Chapitre 1

Béatrice CARRE

Introduction

La génération de code se fait en plusieurs passes :
- analyse lexicale et analyse syntaxique de l' idl donnant un ast de type Idl.file.
- vérification des types de l'ast, donnant un nouvel ast de type CIdl.file.
- la génération des fichiers stub java nécessaires pour un appel callback
- la génération à partir de l'ast CIdl.file du fichier .ml
- la génération à partir du CIdl.file du fichier .mli

Ces différentes étapes seront présentées plus en profondeur.

1.1 La syntaxe de l'idl

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.

1.2 lexing parsing

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

1.3 check

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

1.4 génération stub_file

//TODO

1.5 génération .ml

La génération de ce code se fait en plusieurs passes sur l'ast obtenu après ces précédents phases, le CIdl.file.

1.5.1 schémas de compilation

La génération de code rend du code OCaml (écrit dans un fichier).
Nous considérons un environnement contenant les variables suivantes, initialisées à leur valuer par défaut :

ρ = "" : le nom du **package** où trouver les classes définies.
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.

Et les fonctions suivantes :

init_env () : réinitialise toutes les variables d'environnement
init_class_env () : réinitialise toutes les variables d'environnement sauf ρ.
hd(elt*) : rend le premier élément de la list elt*.
tl(elt*) : rend la liste elt* privée de son premier élément.

file

$[\![package^*]\!] \longrightarrow$
$[\![hd(package^*)]\!]$
`init_env ();`
$[\![tl(package^*)]\!]$

$[\![decl^*]\!] \longrightarrow$
$[\![hd(decl^*)]\!]$
`init_class_env ();`
$[\![tl(decl^*)]\!]$

package

$[\![package\ qname\ ;\ decl^*]\!] \longrightarrow$
$[\![decl^*]\!]_{\rho=qname}$

decl interface

$[\![interface\ name]\!] \longrightarrow$
$[\![class\ name]\!]_{\rho,\Gamma=true}$

decl class

$[\![[callabck]class\ name]\!]_{\rho,\Gamma,\theta,\alpha} \longrightarrow$
$[\![class\ name]\!]_{\rho,\Gamma,\theta=true,\alpha}$

decl class

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

```
let clazz = Jni.find_class PACK/NAME

(** type jni.obj t *)
"type _jni_jNAME = Jni.obj"

(** classe encapsulante *)
"class type jNAME =
   object inherit E
   inherits jI1
   inherits jI2 ...
   method _get_jni_jNAME : _jni_jNAME
   end"

(** upcast jni *)
"let __jni_obj_of_jni_jNAME (java_obj : _jni_jNAME) =
   (Obj.magic : _jni_jNAME -> Jni.obj) java_obj"
(** downcast jni *)
"let __jni_jNAME_of_jni_obj =
   fun (java_obj : Jni.obj) ->
     Jni.is_instance_of java_obj clazz"

(* allocation : si ce n'est pas une interface *)
"let _alloc_jNAME =
     fun () -> (Jni.alloc_object clazz : _jni_jNAME)"

(* capsule wrapper *)
"class _capsule_jNAME = fun (jni_ref : _jni_jNAME) ->
    object (self)
      method _get_jni_jNAME = 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 jNAME_of_top (o : TOP) : jNAME =
    new _capsule_jNAME (__jni_jNAME_of_jni_obj o#_get_jniobj)"
(* instance_of *)
"let _instance_of_jNAME =
    in fun (o : TOP) -> Jni.is_instance_of o#_get_jniobj clazz"

(* tableaux *)
"let _new_jArray_jNAME size =
    let java_obj = Jni.new_object_array size (Jni.find_class \"PACK/NAME\")
```

```
    in
      new JniArray._Array Jni.get_object_array_element Jni.
        set_object_array_element (fun jniobj -> new _capsule_jNAME jniobj)
        (fun obj -> obj#_get_jni_jNAME) java_obj"
"let jArray_init_jNAME size f =
    let a = _new_jArray_jNAME size
    in (for i = 0 to pred size do a#set i (f i) done; a)"

(* inits *)
```

$[[name\ init1]< init >(arg*);...\}]$
$[[name\ init1]< init >(arg*);...\}]$
...

```
(* fonctions et  methodes statiques*)

    (*TODO*)
```

Ce tableau représente le résultat des fonctions str, jni_type, getJni, cast sur les types lors des générations des constructeurs ou des méthodes.

TYPE	str	jni_type	getJni	cast
void	" "||V			
boolean	Z	Jni.Boolean _pi	_pi	_pi
byte	B	Jni.Byte _pi	_pi	_pi
char	C	Jni.Char _pi	_pi	_pi
short	S	Jni.Short _pi	_pi	_pi
int	I	Jni.Camlint _pi	_pi	_pi
long	J	Jni.Long _pi	_pi	_pi
float	F	Jni.Float _pi	_pi	_pi
double	D	Jni.Double _pi	_pi	_pi
string	LJava/lang/String;	Jni.Obj _pi	Jni.string_to_java _pi	_pi
pack/Obj	Lpack/Obj;	Jni.Obj _pi	_pi#_get_jni_jname	(_pi : jObj)

inits

$[[name\ INIT]< init >(A0, A1, ...)] \longrightarrow$

```
"let _init_INIT =
  let id = Jni.get_methodID clazz \"<init>\"
             \"("(toStr A0)(toStr A1)...")V\"
  in
    fun (java_obj : _jni_jNAME) "(cast A0) (cast A1) ..." ->
      ...
      let _p1 = "(getJni A1)" in
      let _p0 = "(getJni A0)" in
      Jni.call_nonvirtual_void_method java_obj clazz id
          [| "(jni\_type A0)"; "(jni\_type A1)"; ... |]

class INIT _p0 _p1 ... =
  let java_obj = _alloc_jNAME ()
```

```
  in let _ = _init_INIT java_obj _p0 _p1 ...
    in object (self) inherit _capsule_jNAME java_obj
end"
```

attributs

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

```
...
(* type class *)
"class type jNAME =
  ...
   method set_ATTR : (j)TYPE -> unit
   method get_ATTR : unit -> (j)TYPE
   ... "
(* capsule *)
"class _capsule_jNAME =
   let __fid_ATTR = try Jni.get_fieldID clazz \"ATTR\" "(toStr TYPE)" in
   ...
   fun (jni_ref : _jni_jNAME) ->
     object (self)
        method set_ATTR =
           fun "(castArg TYPE)" ->
              let _p = "(getJni TYPE)"
              in Jni.set_object_field jni_ref __fid_ATTR _p
        method get_ATTR =
        fun () ->
           (new _capsule_jNAME (Jni.get_object_field jni_ref __fid_ATTR) :
           jNAME)
        ...
   "
```

methodes

TYPE	str	jni_type	getJni	cast
void	" "||V			
boolean	Z	Jni.Boolean _pi	_pi	_pi
byte	B	Jni.Byte _pi	_pi	_pi
char	C	Jni.Char _pi	_pi	_pi
short	S	Jni.Short _pi	_pi	_pi
int	I	Jni.Camlint _pi	_pi	_pi
long	J	Jni.Long _pi	_pi	_pi
float	F	Jni.Float _pi	_pi	_pi
double	D	Jni.Double _pi	_pi	_pi
string	LJava/lang/String;	Jni.Obj _pi	Jni.string_to_java _pi	_pi
pack/Obj	Lpack/Obj;	Jni.Obj _pi	_pi#_get_jni_jname	(_pi : jObj)

$[\![TYPEMETH(ARG1, ARG2, ...)]\!] \longrightarrow$

```
...
(* type class *)
"class type jNAME =
   ...
   method METH : ARG1 -> ARG2 -> ... -> TYPE
```

```
    ... "
(* capsule *)
"class _capsule_jNAME =
    let __mid_METH = Jni.get_methodID clazz "mETH"
          \"("(toStr ARG1)(toStr ARG2)...")"(toStr TYPE)"\"
    ...
    in
    object (self)
"(*TODO*)"       (*method METHObj1Obj2 =
          fun (_p0 : jObj1) ->
            let _p0 = _p0#_get_jni_jObj1
            ...
              in
              (new _capsule_jObj2
                (Jni.call_"Object"_method jni_ref __mid_mETHObj1Obj2
                [| Jni.Obj _p0 |]) : jObj2)
       *)
       method METH =
          fun "(cast A0) (cast A1) ..." ->
            let _p2 = "(getJni ARG2)" in
            let _p1 = "(getJni ARG1)" in
            let _p0 = "(getJni ARG0)"
            in
              Jni.call_"(aJniType TYPE)"_method jni_ref __mid_METH
                [| "(jni\_type ARG0)"; "(jni\_type ARG1)"; ... |]
```

//TODO : retour Obj dans methode array callback

1.6 génération .mli

1.7 Ocaml-Java

1.7.1 Génération de code pour Ocaml-Java

L'idée est de partir de Le type top manipulé sera le type d'instance objet de Ocaml-Java :

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

Exception :

```
exception Null_object of string
```

class

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

```
(** type jni.obj t *)
"type _jni_jNAME = PACK'NAME java_instance;;"

(** classe encapsulante *)
"class type jNAME =
   object inherit E
   inherits jI1
   inherits jI2 ...
   method _get_jni_jNAME : _jni_jNAME
   end"

(* capsule wrapper *)
"class _capsule_jNAME =
  fun (jni_ref : _jni_jNAME) ->
     let _ =
        if Java.is_null jni_ref
        then raise (Null_object "mypack/Point")
        else ()
     in

    object (self)
     (* method _get_jni_jNAME = 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 jNAME_of_top (o : TOP) : jNAME =
```

```
    new _capsule_jNAME (__jni_jNAME_of_jni_obj o#_get_jniobj)"
(* instance_of *)
"let _instance_of_jNAME =
    in fun (o : TOP) -> Jni.is_instance_of o#_get_jniobj clazz"

(* tableaux *)
"let _new_jArray_jNAME size =
    let java_obj = Jni.new_object_array size (Jni.find_class \"PACK/NAME\")
    in
      new JniArray._Array Jni.get_object_array_element Jni.
        set_object_array_element (fun jniobj -> new _capsule_jNAME jniobj)
        (fun obj -> obj#_get_jni_jNAME) java_obj"
"let jArray_init_jNAME size f =
    let a = _new_jArray_jNAME size
    in (for i = 0 to pred size do a#set i (f i) done; a)"
```

methodes Tableau représentant les équivalents en OCaml des types Java manipulés.

Java type	OCaml type
boolean	bool
byte	int
char	int
double	float
float	float
int	int32
long	int64
short	int

Tbleau associant pour chaque types de l'IDL les fonctions utiles aux schémas de compilation manipulant ceux-ci.

TYPE(IDL)	ocamlType	javaType	to_mlType	to_JavaType
void	" "||void			
boolean	boolean	_pi	_pi	_pi
byte	byte	_pi	_pi	_pi
char	char	_pi	_pi	_pi
short	short	_pi	_pi	_pi
int	int	Int32.of_int _pi	Int32.to_int _pi	_pi
long	long	Int64.of_int _pi	Int64.to_int _pi	_pi
float	float	Jni.Float _pi	_pi	_pi
double	double	Jni.Double _pi	_pi	_pi
string	java.lang.String	Jni.Obj _pi	Jni.string_to_java _pi	_pi
pack/Obj	pack.Obj	_pi#_get_jni_jPoint	_pi#_get_jni_jname	_pi

$[\![TYPEMETH(ARG1, ARG2, ...)]\!] \longrightarrow$

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

```
    method METH : ARG1 -> ARG2 -> ... -> RTYPE
    ... "
(* capsule *)
"class _capsule_jNAME =
    object (self)
       method METH =
          fun "(cast A0) (cast A1) ..." ->
            let _p1 = "(getJava ARG1)" _p1 in
            let _p0 = "(getJava ARG0)" _p0
            in
              (getJava RTYPE) "Java.call \"PACK.CLASS.METH("(javaType
            ARG1)","(javaType ARG2)","...")":"(javaType RTYPE)"\"
            jni_ref _p0 _p1 ...
```

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
```

Module Idl

```
(**  module Idl  *)

type ident = {
    id_location: Loc.t;
    id_desc: string
  }
type qident = {
    qid_location: Loc.t;
    qid_package: string list;
    qid_name: ident;
  }
type type_desc =
    Ivoid
  | Iboolean
  | Ibyte
  | Ishort
  | Icamlint
  | Iint
  | Ilong
  | Ifloat
  | Idouble
  | Ichar
  | Istring
  | Itop
  | Iarray of typ
  | Iobject of qident
and typ = {
    t_location: Loc.t;
    t_desc: type_desc;
  }
type modifier_desc =
  | Ifinal
  | Istatic
  | Iabstract
and modifier = {
    mo_location: Loc.t;
    mo_desc: modifier_desc;
}
type ann_desc =
  | Iname of ident
  | Icallback
  | Icamlarray
and annotation = {
    an_location: Loc.t;
    an_desc: ann_desc;
}
type arg = {
    arg_location: Loc.t;
    arg_annot: annotation list;
    arg_type: typ
}
type init = {
    i_location: Loc.t;
    i_annot: annotation list;
    i_args: arg list;
}
type field = {
    f_location: Loc.t;
    f_annot: annotation list;
    f_modifiers: modifier list;
    f_name: ident;
    f_type: typ
}
type mmethod = {
    m_location: Loc.t;
    m_annot: annotation list;
    m_modifiers: modifier list;
    m_name: ident;
    m_return_type: typ;
    m_args: arg list
}
type content =
    | Method of mmethod
    | Field of field
type def = {
    d_location: Loc.t;
    d_super: qident option;
    d_implements: qident list;
    d_annot: annotation list;
    d_interface: bool;
    d_modifiers: modifier list;
    d_name: ident;
    d_inits: init list;
    d_contents: content list;
}
type package = {
    p_name: string list;
    p_defs: def list;
}
type file = package list
```

Module CIdl

```
(**  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;
  }
```

idl_camlgen.make ast

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