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 une liste de valeurs imprimées (dans le fichier engendré), en modifiant Nous considérons un environnement contenant les variables suivantes :

```
let package := ""

let isInterface := false
let decl_name := ""
let isCallback := false
let isAbstractDecl := false
let extends := "JniHierarchy.top"
let implements := []
let init_list := {initname; arg*} list
```

Et les fonctions suivantes :

```
let init_env () =
  package := "";
  init_class_env ()
let init_class_env () =
  isInterface := false;
  decl_name := "";
  isCallback := false;
  isAbstractDecl := false;
  extends := "";
  implements := []
```

file

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

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

package

$[\![package\ qname\ ;\ decl^*]\!] \longrightarrow$
$[\![decl^*]\!]_p ackage := qname$

class

$[\![class]\!]_{isInterface,package,extends,implements,init\_list:\{initname,args^*\}^*,} \longrightarrow$

```
(** type jni.obj t *)
"type _jni_j"^name^" = Jni.obj;;"
(** classe encapsulante *)
"class type j"^name^" ="
(** herite top car extends = "" *)
   "object inherit "^extends^""
    make_implements implements
   "method _get_jni_j"^name^" : _jni_j"^name^"
   end;;"
(** upcast jni *)
"let __jni_obj_of_jni_j"^name^" (java_obj : _jni_j"^name^") =
   (Obj.magic : _jni_j"^name^" -> Jni.obj) java_obj;;"
(** downcast jni *)
"let __jni_j"^name^"_of_jni_obj =
   let clazz =
      try Jni.find_class \""^package^"/"^name^"\"
   with | _ -> failwith "Class not found : "^package^"."^name^"."
   in
      fun (java_obj : Jni.obj) ->
        if not (Jni.is_instance_of java_obj clazz)
        then failwith \"cast error\" : j"^name^" ("^package^"/"^name^")
        else (Obj.magic java_obj : _jni_j"^name^");;"
(* allocation *)
make_alloc package name isInterface;
(* capsule *)
make_capsule package name extends implements;
(* downcast utilisateur *)
"let j"^name^"_of_top (o : JniHierarchy.top) : j"^name^" =
    new _capsule_j"^name^" (__jni_j"^name^"_of_jni_obj o#_get_jniobj);;"
(* instance_of *)
"let _instance_of_j"^name^" =
   let clazz = Jni.find_class \""^package^"/"^name^"\"
    in fun (o : JniHierarchy.top) -> Jni.is_instance_of o#_get_jniobj clazz
       ;;"
(* tableaux *)
"let _new_jArray_j"^name^" size =
    let java_obj = Jni.new_object_array size (Jni.find_class \""^package^"/
       "^name^"\")
    in
      new JniArray._Array Jni.get_object_array_element Jni.
        set_object_array_element (fun jniobj -> new _capsule_j"^name^"
            jniobj)
        (fun obj -> obj#_get_jni_j"^name^") java_obj;;"
"let jArray_init_j"^name^" size f =
    let a = _new_jArray_j"^name^" size
    in (for i = 0 to pred size do a#set i (f i) done; a);;"
make_inits init_list;
```

interface

$[\![interface\ name]\!] \longrightarrow$
$[\![class\ name]\!]_{isInterface:=true}$

$[\![interface\ name\ implements\ qname*]\!] \longrightarrow$

```
let make_alloc name isInterface =
   if not isInterface then
      "let _alloc_j"^name^" =
       let clazz = Jni.find_class "^package^"/"^name^"
       in fun () -> (Jni.alloc_object clazz : _jni_j"^name^");;"
```

extends, implements

$[\![class\ name\ implements\ qname*]\!] \longrightarrow$
let l = String.split "," qname* in
$[\![class\ name]\!]_{implements:=l}$

$[\![class\ name\ extends qname]\!] \longrightarrow$
$[\![class\ name]\!]_{extends:=qname}$

```
let make_inherits_implements implements =
   List.iter (fun int -> "inherit j"^int ) implements

let make_capsule package name extends implements =
     "class _capsule_j"^name^" =
       let clazz = Jni.find_class \""^package^"/"^name^"\"
       in"
    (* si la class extends une autre *)
    if extends != "JniHierarchy.top" then (
        " let _ =
            if not (Jni.is_assignable_from clazz (Jni.find_class \""^package
                ^"/"^extends^"\"))
            then
            failwith \"Wrong super class in IDL : "^package^"/"^classname^"
                not extends"^package^"/"^extends^".\"
            else ()
         in "
    );
   (* pour chaque interface implementee : *)
    List.iter (fun int ->
        "let _ =
            if not (Jni.is_assignable_from clazz (Jni.find_class "^package^"
                /"^int^"))
            then
              failwith \"Wrong implemented interface in IDL : "^package^"."^
                  name^ "does not implements "^package^"."^int^".\"
            else ()
        in"
    ) implements;
```

```
      (* fonction commune *)
       "fun (jni_ref : _jni_j"^name^") ->
         let _ =
           if Jni.is_null jni_ref
           then raise (JniHierarchy.Null_object \""^package^"/"^name^"\")
           else ()
         in
         (* objet avec les methodes d'acces a toutes les classes/interfaces
              implementee ou heritees *)
         object (self)
           method _get_jni_j"^name^" = jni_ref"
      if extends != "JniHierarchy.top" then (
          "  method _get_jni_j"^extends^" = jni_ref"
      );
      List.iter (fun int -> "method _get_jni_j"^int^" = jni_ref;;")
          implements;
          " inherit JniHierarchy.top jni_ref
         end;;"
```

inits

$[\![...class\ name\ ...\{...[name\ initname] < init > (arg*); ...\}]\!] \longrightarrow$
$[\![class\ name]\!]_{init\_list := \{initname; arg*\} :: init\_list}$

```
let getArgsString args =
String.concat (List.map (fun arg ->
   match arg with
   | ("int",name,attr) -> "I"
   | ("boolean",name,attr) -> "Z"
   | ("string",name,attr) -> "LJava/lang/String"
     ...
   | (obj,name,attr) -> "L"^package^obj^";"
in

(* todo : fun pr recup jtype, method get_jni_jtype et jni.mltype *)

let makefun = (* exemples : *)
 "fun (java_obj : _jni_j"^classname^") _p0 _p1 ->
       let _p1 = _p1 in
       let _p0 = _p0
       in
         Jni.call_nonvirtual_void_method java_obj clazz id
           [| Jni. _p0; Jni.Camlint _p1 "|];;"
 fun (java_obj : _jni_jOu) (_p0 : jFormule) (_p1 : jFormule) ->
       let _p1 = _p1#_get_jni_jFormule in
       let _p0 = _p0#_get_jni_jFormule
       in
         Jni.call_nonvirtual_void_method java_obj clazz id
           [| Jni.Obj _p0; Jni.Obj _p1 |];;
in"

"let _init_"^initname^" =
  let clazz = Jni.find_class \""^package^"/"^classname^"\" in
```

```
  let id =
    try Jni.get_methodID clazz \"<init>\" \"("^(getArgsString args)^")V\"
    with
    | _ ->
        failwith
          \"Unknown constructor from IDL in class \"^package^"."^classname
              ^"\" : \"^classname^"(int,int)\".\"
  in
    fun (java_obj : _jni_j"^classname^") _p0 _p1 ->
      let _p1 = _p1 in
      let _p0 = _p0
      in
        Jni.call_nonvirtual_void_method java_obj clazz id
          [| Jni."^^" _p0; Jni.Camlint _p1 |];;
let _init_"^initname^" =
  let clazz = Jni.find_class \"^package^"/"^classname^"\" in
  let id =
    try Jni.get_methodID clazz \"<init>\" \"()V\"
    with
    | _ ->
        failwith
          \"Unknown constructor from IDL in class \"^package^"."^classname
              ^"\" : \"^classname^"()\".\"
  in
    fun (java_obj : _jni_j"^classname^") ->
      Jni.call_nonvirtual_void_method java_obj clazz id [|

       |];;"
```

attributs

methodes

1.6 génération .mli

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