                 moveto(int,int)\"."
52         in
```

```

50     let __fid_y =
51         try Jni.get_fieldID clazz "y" "I"
52     with
53     | _ -> failwith
54         "Unknown field from IDL in class \"mypack.Point\" : \"int
           y\"."
55
56     in
57         let __fid_x =
58             try Jni.get_fieldID clazz "x" "I"
59         with
60         | _ -> failwith
61             "Unknown field from IDL in class \"mypack.Point\" : \"
               int x\"."
62
63     in
64         fun (jni_ref : _jni_jPoint) ->
65             let _ =
66                 if Jni.is_null jni_ref
67                 then raise (JniHierarchy.Null_object "mypack/Point")
68                 else ()
69             in
70                 object (self)
71                 method eq =
72                     fun (_p0 : jPoint) ->
73                         let _p0 = _p0#_get_jni_jPoint
74                         in
75                             Jni.call_boolean_method jni_ref __mid_eq
76                             [| Jni.Obj _p0 |]
77                 method toString =
78                     fun () ->
79                         Jni.string_from_java
80                         (Jni.call_object_method jni_ref __mid_toString
81                         [| |])
82                 method moveto =
83                     fun _p0 _p1 ->
84                         let _p1 = _p1 in
85                         let _p0 = _p0
86                         in
87                             Jni.call_void_method jni_ref __mid_moveto
88                             [| Jni.Camlint _p0; Jni.Camlint _p1 |]
89                 method set_y =
90                     fun _p ->
91                         let _p = _p
92                         in
93                             Jni.set_camlint_field jni_ref __fid_y _p
94                 method get_y =
95                     fun () -> Jni.get_camlint_field jni_ref __fid_y
96                 method set_x =
97                     fun _p ->
98                         let _p = _p
99                         in
100                             Jni.set_camlint_field jni_ref __fid_x _p
101                 method get_x =
102                     fun () -> Jni.get_camlint_field jni_ref __fid_x
103                 method _get_jni_jPoint = jni_ref
104                 inherit JniHierarchy.top jni_ref
105             end;;

```

```

102 let jPoint_of_top (o : JniHierarchy.top) : jPoint =
103   new _capsule_jPoint (_jni_jPoint_of_jni_obj o#_get_jniobj);;
104 let _instance_of_jPoint =
105   let clazz = Jni.find_class "mypack/Point"
106   in fun (o : JniHierarchy.top) -> Jni.is_instance_of o#_get_jniobj clazz;;
107 let _new_jArray_jPoint size =
108   let java_obj = Jni.new_object_array size (Jni.find_class "mypack/Point")
109   in
110     new JniArray._Array Jni.get_object_array_element Jni.
111       set_object_array_element (fun jniobj -> new _capsule_jPoint jniobj)
112       (fun obj -> obj#_get_jni_jPoint) java_obj;;
113 let jArray_init_jPoint size f =
114   let a = _new_jArray_jPoint size
115   in (for i = 0 to pred size do a#set i (f i) done; a);;
116 let _init_point =
117   let clazz = Jni.find_class "mypack/Point" in
118   let id =
119     try Jni.get_methodID clazz "<init>" "(II)V"
120     with | _ -> failwith
121       "Unknown constructor from IDL in class \"mypack.Point\" : \"Point(int,
122         int)\".\"
123   in
124     fun (java_obj : _jni_jPoint) _p0 _p1 ->
125       let _p1 = _p1 in
126       let _p0 = _p0
127       in
128         Jni.call_nonvirtual_void_method java_obj clazz id
129         [| Jni.Camlint _p0; Jni.Camlint _p1 |];;
130 let _init_default_point =
131   let clazz = Jni.find_class "mypack/Point" in
132   let id =
133     try Jni.get_methodID clazz "<init>" "()V"
134     with | _ -> failwith
135       "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
136         ()\".\"
137   in
138     fun (java_obj : _jni_jPoint) ->
139       Jni.call_nonvirtual_void_method java_obj clazz id [| |];;
140 class point _p0 _p1 =
141   let java_obj = _alloc_jPoint ()
142   in let _ = _init_point java_obj _p0 _p1
143   in object (self) inherit _capsule_jPoint java_obj end;;
144 class default_point () =
145   let java_obj = _alloc_jPoint ()
146   in let _ = _init_default_point java_obj
147   in object (self) inherit _capsule_jPoint java_obj end;;

```

### 4.3 Génération de la classe Point par O'Jacaré 2

```
1 type top = java'lang'Object java_instance;;
2 exception Null_object of string
3 type _jni_jPoint = mypack'Point java_instance;;
4
5 class type jPoint =
6   object
7     method _get_jni_jPoint : _jni_jPoint
8     method set_x : int -> unit
9     method get_x : unit -> int
10    method set_y : int -> unit
11    method get_y : unit -> int
12    method moveto : int -> int -> unit
13    method rmoveto : int -> int -> unit
14    method toString : unit -> string
15    method display : unit -> unit
16    method distance : unit -> float
17    method eq : jPoint -> bool
18  end
19
20 class _capsule_jPoint =
21   fun (jni_ref : _jni_jPoint) ->
22     let _ =
23       if Java.is_null jni_ref
24       then raise (Null_object "mypack/Point")
25       else ()
26     in
27   object (self)
28     method eq =
29       fun (_p0 : jPoint) ->
30         let _p0 = _p0#_get_jni_jPoint in
31         Java.call "mypack.Point.eq(mypack.Point):boolean" jni_ref _p0
32     method distance =
33       fun () ->
34         Java.call "mypack.Point.distance():double" jni_ref
35     method display =
36       fun () ->
37         Java.call "mypack.Point.display():void" jni_ref
38     method toString =
39       fun () ->
40         JavaString.to_string
41         (Java.call "mypack.Point.toString():java.lang.String" jni_ref)
42     method rmoveto =
43       fun _p0 _p1 ->
44         let _p1 = Int32.of_int _p1 in
45         let _p0 = Int32.of_int _p0
46         in Java.call "mypack.Point.rmoveto(int,int):void" jni_ref _p0 _p1
47     method moveto =
48       fun _p0 _p1 ->
49         let _p1 = Int32.of_int _p1 in
50         let _p0 = Int32.of_int _p0
51         in Java.call "mypack.Point.moveto(int,int):void" jni_ref _p0 _p1
52     method set_y =
```

```

53     fun _p ->
54         let _p = Int32.of_int _p
55         in Java.set "mypack.Point.y:int" jni_ref _p
56     method get_y =
57         fun () -> Int32.to_int (Java.get "mypack.Point.y:int" jni_ref)
58     method set_x =
59         fun _p ->
60             let _p = Int32.of_int _p
61             in Java.set "mypack.Point.x:int" jni_ref _p
62     method get_x =
63         fun () -> Int32.to_int (Java.get "mypack.Point.x:int" jni_ref)
64     method _get_jni_jPoint = jni_ref
65 end;;
66
67 let jPoint_of_top (o : top) : jPoint =
68     new _capsule_jPoint (Java.cast "mypack.Point" o);;
69 let _instance_of_jPoint (o : top) =
70     Java.instanceof "mypack.Point" o;;
71
72 class point _p0 _p1 =
73     let _p1 = Int32.of_int _p1 in
74     let _p0 = Int32.of_int _p0 in
75     let java_obj = Java.make "mypack.Point(int,int)" _p0 _p1
76     in object (self) inherit _capsule_jPoint java_obj end;;
77 class default_point () =
78     let java_obj = Java.make "mypack.Point()" ()
79     in object (self) inherit _capsule_jPoint java_obj end;;

```

## 4.4 Exemple d'application : l'affichage d'une base de donnée

FrameDB.java :

```
1 package mypack;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class FrameDB extends JFrame {
7     public FrameDB (String title, String [] fields, String[][] data) {
8         super();
9         setDefaultCloseOperation(EXIT_ON_CLOSE);
10        setTitle(title);
11
12        JTable tableau = new JTable(data, fields);
13
14        getContentPane().add(tableau.getTableHeader(), BorderLayout.NORTH);
15        getContentPane().add(tableau, BorderLayout.CENTER);
16
17        pack();
18        setVisible(true);
19    }
20 }
```

jdb.idl :

```
1 package mypack;
2
3 class FrameDB {
4     [name framedb] <init> (string, [array]string, [array,array]string);
5 }
6
7 type top = java'lang'Object java_instance;;
8 exception Null_object of string;;
9 type _jni_jFrameDB = mypack'FrameDB java_instance;;
10 class type jFrameDB =
11     object method _get_jni_jFrameDB : _jni_jFrameDB end;;
12 class _capsule_jFrameDB (jni_ref : _jni_jFrameDB) =
13     let _ =
14         if Java.is_null jni_ref
15         then raise (Null_object "mypack/FrameDB")
16         else ()
17     in object (self) method _get_jni_jFrameDB = jni_ref end;;
18 let jFrameDB_of_top (o : top) : jFrameDB =
19     new _capsule_jFrameDB (Java.cast "mypack.FrameDB" o);;
20 let _instance_of_jFrameDB (o : top) =
21     Java.instanceof "mypack.FrameDB" o;;
22
23 let getJarray _p1 =
24     let _p1a = Java.make_array "java.lang.String[]" (Int32.of_int (Array.length _p1)) in
25     for i=0 to ((Array.length _p1)-1) do
```



```

21     JavaReferenceArray.set _p1a (Int32.of_int i) (JavaString.of_string _p1
22         .(i))
23     done;
24     _p1a
25 let get_array_array _p2 =
26     let _p2a =
27         Java.make_array "java.lang.String [][] " (Int32.of_int( Array.length _p2)
28             ) (Int32.of_int( Array.length _p2.(0))) in
29         for i=0 to ((Array.length _p2)-1) do
30             for j=0 to (Array.length _p2.(0))-1 do
31                 JavaReferenceArray.set (JavaReferenceArray.get _p2a (Int32.of_int i))
32                     (Int32.of_int j) (JavaString.of_string _p2.(i).(j))
33             done
34         done;
35     _p2a
36
37 class frameDB _p0 _p1 _p2 =
38     let _p2a = get_array_array _p2 in
39     let _p1a = getJarray _p1 in
40
41     let _p0 = JavaString.of_string _p0
42     in
43         let java_obj =
44             Java.make
45                 "mypack.FrameDB(java.lang.String,java.lang.String[],java.lang.
46                     String[][])"
47                 _p0 _p1a _p2a
48         in object (self) inherit _capsule_jFrameDB java_obj end;;

```

db.ml :

```

1 open Jdb
2 ...
3 ignore(new frameDB "affichage DB" fields rows)

```