L'interopérabilité entre OCaml et Java

Béatrice Carré
beatrice.carre@etu.upmc.fr

Encadrants : Emmanuel Chailloux, Xavier Clerc et Grégoire Henry

7 mai 2014

Table des matières

Introduction

Il est parfois utile de réutiliser des structures écrites dans un certain langage sans avoir à les réécrire entièrement. Il peut arriver de vouloir utiliser l'efficacité et l'élégance du style fonctionnel d'Ocaml dans un programme autant que la portabilité du 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), 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 le code nécessaire à cet accès grâce à O'Jacaré et 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), ainsi que les tableaux.

Dans ce rapport, nous décrivons le schéma global du générateur de code d'O'Jacaré, et celui du compilateur d'Ocaml-Java pour en faire ressortir les avantages d'un portage d'O'Jacaré 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 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, pour permettre aux interfaces avec C de chacunes de communiquer.

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 TODO. Un schéma décrit ces étapes dans la figure 2.

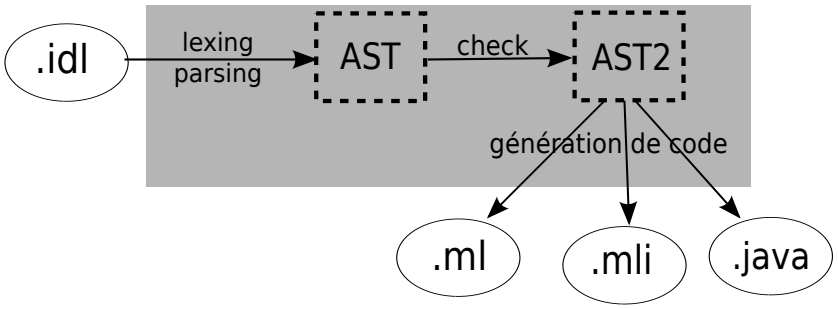

FIGURE 1 – Schéma global de la génération d'O'Jacaré

1.1.2 la génération de code

La génération de code se fait à partir d'un IDL, dont la grammaire (BNF) est définie en annexe. 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 pour la fameuse classe Point dans un fichier 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 }
11 interface Colored {
12   string getColor();
13   void setColor(string);
14 }
15 class ColoredPoint extends Point implements Colored {
16   [name default_colored_point] <init> ();
17   [name colored_point] <init> (int,int,string);
18   [name eq_colored_point] boolean eq(ColoredPoint);
19 }
```

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

- Un type objet 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

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

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 l'autre monde avant de le passer en référence.

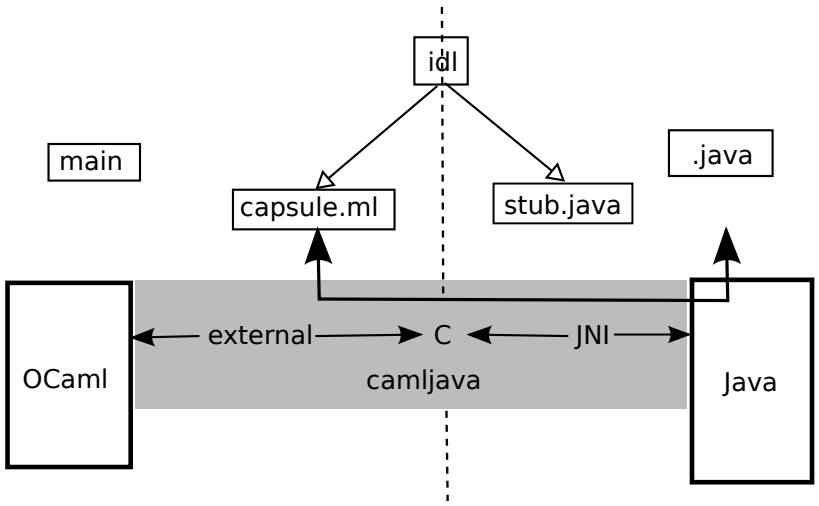

FIGURE 2 – La communication grâce à Camljava

1.2 OCaml-Java et la compilation de code OCaml vers du bytecode Java

1.2.1 Principe Global

intro TODO presentation OCaml-Java

1.2.2 Accès à du code Java

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

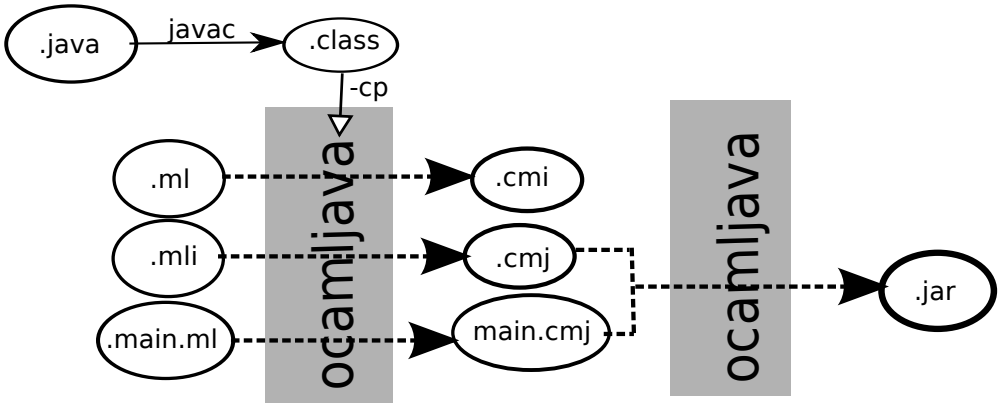

FIGURE 3 – Schéma global du compilateur d'OCaml-Java

type Java	description et exemple
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"
java_proxy	type d'une interface "java.lang.Comparable"

Description 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 définie dans le module pour permettre d'attraper les exceptions du côté OCaml :

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

Exemple :

```
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
```

```
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;;
```

1.3 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. Cette interface de bas-niveau

OCaml-Java permet l'accès à toute l'API Java depuis OCaml TODO

Solution problèmes : TODO - gestion de la surchage (absente en OCaml) -> renommage obligatoire dans IDL. - 2 runtimes vs 1 seul (-> résoud pd de communication GC et exceptions) - gestion du typage (statype en OCaml vs dynamique en Java) -

sortie prématurée du sensé binome -> manque de temps. Restriction : pas CB, pas gestion array.

1.4 travail à effectuer

étude outils O'Jacaré, mais aussi fonctionnement Camljava pour pouvoir comparer avec cible du fichier généré : OCamljava à) la place.

2 Portage d'O'Jacaré pour OCaml-Java

2.1 La génération d'O'Jacaré

2.1.1 La syntaxe de l'idl

L'IDL est défini pour la construction des interfaces entreO'Caml et Java, il est donc défini en s'approchant de l'intersection des deux mondes objet (TODO : A completer : definir cette intersection)

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

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

2.1.2 Analyse lexical, syntaxique et sémantique

La première phase est celle d'analyse lexicale et syntaxique, séparant l'idl en lexèmes et construisant l'AST, défini en annexe par Idl.file, dont la structure est définie en annexe.

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 allant être manipulé dans les passes de génération de code. Il est décrit en annexe.

2.1.3 génération stub_file

pour callback : génération java

2.1.4 génération des classes encapsulantes

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

2.2 schémas de compilation actuels d'O'Jacaré

TYPE	jni_type	getJni
void		
boolean	Jni.Boolean	
byte	Jni.Byte	
char	Jni.Char	
short	Jni.Short	
int	Jni.Camlint	
long	Jni.Long	
float	Jni.Float	
double	Jni.Double	
string	Jni.Obj _pi	Jni.string_to_java _pi
pack/Obj	Jni.Obj _pi	_pi#_get_jni_jname

2.3 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)
        ...
   "
```

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

BNF

```
class

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
```

3.1 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
```

```
50                "Unknown method from IDL in class \"mypack.Point\" : \"void
                      moveto(int,int)\"."
51          in
52            let __fid_y =
53              try Jni.get_fieldID clazz "y" "I"
54              with
55              | _ ->
56                  failwith
57                    "Unknown field from IDL in class \"mypack.Point\" : \"int
                          y\"."
58            in
59              let __fid_x =
60                try Jni.get_fieldID clazz "x" "I"
61                with
62                | _ ->
63                    failwith
64                      "Unknown field from IDL in class \"mypack.Point\" : \"
                            int x\"."
65              in
66                fun (jni_ref : _jni_jPoint) ->
67                  let _ =
68                    if Jni.is_null jni_ref
69                    then raise (JniHierarchy.Null_object "mypack/Point")
70                    else ()
71                  in
72                    object (self)
73                      method eq =
74                        fun (_p0 : jPoint) ->
75                          let _p0 = _p0#_get_jni_jPoint
76                          in
77                            Jni.call_boolean_method jni_ref __mid_eq
78                              [| Jni.Obj _p0 |]
79                      method toString =
80                        fun () ->
81                          Jni.string_from_java
82                            (Jni.call_object_method jni_ref __mid_toString
83                               [|  |])
84                      method moveto =
85                        fun _p0 _p1 ->
86                          let _p1 = _p1 in
87                          let _p0 = _p0
88                          in
89                            Jni.call_void_method jni_ref __mid_moveto
90                              [| Jni.Camlint _p0; Jni.Camlint _p1 |]
91                      method set_y =
92                        fun _p ->
93                          let _p = _p
94                          in Jni.set_camlint_field jni_ref __fid_y _p
95                      method get_y =
96                        fun () -> Jni.get_camlint_field jni_ref __fid_y
97                      method set_x =
98                        fun _p ->
99                          let _p = _p
100                         in 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;;
106 let jPoint_of_top (o : JniHierarchy.top) : jPoint =
107   new _capsule_jPoint (__jni_jPoint_of_jni_obj o#_get_jniobj);;
108 let _instance_of_jPoint =
109   let clazz = Jni.find_class "mypack/Point"
110   in fun (o : JniHierarchy.top) -> Jni.is_instance_of o#_get_jniobj clazz;;
111 let _new_jArray_jPoint size =
112   let java_obj = Jni.new_object_array size (Jni.find_class "mypack/Point")
113   in
114     new JniArray._Array Jni.get_object_array_element Jni.
115       set_object_array_element (fun jniobj -> new _capsule_jPoint jniobj)
116       (fun obj -> obj#_get_jni_jPoint) java_obj;;
117 let jArray_init_jPoint size f =
118   let a = _new_jArray_jPoint size
119   in (for i = 0 to pred size do a#set i (f i) done; a);;
120 let _init_point =
121   let clazz = Jni.find_class "mypack/Point" in
122   let id =
123     try Jni.get_methodID clazz "<init>" "(II)V"
124     with
125     | _ ->
126         failwith
127           "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
                (int,int)\"."
128   in
129     fun (java_obj : _jni_jPoint) _p0 _p1 ->
130       let _p1 = _p1 in
131       let _p0 = _p0
132       in
133         Jni.call_nonvirtual_void_method java_obj clazz id
134           [| Jni.Camlint _p0; Jni.Camlint _p1 |];;
135 let _init_default_point =
136   let clazz = Jni.find_class "mypack/Point" in
137   let id =
138     try Jni.get_methodID clazz "<init>" "()V"
139     with
140     | _ ->
141         failwith
142           "Unknown constructor from IDL in class \"mypack.Point\" : \"Point
                ()\"."
143   in
144     fun (java_obj : _jni_jPoint) ->
145       Jni.call_nonvirtual_void_method java_obj clazz id [|  |];;
146
147 class point _p0 _p1 =
148   let java_obj = _alloc_jPoint ()
149   in let _ = _init_point java_obj _p0 _p1
150     in object (self) inherit _capsule_jPoint java_obj end;;
151 class default_point () =
152   let java_obj = _alloc_jPoint ()
```

```
153   in let _ = _init_default_point java_obj
154     in object (self) inherit _capsule_jPoint java_obj end;;
```

Module CIdl, structure manipulée par O'Jacaré à partir de l'IDL

```
(**  module CIdl  *)
type typ =
  | Cvoid
  | Cboolean (** boolean -> bool *)
  | Cchar (** char -> char *)
  | Cbyte (** byte -> int *)
  | Cshort (** short -> int *)
  | Ccamlint (** int -> int<31> *)
  | Cint (** int -> int32 *)
  | Clong (** long -> int64 *)
  | Cfloat (** float -> float *)
  | Cdouble (** double -> float *)
  | Ccallback of Ident.clazz
  | Cobject of object_type (** object -> ... *)
and object_type =
  | Cname of Ident.clazz (** ... -> object *)
  | Cstring (** ... -> string *)
  | Cjavaarray of typ (** ... -> t jArray *)
  | Carray of typ (** ... -> t array *)
  | Ctop

type clazz = {
    cc_abstract: bool;
    cc_callback: bool;
    cc_ident: Ident.clazz;
    cc_extend: clazz option; (* None = top *)
    cc_implements: clazz list;
    cc_all_inherited: clazz list; (* tout jusque top ... (et avec les
        interfaces) sauf elle-meme. *)
    cc_inits: init list;
    cc_methods: mmethod list; (* methodes + champs *)
    cc_public_methods: mmethod list; (* methodes declarees + celles
        heritees *)
    cc_static_methods: mmethod list;
  }
and mmethod_desc =
  | Cmethod of bool * typ * typ list (* abstract, rtype, args *)
  | Cget of typ
  | Cset of typ
and mmethod = {
    cm_class: Ident.clazz;
    cm_ident: Ident.mmethod;
    cm_desc: mmethod_desc;
  }
and init = {
    cmi_ident: Ident.mmethod;
    cmi_class: Ident.clazz;
    cmi_args: typ list;
  }
type file = clazz list
```

module Ident

```
(* module Ident  *)
(* le type des identifiants de classe de l'IDL *)
type clazz = {
    ic_id: int;
    ic_interface: bool;
    ic_java_package: string list;
    ic_java_name: string;
    ic_ml_name: string;
    ic_ml_name_location: Loc.t;
    ic_ml_name_kind: ml_kind;
  }
type mmethod = {
    im_java_name: string;
    im_ml_id: int; (** entier unique pour une nom ml *)
    im_ml_name: string;
    im_ml_name_location:Loc.t;
    im_ml_name_kind: ml_kind;
  }
```

phases de

Type jni
    MlClass.make_jni_type
Class type
    MlClass.make_class_type
Cast JNI
    MlClass.make_jniupcast
    MlClass.make_jnidowncast
Fonction d'allocation
    MlClass.make_alloc
    MlClass.make_alloc_stub
Capsule / souche
    MlClass.make_wrapper
Downcast utilisateur
    MlClass.make_downcast
    MlClass.make_instance_of
Tableaux
    MlClass.make_array
Fonction d'initialisation
    MlClass.make_fun
Classe de construction
    MlClass.make_class
fonctions / methodes static
    MlClass.make_static