L'interopérabilité entre OCaml et Java

Béatrice Carré
beatrice.carre@etu.upmc.fr

Encadrants : Emmanuel Chailloux, Xavier Clerc et Grégoire Henry

8 mai 2014

Table des matières

Introduction **3**

1 L'interopérabilité entre OCaml et Java **4**
1.1 O'Jacaré, un générateur de code d'interface 4
1.1.1 Principe global 4
1.1.2 La génération de code 5
1.1.3 compilation par camljava 6
1.2 OCaml-Java : 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

2 Portage d'O'Jacaré pour OCaml-Java : *O'Jacaré 2* **9**
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 stub_file 9
2.1.4 génération des classes encapsulantes 9
2.2 Génération de code pour Ocaml-Java 10
2.3 Comparaison d'O'Jacaré avec O'Jacaré 2 13

3 Application et performance **14**

Conclusion **14**

Bibliographie, références **15**

Annexe **16**
3.1 grammaire de l'IDL d'O'Jacaré 16
3.2 Génération de la classe Point par O'Jacaré 17
3.3 Génération de la classe Point par O'Jacaré 2 20

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 [1]), et des caractéristiques de programmation qui ne sont pas forcément gérées par les deux langages.

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

caractéristiques	*Java*	*OCaml*
accès champs	selon la visibilité	via appels de méthode
variables/méthodes statiques	✓	fonctions/décl. globales
typage dynamique	✓	pas de downcast
héritage ≡ sous-typage ?	✓	×
surcharge	✓	×
héritage multiple	seulement pour les interfaces	✓
packetages/modules	pas de modules paramétrés	✓

Deux études ont déjà été réalisées pour l'interopérabilité entre Ocaml et Java à travers leur modèle object respectif :

- *O'Jacaré (et Camljava)* conserve les runtimes des deux langages (GC, Exceptions, ...) et les fait communiquer par l'interface camljava, avec l'aide de classes encapsulantes générées par O'Jacaré.
- *OCaml-java 2.0* utilise un seul runtime, en compilant le OCaml en byte-code 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* [2]).

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 vous 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 [3], pour permettre aux interfaces avec C de chacuns 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. Les stubs générés par le callback vont 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 (Java Native Interface) 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.
- la génération des fichiers stub 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.

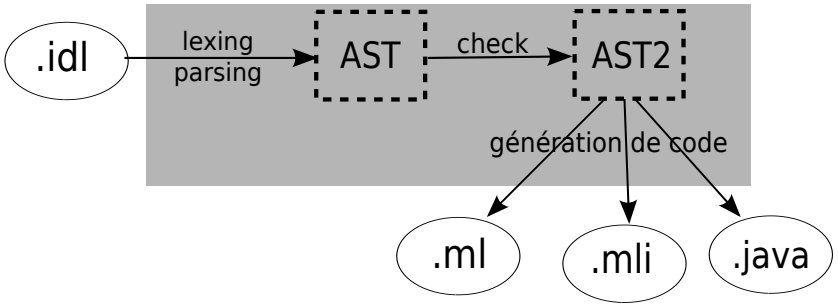

FIGURE 1 – Schéma global de la génération d'O'Jacaré

3. Langage de définition d'interface

1.1.2 La génération de code

La génération de code se fait à partir d'un IDL, dont la grammaire (BNF [4]) est définie en annexe 3.1. 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 liens entre les classes/interfaces de chaque côté, il n'est donc nécessaire de définir dans l'IDL uniquement 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 nécéssaire de définir dans l'IDL uniquement les méthodes ou attributs que l'on veut manipuler depuis Java.

Point.java

```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;
  }
}
```

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);
}
```

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.

4. Backus-Naur Form

1.1.3 compilation par camljava

Camljava gère l'interfacage entre OCaml et Java avec C, comme décrit dans la figure 3. Les références sur les objets Java correspondent à un type abstrait en OCaml, sur lesquels des opérations donnent accès à des méthodes, à des champs ou autres.

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 racine pour la mémoire reste incertaine, dans la mesure où TODO

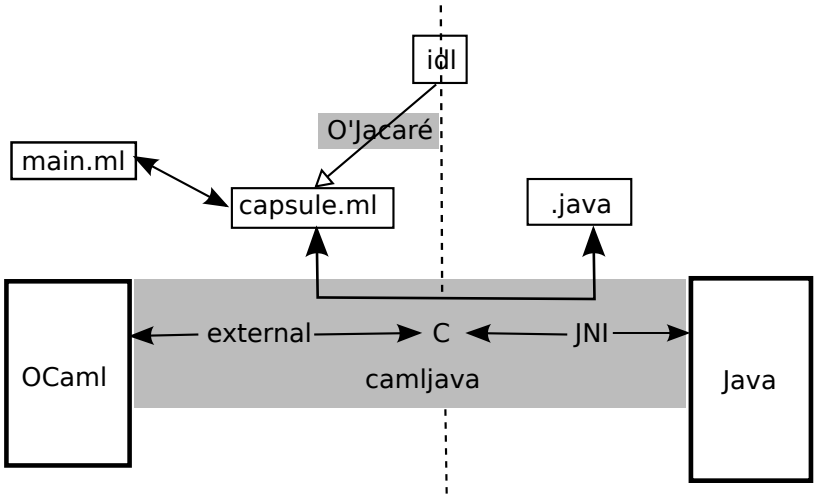

FIGURE 2 – La communication grâce à Camljava

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 [5]

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

5. Java Virtual Machine

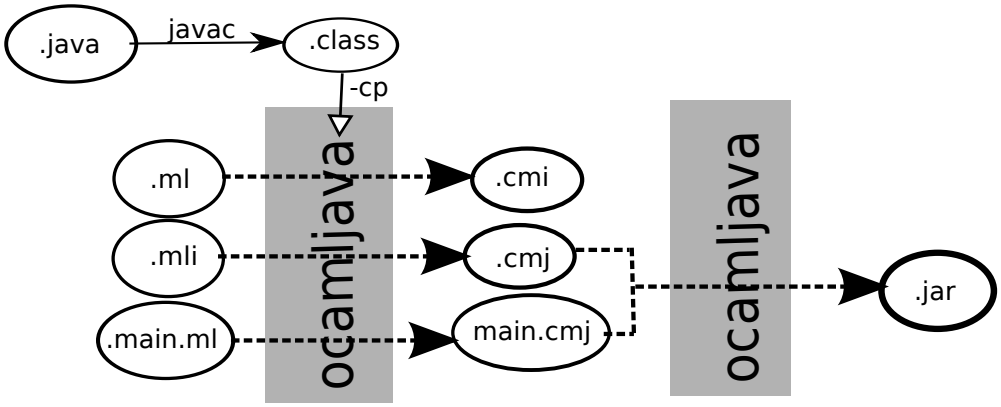

FIGURE 3 – Schéma global du compilateur d'OCaml-Java

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 :

types OCaml-Java	*descriptions et exemples*
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

```
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
```

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

```
exception Java_exception of java'lang'Throwable java_instance
```

Voici un exemple d'utilisation du module Java 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 ammène 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 Δ.

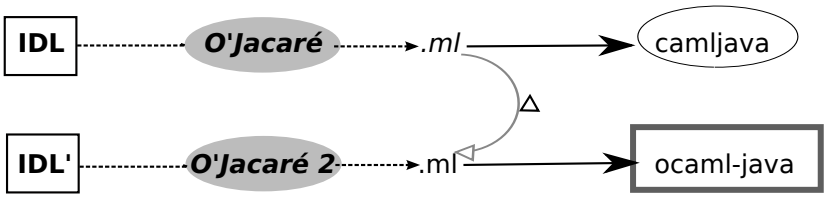

FIGURE 4 – Le travail à effectuer

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 [6], structurant les déclarations de l'IDL.

Vient ensuite la phase d'analyse sémantique, analysant l'AST obtenue par la phase précédente, vérifiant si le programme est correct, et

construisant une liste de CIdl.clazz, restructurant chaque classe ou interface définie dans l'idl. Le module Cidl définit le nouvel AST nommé CAST [7] allant être manipulé dans les passes de génération de code.

2.1.3 génération stub_file

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 notre nouvel outil ne gérant pas ce sens de communication, nous ne nous intéresserons pas à cette génération de code.

2.1.4 génération des classes encapsulantes

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 :

6. *Abstract syntax tree* ou arbre syntaxique abstrait
7. Pour *Checked AST*

élément	descriptions supplémentaires	modifications à faire
Type abstrait jni	référence sur l'objet Java	de type java_instanc
Class type t	type objet respectant le type de la classe Java	aucunes
Cast JNI (up et down)	utile pour le cast utilisateur	inutile : une fonction
Fonction d'allocation	alloue la référence Java du côté OCaml	inutile : OCaml-Java
Capsule	vérifications avant l'appel vers Java arguements des méthodes	inutile : les vérificatio conversion des types
Downcast utilisateur	cast de top vers t	simplifié grâce à la fo
Instance_of utilisateur	teste si un objet est une instance de la classe Java	
Fonction d'initialisation et Classe de construction		

Type jni
Class type
Cast JNI (up et down)
Fonction d'allocation
Capsule
Downcast utilisateur (_downcast, _instance_of)
Fonction d'initialisation et Classe de construction
fonctions / methodes static

2.2 Génération de code pour Ocaml-Java

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

ρ = "" : le nom du **package** où trouver les classes définies.
Λ = "" : le **nom de la classe** courant.
γ = false : si la déclaration est une **interface**.
θ = false : si l'élément porte l'attribut **callback**.
α = false : si l'élément est déclaré **abstract**.
δ = "JniHierarchy.top" : la classe dont **extends** la classe courante.
Δ = [] : les interfaces qu'**implements** la classe courante.

Tableau représentant les équivalents en OCaml des types Java manipulés par OCaml-Java. La troisième colonne représente les types manipulés par les programmes OCaml écrit par l'utilisateur du nouvel outil. Le problème est donc de convertir du deuxième au troisième type pour la manipulation côté OCaml et du troisième au second lors d'un appel à une fonction du module Java (un appel, un constructeur ou autre).

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

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

TYPE IDL	to_oj_Type ARGi	to_ml_type ARGi	fcast
void			
boolean			_pi
byte			_pi
char	TODO	TODO	_pi
short			
int	Int32.of_int	Int32.to_int	_pi
long	Int64.of_int	Int64.to_int	_pi
float			_pi
double			_pi
string	JavaString.of_string	JavaString.to_string	_pi
pack/Obj	_pi#_get_jni_jObj	(new _capsule_jObj ... : jObj)	(_pi : jObj)

Le type top manipulé sera le type d'instance objet de Ocaml-Java :

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

Exception :

```
1 exception Null_object of string
```

class

Le schéma de compilation de base pour une classe est largement allégé. En effet,la fonction downcast jni est inutile, puisqu'on a la fonction Java.cast, effectuant tout le travail.

De même, l'upcast ->

TODO : voir http ://www.pps.univ-paris-diderot.fr/ henry/ojacare/doc/ojacare006.html. (** cast JNI, exporté pour préparer la fonction 'import' *)

L'allocation n'est pas non plus nécessaire, OCaml-Java gérant tout ça côté Java.

La capsule est aussi très simplifiée, les tests d'existance des méthodes classes etc est aussi géré par COaml-Java.

$[\![class\ CLASS\ extends\ E\ implements\ I1, I2...\{$
$attr1; attr2; ...;$
$m1; m2; ...;$
$init1; init2; ...;$
$\}]\!]_{\rho,CB} \longrightarrow$

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

(** classe encapsulante *)
"class type jCLASS =
   object inherit E
   inherits jI1
   inherits jI2 ...
   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 "mypack/Point")
        else ()
     in

    object (self)
     (* method _get_jni_jCLASS = jni_ref
      method _get_jni_jE = jni_ref
      method _get_jni_jI1 = jni_ref
      method _get_jni_jI2 = jni_ref*)
      inherit JniHierarchy.top jni_ref
    end"

(* downcast utilisateur *)
"let jCLASS_of_top (o : TOP) : jCLASS =
    new _capsule_jCLASS (__jni_jCLASS_of_jni_obj o#_get_jniobj)"
(* instance_of *)
"let _instance_of_jCLASS =
    in fun (o : TOP) -> Jni.is_instance_of o#_get_jniobj clazz"
```

methodes

$[\![\ RTYPE\ \ METH\ (TARG1,\ TARG2, ...)]\!]_{TODO} \longrightarrow$

```
...
(* type class *)
"class type jCLASS =
```

```
    ...
    method METH : "(ml_type TARG1) -> (ml_type TARG2)" -> ... ->(ml_type
        RTYPE)
    ... "
(* capsule *)
"class _capsule_jCLASS =
    object (self)
       method METH =
          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)"\" jni_ref _p0 _p1 ..."
```

inits

$[\![[name\ INIT] < init > (TARG0,\ TARG1, ...)]\!] \longrightarrow$

```
"class INIT _p0 _p1 ... =
  let _p1 = "(to_oj_type TARG1)"  in
  let _p0 = "(to_oj_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;;"
```

attributs

$[\![TYPE\ \ ATTR;]\!] \longrightarrow$

```
...
(* type class *)
"class type jCLASS =
  ...
   method set_ATTR : "(ml_type TYPE)" -> unit
   method get_ATTR : unit -> "(ml_type TYPE)"
   ... "
(* capsule *)
"class _capsule_jCLASS =
   ...
   fun (jni_ref : _jni_jCLASS) ->
     object (self)
     ...
        method set_ATTR =
           fun "(fcast TYPE)" ->
              let _p = "(to_oj_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)
        ...
```

"

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

```
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;;
```

3 Application et performance

Conclusion

Bibliographie, 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] CHAILLOUX E., HENRY G., *O'Jacaré, une interface objet entre Objective Caml et Java*, 2004,

[3] CLERC X., *OCaml-Java : Typing Java Accesses from OCaml Programs*, Trends in Functional Programming, Lecture Notes in Computer Science Volume 7829, 2013, lien

[4] CLERC X., *OCaml-Java : OCaml on the JVM*, Trends in Functional Programming, 2012,lien

[5] CLERC X., *OCaml-Java :OCaml-Java : from OCaml sources to Java bytecodes* , Trends in Functional Programming, 2012,lien

[6] Leroy X., *The camljava project*, (`http://forge.ocamlcore.org/projects/camljava/`)

[7] CLERC X.,*OCaml-java : module Java* `http://ocamljava.x9c.fr/preview/javalib/index.html`

[8] CamlP4 (* todo *)

Annexe

3.1 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
```

3.2 Génération de la classe Point par O'Jacaré

```
type _jni_jPoint = Jni.obj;;
class type jPoint =
  object
    inherit JniHierarchy.top
    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;;
let __jni_obj_of_jni_jPoint (java_obj : _jni_jPoint) =
  (Obj.magic : _jni_jPoint -> Jni.obj) java_obj;;
let __jni_jPoint_of_jni_obj =
  let clazz =
    try Jni.find_class "mypack/Point"
    with | _ -> failwith "Class not found : mypack.Point."
  in
    fun (java_obj : Jni.obj) ->
      if not (Jni.is_instance_of java_obj clazz)
      then failwith "``cast error'' : jPoint (mypack/Point)"
      else (Obj.magic java_obj : _jni_jPoint);;
let _alloc_jPoint =
  let clazz = Jni.find_class "mypack/Point"
  in fun () -> (Jni.alloc_object clazz : _jni_jPoint);;

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

```
let jPoint_of_top (o : JniHierarchy.top) : jPoint =
  new _capsule_jPoint (__jni_jPoint_of_jni_obj o#_get_jniobj);;
let _instance_of_jPoint =
  let clazz = Jni.find_class "mypack/Point"
  in fun (o : JniHierarchy.top) -> Jni.is_instance_of o#_get_jniobj clazz;;
let _new_jArray_jPoint size =
  let java_obj = Jni.new_object_array size (Jni.find_class "mypack/Point")
  in
    new JniArray._Array Jni.get_object_array_element Jni.
      set_object_array_element (fun jniobj -> new _capsule_jPoint jniobj)
      (fun obj -> obj#_get_jni_jPoint) java_obj;;
let jArray_init_jPoint size f =
  let a = _new_jArray_jPoint size
  in (for i = 0 to pred size do a#set i (f i) done; a);;
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 |];;
let _init_default_point =
  let clazz = Jni.find_class "mypack/Point" in
  let id =
    try Jni.get_methodID clazz "<init>" "()V"
    with | _ -> failwith
         "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
            ()\"."
  in
    fun (java_obj : _jni_jPoint) ->
      Jni.call_nonvirtual_void_method java_obj clazz id [|  |];;

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 default_point () =
  let java_obj = _alloc_jPoint ()
  in let _ = _init_default_point java_obj
    in object (self) inherit _capsule_jPoint java_obj end;;
```

3.3 Génération de la classe Point par O'Jacaré 2

```
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 rmoveto : int -> int -> unit
    method toString : unit -> string
    method display : unit -> unit
    method distance : unit -> float
    method eq : jPoint -> bool
  end

class _capsule_jPoint =
  fun (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 distance =
    fun () ->
      Java.call "mypack.Point.distance():double" jni_ref
  method display =
    fun () ->
      Java.call "mypack.Point.display():void" jni_ref
  method toString =
    fun () ->
      JavaString.to_string
        (Java.call "mypack.Point.toString():java.lang.String" jni_ref)
  method rmoveto =
    fun _p0 _p1 ->
      let _p1 = Int32.of_int _p1 in
      let _p0 = Int32.of_int _p0
      in Java.call "mypack.Point.rmoveto(int,int):void" jni_ref _p0 _p1
  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 =
```

```
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;;
```