Code ILP6

Christian.Queinnec@lip6.fr

27 août 2013

Table des matières

Grammars/grammar6.rnc

```
# Sixième version du langage étudié: ILP6 pour « Inouï Langage
# Performant ». Pourquoi pas ajouter un attribut pour retenir les
# numéros de ligne du programme initial ?

include "grammar4.rnc"
start |= programme6

# On ajoute quelques expressions permettant de manipuler des objets
# (des tables associatives qui ont de la classe).

expression |=
   creationObjet
 | lectureChamp
 | ecritureChamp
 | envoiMessage
 | appelSuper
 | moi

# Un programme peut aussi contenir des définitions de classes en tête.

programme6 = element programme6 {
   ( definitionFonction | definitionClasse ) *,
   expression +
}

# Une définition de classe comporte un nom, le nom de la super-classe
# (c'est de l'héritage simple), des champs et des méthodes. Une
# restriction du langage est que tous les noms de champs distincts
# doivent être différents ou, plus exactement, quand un champ F est
# défini pour une classe C, seules les sous-classes de C peuvent
# parler de F et aucune autre classe ne peut définir un autre champ
# pareillement nommé (un renommage global permet d'assurer cela).

definitionClasse = element definitionClasse {
   attribute nom    { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
   attribute parent { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) } ?,
   element champs {
      element champ {
         attribute nom { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) }
      } *
   } ?,
   element methodes {
      element methode {
         attribute nom { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
         element variables { variable * },
         element corps     { expression + }
      } *
   } ?
}

# Allouer un objet.

creationObjet = element creationObjet {    # new Classe(arguments)
   attribute classe  { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
   expression *
}

# Lire un champ d'objet

lectureChamp = element lectureChamp {      # objet.champ
   attribute champ { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
   element cible   { expression }
}

# Écrire dans un champ d'objet

ecritureChamp = element ecritureChamp {    # objet.champ = expression
   attribute champ { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
   element cible   { expression },
   element valeur  { expression }
}

# Envoyer un message à un objet accompagné de quelques arguments.

envoiMessage = element envoiMessage {      # receveur.message(arguments)
   attribute message { xsd:Name - ( xsd:Name { pattern = "(ilp|ILP)" } ) },
   element receveur  { expression },
   element arguments { expression * }
}
```

```
# Invoquer la méthode de la super-classe (la méthode qui aurait été
# invoquée à la place de l'actuelle).

appelSuper = element appelSuper {          # super()
   empty
}

# La pseudo-variable permettant de parler de soi.

moi = element moi {                        # this
   empty
}
```

C/ilpObj.c

```
   $Id: ilpObj.c 1057 2011-08-19 12:14:07Z queinnec $ */
/* ************************************************************
 * ILP -- Implantation d'un langage de programmation.
 * Copyright (C) 2004 <Christian.Queinnec@lip6.fr>
 * $Id: ilpObj.c 1057 2011-08-19 12:14:07Z queinnec $
 * GPL version>=2
 * ************************************************************/

/** Ce fichier constitue la bibliothèque d'exécution d'ILP6. */

#include "ilpObj.h"

char *ilpObj_Id = "$Id: ilpObj.c 1057 2011-08-19 12:14:07Z queinnec $";

/** Les classes de base. */

extern struct ILP_Class ILP_object_Object_class;

struct ILP_Class ILP_object_Class_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Class",
         1,
         &ILP_object_super_field,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Object_class = {
     &ILP_object_Class_class,
     { { NULL,
         "Object",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Method_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Method",
         0,
         NULL,
         3,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Field_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Field",
         1,
         &ILP_object_defining_class_field,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Integer_class = {
     &ILP_object_Class_class,
```



```
     { { &ILP_object_Object_class,
         "Integer",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Float_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Float",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Boolean_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Boolean",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_String_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "String",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

struct ILP_Class ILP_object_Exception_class = {
     &ILP_object_Class_class,
     { { &ILP_object_Object_class,
         "Exception",
         0,
         NULL,
         2,
         { ILP_print,
           ILP_classOf } } }
};

/** Les champs prédéfinis.
 *
 * Tous les champs des structures C qui ne mènent pas à des valeurs
 * d'ILP ne sont pas considérés comme des champs. C'est pourquoi (1)
 * ces champs sont regroupés en tête de structure et (2) il y en a peu.
 */

struct ILP_Field ILP_object_super_field = {
     &ILP_object_Field_class,
     { { &ILP_object_Class_class,
         NULL,
         "super",
         0 } }
};

struct ILP_Field ILP_object_defining_class_field = {
     &ILP_object_Field_class,
     { { &ILP_object_Field_class,
         NULL,
         "defining_class",
         0 } }
};

struct ILP_Field ILP_object_previous_field_field = {
     &ILP_object_Field_class,
     { { &ILP_object_Field_class,
         &ILP_object_defining_class_field,
         "previous_field",
         1 } }
```

```
};

struct ILP_Field ILP_object_class_defining_field = {
     &ILP_object_Field_class,
     { { &ILP_object_Method_class,
         NULL,
         "class_defining",
         0 } }
};

/** Les méthodes prédéfinies.
 *
 * Il n'y en a que deux (pour l'instant) d'arité nulle:
 *     o.print() qui imprime l'objet
 *     o.getClass() qui renvoie sa classe.
 */

struct ILP_Method ILP_object_print_method = {
     &ILP_object_Method_class,
     { { &ILP_object_Object_class,
         "print",
         1,       /* arité (incluant self) */
         0        /* offset */
     } }
};

struct ILP_Method ILP_object_classOf_method = {
     &ILP_object_Method_class,
     { { &ILP_object_Object_class,
         "classOf",
         1,       /* arité (incluant self) */
         1        /* offset */
     } }
};

/** Booléens.
 *
 * On alloue statiquement les deux booléens.
 */

struct ILP_Object ILP_object_true = {
     &ILP_object_Boolean_class,
     { ILP_BOOLEAN_TRUE_VALUE }
};

struct ILP_Object ILP_object_false = {
     &ILP_object_Boolean_class,
     { ILP_BOOLEAN_FALSE_VALUE }
};

/** Exceptions
 *
 * L'exception courante.
 * NOTA: il serait mieux qu'elle soit dynamiquement allouée.
 */

static struct ILP_Exception ILP_the_exception =  {
     &ILP_object_Exception_class,
     { { "",
         { NULL } } }
};

/** Ces variables globales contiennent:
 *  -- le rattrapeur d'erreur courant
 *  -- l'exception courante (lorsque signalée)
 */

static struct ILP_catcher ILP_the_original_catcher = {
     NULL
};
struct ILP_catcher *ILP_current_catcher = &ILP_the_original_catcher;

ILP_Object ILP_current_exception = NULL;

/** Signaler une exception. */

ILP_Object
ILP_throw (ILP_Object exception)
{
     ILP_current_exception = exception;
     if ( ILP_current_catcher == &ILP_the_original_catcher ) {
          ILP_die("No current catcher!");
     };
     longjmp(ILP_current_catcher->_jmp_buf, 1);
     /** UNREACHABLE */
     return NULL;
}

/** Chaîner le nouveau rattrapeur courant avec l'ancien. */

void
ILP_establish_catcher (struct ILP_catcher *new_catcher)
{
     new_catcher->previous = ILP_current_catcher;
     ILP_current_catcher = new_catcher;
}

/** Remettre en place un rattrapeur. */

void
ILP_reset_catcher (struct ILP_catcher *catcher)
{
     ILP_current_catcher = catcher;
}

/**
 * Signalement d'une erreur.
 */

ILP_Object
ILP_error (char *message)
{
     snprintf(ILP_the_exception._content.asException.message,
              ILP_EXCEPTION_BUFFER_LENGTH,
              "Error: %s\n",
              message);
     fprintf(stderr, "%s", ILP_the_exception._content.asException.message);
     ILP_the_exception._content.asException.culprit[0] = NULL;
     return ILP_throw((ILP_Object) &ILP_the_exception);
}

/** Une fonction pour signaler qu'un argument n'est pas du type attendu. */

ILP_Object
ILP_domain_error (char *message, ILP_Object o)
{
     snprintf(ILP_the_exception._content.asException.message,
              ILP_EXCEPTION_BUFFER_LENGTH,
              "Domain error: %s\nCulprit: 0x%p\n",
              message, (void*) o);
     fprintf(stderr, "%s", ILP_the_exception._content.asException.message);
     ILP_the_exception._content.asException.culprit[0] = o;
     ILP_the_exception._content.asException.culprit[1] = NULL;
     return ILP_throw((ILP_Object) &ILP_the_exception);
}

/** Une fonction pour stopper abruptement l'application. */

ILP_Object
ILP_die (char *message)
{
     fputs(message, stderr);
     fputc('\n', stderr);
     fflush(stderr);
     exit(EXIT_FAILURE);
}

/** Vérifier si une instance est d'une certaine classe.
 * Cet algorithme est linéaire: on peut faire mieux! */

int /* boolean */
ILP_is_a (ILP_Object o, ILP_Class class)
{
     ILP_Class oclass = o->_class;
     if ( oclass == class ) {
          return 1;
     } else {
          oclass = oclass->_content.asClass.super;
          /* Object a NULL pour superclasse. */
          while ( oclass ) {
               if ( oclass == class ) {
                    return 1;
               };
```

```
            oclass = oclass->_content.asClass.super;
        }
        return 0;
    }
}

static int
ILP_is_subclass_of (ILP_Class oclass, ILP_Class otherclass)
{
    if ( oclass == otherclass ) {
        return 1;
    } else {
        oclass = oclass->_content.asClass.super;
        /* Object a NULL pour superclasse. */
        while ( oclass ) {
            if ( oclass == otherclass ) {
                return 1;
            };
            oclass = oclass->_content.asClass.super;
        }
        return 0;
    }
}

/** Déterminer une méthode. */

ILP_general_function
ILP_find_method (ILP_Object receiver,
                 ILP_Method method,
                 int argc)
{
    ILP_Class oclass = receiver->_class;
    if ( ! ILP_is_subclass_of(oclass,
                              method->_content.asMethod.class_defining) ) {
        /* Signaler une absence de méthode */
        snprintf(ILP_the_exception._content.asException.message,
                 ILP_EXCEPTION_BUFFER_LENGTH,
                 "No such method %s\nCulprit: 0x%p\n",
                 method->_content.asMethod.name,
                 (void*) receiver);
        /*DEBUG*/
        fprintf(stderr, "%s", ILP_the_exception._content.asException.message);
        ILP_the_exception._content.asException.culprit[0] = receiver;
        ILP_the_exception._content.asException.culprit[1] =
            (ILP_Object) method;
        ILP_the_exception._content.asException.culprit[2] = NULL;
        ILP_throw((ILP_Object) &ILP_the_exception);
        /* UNREACHED */
        return NULL;
    };
    if ( argc != method->_content.asMethod.arity ) {
        /* Signaler une erreur d'arité */
        snprintf(ILP_the_exception._content.asException.message,
                 ILP_EXCEPTION_BUFFER_LENGTH,
                 "Method %s arity error: %d instead of %d\nCulprit: 0x%p\n",
                 method->_content.asMethod.name,
                 argc,
                 method->_content.asMethod.arity,
                 (void*) receiver);
        /*DEBUG*/
        fprintf(stderr, "%s", ILP_the_exception._content.asException.message);
        ILP_the_exception._content.asException.culprit[0] = receiver;
        ILP_the_exception._content.asException.culprit[1] =
            (ILP_Object) method;
        ILP_the_exception._content.asException.culprit[2] = NULL;
        ILP_throw((ILP_Object) &ILP_the_exception);
        /* UNREACHED */
        return NULL;
    };
    {
        int index = method->_content.asMethod.index;
        return oclass->_content.asClass.method[index];
    }
}

#define DefineSuperMethodCaller(i) \
ILP_Object \
ILP_find_and_call_super_method##i ( \
    ILP_Object self, \
    ILP_Method current_method, \
    ILP_general_function super_method, \
```



```
    ILP_Object arguments[1] ) \
{ \
    /* assert( super_method != NULL ); */ \
    switch ( i ) { \
        case 0: { \
            return (*super_method)(self); \
        } \
        case 1: { \
            return (*super_method)(self, arguments[1]); \
        } \
        case 2: { \
            return (*super_method)(self, arguments[1], arguments[2]); \
        } \
        case 3: { \
            return (*super_method)(self, arguments[1],  \
                                   arguments[2],  \
                                   arguments[3]); \
        } \
        default: { \
            snprintf(ILP_the_exception._content.asException.message, \
                     ILP_EXCEPTION_BUFFER_LENGTH, \
                     "Cannot invoke supermethod %s\nCulprit: 0x%p\n", \
                     current_method->_content.asMethod.name,  \
                     (void*) self ); \
            /*DEBUG*/ \
            fprintf(stderr, "%s", ILP_the_exception._content.asException.message); \
            ILP_the_exception._content.asException.culprit[0] = self; \
            ILP_the_exception._content.asException.culprit[1] =  \
                (ILP_Object) current_method; \
            ILP_the_exception._content.asException.culprit[2] = NULL; \
            ILP_throw((ILP_Object) &ILP_the_exception); \
            /* UNREACHED */ \
            return NULL; \
        } \
    } \
}

DefineSuperMethodCaller(0)
DefineSuperMethodCaller(1)
DefineSuperMethodCaller(2)
DefineSuperMethodCaller(3)

ILP_Object
ILP_dont_call_super_method (
    ILP_Object self,
    ILP_Method current_method,
    ILP_general_function super_method,
    ILP_Object arguments[1] )
{
    /* assert ( super_method == NULL ); */
    snprintf(ILP_the_exception._content.asException.message,
             ILP_EXCEPTION_BUFFER_LENGTH,
             "No supermethod %s\nCulprit: 0x%p\n",
             current_method->_content.asMethod.name,
             (void*) self );
    /*DEBUG*/
    fprintf(stderr, "%s", ILP_the_exception._content.asException.message);
    ILP_the_exception._content.asException.culprit[0] = self;
    ILP_the_exception._content.asException.culprit[1] =
        (ILP_Object) current_method;
    ILP_the_exception._content.asException.culprit[2] = NULL;
    ILP_throw((ILP_Object) &ILP_the_exception);
    /* UNREACHED */
    return NULL;
}

/** Allocateurs. ILP_malloc est paramétré par l'allocateur de bas
 * niveau employé, par défaut: malloc() mais ce peut être GC_malloc()
 * Cf. ilpObj.h pour plus de détails sur l'emploi d'un GC. */

ILP_Object
ILP_malloc (int size, ILP_Class class)
{
    ILP_Object result = ILP_MALLOC(size);
    if ( result == NULL ) {
        return ILP_die("Memory exhaustion");
    };
    result->_class = class;
    return result;
}
```

```
ILP_Object
ILP_make_instance (ILP_Class class)
{
     int size = sizeof(ILP_Class);
     size += sizeof(ILP_Object) * class->_content.asClass.fields_count;
     return ILP_malloc(size, class);
}

/** Ce n'est pas une vraie allocation mais une simple conversion. */

ILP_Object
ILP_make_boolean (int b)
{
     if ( b ) {
          return ILP_TRUE;
     } else {
          return ILP_FALSE;
     }
}

ILP_Object
ILP_make_integer (int d)
{
     ILP_Object result = ILP_AllocateInteger();
     result->_content.asInteger = d;
     return result;
}

ILP_Object
ILP_make_float (double d)
{
     ILP_Object result = ILP_AllocateFloat();
     result->_content.asFloat = d;
     return result;
}

ILP_Object
ILP_pi ()
{
     static ILP_Object object_pi = NULL;
     if ( object_pi == NULL ) {
          object_pi = ILP_make_float(ILP_PI_VALUE);
     }
     return object_pi;
}

ILP_Object
ILP_make_string (char *s)
{
     int size = strlen(s);
     ILP_Object result = ILP_AllocateString(size);
     result->_content.asString._size = size;
     memmove(result->_content.asString.asCharacter, s, size);
     return result;
}

/** String primitives */

static ILP_Object
ILP_concatenate_strings (ILP_Object o1, ILP_Object o2)
{
     int size1 = o1->_content.asString._size;
     int total_size = size1 + o2->_content.asString._size;
     ILP_Object result = ILP_AllocateString(total_size);
     memmove(&(result->_content.asString.asCharacter[0]),
             o1->_content.asString.asCharacter,
             o1->_content.asString._size);
     memmove(&(result->_content.asString.asCharacter[size1]),
             o2->_content.asString.asCharacter,
             o2->_content.asString._size);
     return result;
}

/** Opérateurs unaires. */

ILP_Object
ILP_make_opposite (ILP_Object o)
{
     ILP_CheckIfInteger(o);
     {
          ILP_Object result = ILP_AllocateInteger();
```



```
          result->_content.asInteger = (- o->_content.asInteger);
          return result;
     }
}

ILP_Object
ILP_make_negation (ILP_Object o)
{
     ILP_CheckIfBoolean(o);
     {
          if ( ILP_isTrue(o) ) {
               return ILP_FALSE;
          } else {
               return ILP_TRUE;
          }
     }
}

/** Opérateurs binaires. */

/* DefineOperator(addition, +) est incorrect car + représente également
 * la concaténation des chaînes de caractères. */

ILP_Object
ILP_make_addition (ILP_Object o1, ILP_Object o2)
{
     if ( ILP_isInteger(o1) ) {
          if ( ILP_isInteger(o2) ) {
               ILP_Object result = ILP_AllocateInteger();
               result->_content.asInteger =
                    o1->_content.asInteger + o2->_content.asInteger;
               return result;
          } else if ( ILP_isFloat(o2) ) {
               ILP_Object result = ILP_AllocateFloat();
               result->_content.asFloat =
                    o1->_content.asInteger + o2->_content.asFloat;
               return result;
          } else {
               return ILP_domain_error("Not a number", o2);
          }
     } else if ( ILP_isFloat(o1) ) {
          if ( ILP_isInteger(o2) ) {
               ILP_Object result = ILP_AllocateFloat();
               result->_content.asFloat =
                    o1->_content.asFloat + o2->_content.asInteger;
               return result;
          } else if ( ILP_isFloat(o2) ) {
               ILP_Object result = ILP_AllocateFloat();
               result->_content.asFloat =
                    o1->_content.asFloat + o2->_content.asFloat;
               return result;
          } else {
               return ILP_domain_error("Not a number", o2);
          }
     } else if ( ILP_isString(o1) ) {
          if ( ILP_isString(o2) ) {
               return ILP_concatenate_strings(o1, o2);
          } else {
               return ILP_domain_error("Not a string", o2);
          }
     } else {
          return ILP_domain_error("Not addable", o1);
     }
}

#define DefineOperator(name,op)                                          \
ILP_Object                                                               \
ILP_make_##name (ILP_Object o1, ILP_Object o2)                           \
{                                                                        \
     if ( ILP_isInteger(o1) ) {                                          \
          if ( ILP_isInteger(o2) ) {                                     \
               ILP_Object result = ILP_AllocateInteger();                \
               result->_content.asInteger =                              \
                    o1->_content.asInteger op o2->_content.asInteger;    \
               return result;                                            \
          } else if ( ILP_isFloat(o2) ) {                                \
               ILP_Object result = ILP_AllocateFloat();                  \
               result->_content.asFloat =                                \
                    o1->_content.asInteger op o2->_content.asFloat;      \
               return result;                                            \
```

```
            } else {                                                    \
                return ILP_domain_error("Not a number", o2);            \
            }                                                           \
        } else if ( ILP_isFloat(o1) ) {                                 \
            if ( ILP_isInteger(o2) ) {                                  \
                ILP_Object result = ILP_AllocateFloat();                \
                result->_content.asFloat =                              \
                    o1->_content.asFloat op o2->_content.asInteger;     \
                return result;                                          \
            } else if ( ILP_isFloat(o2) ) {                             \
                ILP_Object result = ILP_AllocateFloat();                \
                result->_content.asFloat =                              \
                    o1->_content.asFloat op o2->_content.asFloat;       \
                return result;                                          \
            } else {                                                    \
                return ILP_domain_error("Not a number", o2);            \
            }                                                           \
        } else {                                                        \
            return ILP_domain_error("Not a number", o1);                \
        }                                                               \
    }

    DefineOperator(subtraction, -)
    DefineOperator(multiplication, *)
    DefineOperator(division, /)

    /* DefineOperator(modulo, %) est incorrect car le modulo ne se prend
     * que sur de entiers. */

    ILP_Object
    ILP_make_modulo (ILP_Object o1, ILP_Object o2)
    {
        if ( ILP_isInteger(o1) ) {
            if ( ILP_isInteger(o2) ) {
                ILP_Object result = ILP_AllocateInteger();
                result->_content.asInteger =
                    o1->_content.asInteger % o2->_content.asInteger;
                return result;
            } else {
                return ILP_domain_error("Not an integer", o2);
            }
        } else {
            return ILP_domain_error("Not an integer", o1);
        }
    }

    #define DefineComparator(name,op)                                   \
    ILP_Object                                                          \
    ILP_compare_##name (ILP_Object o1, ILP_Object o2)                   \
    {                                                                   \
        if ( ILP_isInteger(o1) ) {                                      \
            if ( ILP_isInteger(o2) ) {                                  \
                return ILP_make_boolean(                                \
                    o1->_content.asInteger op o2->_content.asInteger);  \
            } else if ( ILP_isFloat(o2) ) {                             \
                return ILP_make_boolean(                                \
                    o1->_content.asInteger op o2->_content.asFloat);    \
            } else {                                                    \
                return ILP_domain_error("Not a number", o2);            \
            }                                                           \
        } else if ( ILP_isFloat(o1) ) {                                 \
            if ( ILP_isInteger(o2) ) {                                  \
                return ILP_make_boolean(                                \
                    o1->_content.asFloat op o2->_content.asInteger);    \
            } else if ( ILP_isFloat(o2) ) {                             \
                return ILP_make_boolean(                                \
                    o1->_content.asFloat op o2->_content.asFloat);      \
            } else {                                                    \
                return ILP_domain_error("Not a number", o2);            \
            }                                                           \
        } else {                                                        \
            return ILP_domain_error("Not a number", o1);                \
        }                                                               \
    }

    DefineComparator(less_than, <)
    DefineComparator(less_than_or_equal, <=)
    DefineComparator(equal, ==)
    DefineComparator(greater_than, >)
    DefineComparator(greater_than_or_equal, >=)
```



```
    DefineComparator(not_equal, !=)

    /** Primitives */

    ILP_Object
    ILP_newline ()
    {
        fputc('\n', stdout);
        return ILP_FALSE;
    }

    /** Imprimer le contenu d'une instance. */

    void
    ILP_print_fields (ILP_Object o,
                      ILP_Field last)
    {
        if ( last == NULL ) {
            return;
        };
        ILP_print_fields(o, last->_content.asField.previous_field);
        fprintf(stdout, ":%s=", last->_content.asField.name);
        ILP_print(o->_content.asInstance.field[last->_content.asField.offset]);
    }

    /** Cette fonction peut être utilisée comme une méthode. Elle imprime
     * le receveur sur le flux de sortie. */

    ILP_Object
    ILP_print (ILP_Object self)
    {
        if ( self->_class == &ILP_object_Integer_class ) {
            fprintf(stdout, "%d", self->_content.asInteger);
        } else if ( self->_class == &ILP_object_Float_class ) {
            fprintf(stdout, "%12.5g", self->_content.asFloat);
        } else if ( self->_class == &ILP_object_Boolean_class ) {
            fprintf(stdout, "%s", (ILP_isTrue(self) ? "true" : "false"));
        } else if ( self->_class == &ILP_object_String_class ) {
            fprintf(stdout, "%s", self->_content.asString.asCharacter);
        } else if ( self->_class == &ILP_object_Class_class ) {
            fprintf(stdout, "<Class:%s>", self->_content.asClass.name);
        } else if ( self->_class == &ILP_object_Method_class ) {
            fprintf(stdout, "<Method:%s>", self->_content.asMethod.name);
        } else if ( self->_class == &ILP_object_Field_class ) {
            fprintf(stdout, "<Field:%s>", self->_content.asField.name);
        } else {
            fprintf(stdout, "<%s", self->_class->_content.asClass.name);
            ILP_print_fields(self, self->_class->_content.asClass.last_field);
            fprintf(stdout, ">");
        }
        return ILP_FALSE;
    }

    /** Cette fonction renvoie la classe du receveur. La classe est
     * également un objet obéissant au modèle ObjVlisp. */

    ILP_Object
    ILP_classOf (ILP_Object self)
    {
        return (ILP_Object) (self->_class);
    }

    /* end of ilpObj.c */
```

C/ilpObj.h

```
    #ifndef ILPOBJ_H
    #define ILPOBJ_H

    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <setjmp.h>

    /** Compatibilite inter-plateforme */
    #if !defined(__APPLE_CC__)
    extern int snprintf(char *str, size_t size, const char *format, ...);
```

```
#endif

/** Les fonctions d'ILP sont représentées en C par des fonctions qui
 * prennent des ILP_Object et renvoie un ILP_Object. */

typedef struct ILP_Object* (*ILP_general_function)();

/** Il y a deux sortes de booléens et ces deux constantes les repèrent. */

enum ILP_BOOLEAN_VALUE {
     ILP_BOOLEAN_FALSE_VALUE = 0,
     ILP_BOOLEAN_TRUE_VALUE  = 1
};


#define ILP_EXCEPTION_BUFFER_LENGTH    1000
#define ILP_EXCEPTION_CULPRIT_LENGTH     10


/** Toutes les valeurs manipulées ont cette forme:
 * un entête indiquant la classe suivi des champs appropriés.
 */

typedef struct ILP_Object {
     struct ILP_Class*  _class;
     union {
          unsigned char asBoolean;
          int           asInteger;
          double        asFloat;
          struct asString {
               int      _size;
               char     asCharacter[1];
          } asString;
          struct asException {
               char              message[ILP_EXCEPTION_BUFFER_LENGTH];
               struct ILP_Object*  culprit[ILP_EXCEPTION_CULPRIT_LENGTH];
          } asException;
          struct asClass {
               struct ILP_Class*   super;
               char*               name;
               int                 fields_count;
               struct ILP_Field*   last_field;
               int                 methods_count;
               ILP_general_function method[1];
          } asClass;
          struct asMethod {
               struct ILP_Class*   class_defining;
               char*               name;
               short               arity;
               short               index;
          } asMethod;
          struct asField {
               struct ILP_Class*   defining_class;
               struct ILP_Field*   previous_field;
               char*               name;
               short               offset;
          } asField;
          struct asInstance {
               struct ILP_Object*  field[1];
          } asInstance;
     }                   _content;
} *ILP_Object;

/** On identifie ces structures car la sémantique de C ne permet de
 * faire d'allocations statiques pour une variante d'union (autre que
 * la première). */

typedef struct ILP_Exception {
     struct ILP_Class* _class;
     union {
          struct asException_ {
               char              message[ILP_EXCEPTION_BUFFER_LENGTH];
               struct ILP_Object*  culprit[ILP_EXCEPTION_CULPRIT_LENGTH];
          } asException;
     }                   _content;
} *ILP_Exception;

typedef struct ILP_Class {
     struct ILP_Class* _class;
     union {
```



```
          struct asClass_ {
               struct ILP_Class*   super;
               char*               name;
               int                 fields_count;
               struct ILP_Field*   last_field;
               int                 methods_count;
               ILP_general_function method[2];
          } asClass;
     }                   _content;
} *ILP_Class;

typedef struct ILP_Method {
     struct ILP_Class* _class;
     union {
          struct asMethod_ {
               struct ILP_Class*   class_defining;
               char*               name;
               short               arity;
               short               index;
          } asMethod;
     }                   _content;
} *ILP_Method;

typedef struct ILP_Field {
     struct ILP_Class* _class;
     union {
          struct asField_ {
               struct ILP_Class*   defining_class;
               struct ILP_Field*   previous_field;
               char*               name;
               short               offset;
          } asField;
     }                   _content;
} *ILP_Field;

/** Engendrer le type des classes à i méthodes.
 *
 * C'est nécessaire car gcc maintenant supprime les initialisations
 * superflues. Il faut donc soit allouer et initialiser dynamiquement
 * les classes soit créer autant de types que nécessaires (comment
 * éviter les doublons ?)
 */

#define ILP_GenerateClass(i) \
typedef struct ILP_Class##i {                                        \
     struct ILP_Class* _class;                                       \
     union {                                                         \
          struct asClass_##i {                                       \
               struct ILP_Class*   super;                            \
               char*               name;                             \
               int                 fields_count;                     \
               struct ILP_Field*   last_field;                       \
               int                 methods_count;                    \
               ILP_general_function method[i];                       \
          } asClass;                                                 \
     }                   _content;                                   \
} *ILP_Class##i

#define ILP_FindAndCallSuperMethod(i) \
  (((ilp_SuperMethod != NULL) \
    ? (*ILP_find_and_call_super_method##i) \
    : (*ILP_dont_call_super_method) )( \
   ilp_Self, ilp_CurrentMethod, ilp_SuperMethod, ilp_CurrentArguments))

extern ILP_Object ILP_find_and_call_super_method0(
     ILP_Object self,
     ILP_Method current_method,
     ILP_general_function super_method,
     ILP_Object arguments[] );
extern ILP_Object ILP_find_and_call_super_method1(
     ILP_Object self,
     ILP_Method current_method,
     ILP_general_function super_method,
     ILP_Object arguments[] );
extern ILP_Object ILP_find_and_call_super_method2(
     ILP_Object self,
     ILP_Method current_method,
     ILP_general_function super_method,
     ILP_Object arguments[] );
extern ILP_Object ILP_find_and_call_super_method3(
```

```
      ILP_Object self,
      ILP_Method current_method,
      ILP_general_function super_method,
      ILP_Object arguments[] );
extern ILP_Object ILP_dont_call_super_method(
      ILP_Object self,
      ILP_Method current_method,
      ILP_general_function super_method,
      ILP_Object arguments[] );

/** ------------------------------------------
 * Des macros pour manipuler toutes ces valeurs.
 */

#define ILP_IsA(o,c) \
  ILP_is_a(o, c)

/** Booléens. */

#define ILP_Boolean2ILP(b) \
  ILP_make_boolean(b)

#define ILP_isBoolean(o) \
  ((o)->_class == &ILP_object_Boolean_class)

#define ILP_isTrue(o) \
  (((o)->_class == &ILP_object_Boolean_class) && \
   ((o)->_content.asBoolean))

#define ILP_TRUE  (&ILP_object_true)
#define ILP_FALSE (&ILP_object_false)

#define ILP_isEquivalentToTrue(o) \
  ((o) != ILP_FALSE)

#define ILP_CheckIfBoolean(o) \
  if ( ! ILP_isBoolean(o) ) { \
       ILP_domain_error("Not a boolean", o); \
  };

/** Entiers */

#define ILP_Integer2ILP(i) \
  ILP_make_integer(i)

#define ILP_AllocateInteger() \
  ILP_malloc(sizeof(struct ILP_Object), &ILP_object_Integer_class)

#define ILP_isInteger(o) \
  ((o)->_class == &ILP_object_Integer_class)

#define ILP_CheckIfInteger(o) \
  if ( ! ILP_isInteger(o) ) { \
       ILP_domain_error("Not an integer", o); \
  };

/** Flottants */

#define ILP_Float2ILP(f) \
  ILP_make_float(f)

#define ILP_AllocateFloat() \
  ILP_malloc(sizeof(struct ILP_Object), &ILP_object_Float_class)

#define ILP_isFloat(o) \
  ((o)->_class == &ILP_object_Float_class)

#define ILP_CheckIfFloat(o) \
  if ( ! ILP_isFloat(o) ) { \
       ILP_domain_error("Not a float", o); \
  };

#define ILP_PI_VALUE 3.1415926535
#define ILP_PI (ILP_pi())

/** Chaînes de caractères */

#define ILP_String2ILP(s) \
  ILP_make_string(s)
```



```
#define ILP_AllocateString(length) \
  ILP_malloc(sizeof(struct ILP_Object) \
             + (sizeof(char) * (length)), &ILP_object_String_class)

#define ILP_isString(o) \
  ((o)->_class == &ILP_object_String_class)

#define ILP_CheckIfString(o) \
  if ( ! ILP_isString(o) ) { \
       ILP_domain_error("Not a string", o); \
  };

/** Opérateurs unaires */

#define ILP_Opposite(o) \
  ILP_make_opposite(o)

#define ILP_Not(o) \
  ILP_make_negation(o)

/** Opérateurs binaires */

#define ILP_Plus(o1,o2) \
  ILP_make_addition(o1, o2)

#define ILP_Minus(o1,o2) \
  ILP_make_subtraction(o1, o2)

#define ILP_Times(o1,o2) \
  ILP_make_multiplication(o1, o2)

#define ILP_Divide(o1,o2) \
  ILP_make_division(o1, o2)

#define ILP_Modulo(o1,o2) \
  ILP_make_modulo(o1, o2)

#define ILP_LessThan(o1,o2) \
  ILP_compare_less_than(o1,o2)

#define ILP_LessThanOrEqual(o1,o2) \
  ILP_compare_less_than_or_equal(o1,o2)

#define ILP_GreaterThan(o1,o2) \
  ILP_compare_greater_than(o1,o2)

#define ILP_GreaterThanOrEqual(o1,o2) \
  ILP_compare_greater_than_or_equal(o1,o2)

#define ILP_Equal(o1,o2) \
  ILP_compare_equal(o1,o2)

#define ILP_NotEqual(o1,o2) \
  ILP_compare_not_equal(o1,o2)

/** Constantes de classes. */

extern struct ILP_Class ILP_object_Object_class;
extern struct ILP_Class ILP_object_Class_class;
extern struct ILP_Class ILP_object_Method_class;
extern struct ILP_Class ILP_object_Field_class;
extern struct ILP_Class ILP_object_Integer_class;
extern struct ILP_Class ILP_object_Float_class;
extern struct ILP_Class ILP_object_Boolean_class;
extern struct ILP_Class ILP_object_String_class;
extern struct ILP_Class ILP_object_Exception_class;
extern struct ILP_Field ILP_object_super_field;
extern struct ILP_Field ILP_object_defining_class_field;
extern struct ILP_Method ILP_object_print_method;
extern struct ILP_Method ILP_object_classOf_method;

/** Primitives. */

extern struct ILP_Object ILP_object_true;
extern struct ILP_Object ILP_object_false;
extern ILP_Object ILP_die (char *message);
extern ILP_Object ILP_make_boolean (int b);
extern ILP_Object ILP_make_integer (int d);
```

```
extern ILP_Object ILP_make_float (double d);
extern ILP_Object ILP_pi ();
extern ILP_Object ILP_make_string (char *s);
extern ILP_Object ILP_make_opposite (ILP_Object o);
extern ILP_Object ILP_make_negation (ILP_Object o);
extern ILP_Object ILP_make_addition (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_make_subtraction (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_make_multiplication (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_make_division (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_make_modulo (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_less_than (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_less_than_or_equal (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_equal (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_greater_than (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_greater_than_or_equal (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_compare_not_equal (ILP_Object o1, ILP_Object o2);
extern ILP_Object ILP_newline ();
extern ILP_Object ILP_print (ILP_Object self);
extern ILP_Object ILP_classOf (ILP_Object self);

extern ILP_Object ILP_malloc (int size, ILP_Class class);
extern ILP_Object ILP_make_instance (ILP_Class class);
extern int /* boolean */ ILP_is_a (ILP_Object o, ILP_Class class);
extern ILP_general_function ILP_find_method (ILP_Object receiver,
                                             ILP_Method method,
                                             int argc);

/** Mecanisme d'allocation */

#ifdef WITH_GC
   /* Le GC de Boehm se trouve là: */
#  include "include/gc.h"
#  define ILP_START_GC GC_init()
#  define ILP_MALLOC GC_malloc
#else
#  define ILP_START_GC
#  define ILP_MALLOC malloc
#endif

/** Exceptions. */

struct ILP_catcher {
     struct ILP_catcher *previous;
     jmp_buf _jmp_buf;
};

extern struct ILP_catcher *ILP_current_catcher;
extern ILP_Object ILP_current_exception;
extern ILP_Object ILP_throw (ILP_Object exception);
extern void ILP_establish_catcher (struct ILP_catcher *new_catcher);
extern void ILP_reset_catcher (struct ILP_catcher *catcher);
extern ILP_Object ILP_error (char *message);
extern ILP_Object ILP_domain_error (char *message, ILP_Object o);

#define ILP_UnknownFieldError(f, o) \
  ILP_domain_error("Unfound field " f, o);

#endif /* ILPOBJ_H */

/* end of ilpObj.h */
```

Java/src/fr/upmc/ilp/ilp6/Process.java

```
package fr.upmc.ilp.ilp6;

import java.io.IOException;

import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp2.runtime.ConstantsStuff;
import fr.upmc.ilp.ilp3.ThrowPrimitive;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.ast.NormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp4.runtime.LexicalEnvironment;
import fr.upmc.ilp.ilp4.runtime.PrintStuff;
import fr.upmc.ilp.ilp6.ast.CEAST6;
import fr.upmc.ilp.ilp6.ast.CEAST6Factory;
```



```
import fr.upmc.ilp.ilp6.ast.CEASTParser;
import fr.upmc.ilp.ilp6.ast.NormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.cgen.CgenEnvironment6;
import fr.upmc.ilp.ilp6.cgen.CgenLexicalEnvironment6;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.runtime.CommonPlus;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.tool.FileTool;
import fr.upmc.ilp.tool.IFinder;
import fr.upmc.ilp.tool.ProgramCaller;

/** Cette classe pr?cise comment est trait? un programme d'ILP6. */

public class Process extends fr.upmc.ilp.ilp4.Process {

    /** Un constructeur utilisant toutes les valeurs par defaut possibles. */

    public Process (IFinder finder) throws IOException {
        super(finder); // pour m?moire!
        setGrammar(getFinder().findFile("grammar6.rng"));
        IAST6Factory factory = new CEAST6Factory();
        setFactory(factory);
        setParser(new CEASTParser(factory));
    }

    /** Profitons de la covariance! */
    @Override
    public IAST6program getCEAST () {
        return CEAST6.narrowToIAST6program(super.getCEAST());
    }
    @Override
    public IAST6Factory getFactory () {
        return CEAST6.narrowToIAST6Factory(super.getFactory());
    }

    /** Initialisation: @see fr.upmc.ilp.tool.AbstractProcess. */

    /** Pr?paration (heritee) */

    /** Normalisation */

    @Override
    public IAST6program performNormalization()
    throws NormalizeException {
        final IAST6Factory factory = getFactory();
        final INormalizeLexicalEnvironment normlexenv =
            new NormalizeLexicalEnvironment.EmptyNormalizeLexicalEnvironment();
        final INormalizeGlobalEnvironment normcommon =
            new NormalizeGlobalEnvironment();
        normcommon.addPrimitive(factory.newGlobalVariable("print"));
        normcommon.addPrimitive(factory.newGlobalVariable("newline"));
        normcommon.addPrimitive(factory.newGlobalVariable("throw"));
        final IAST6program program =
            getCEAST().normalize6(normlexenv, normcommon, factory);
        return program;
    }

    /** Interpretation */

    @Override
    public void interpret() {
        try {
            assert this.prepared;
            final ICommon intcommon = new CommonPlus();
            intcommon.bindPrimitive("throw", ThrowPrimitive.create());
            final ILexicalEnvironment intlexenv =
                LexicalEnvironment.EmptyLexicalEnvironment.create();
            final PrintStuff intps = new PrintStuff();
            intps.extendWithPrintPrimitives(intcommon);
            final ConstantsStuff csps = new ConstantsStuff();
            csps.extendWithPredefinedConstants(intcommon);

            this.result = getCEAST().eval6(intlexenv, intcommon);
            this.printing = intps.getPrintedOutput().trim();

            this.interpreted = true;
```

```
95
        } catch (Throwable e) {
            this.interpretationFailure = e;
        }
    }
100
    /** Compilation vers C. */

    @Override
    public void compile() {
105     try {
            assert this.prepared;
            ICgenEnvironment common = new CgenEnvironment6();
            common.bindPrimitive("throw");
            ICgenLexicalEnvironment lexenv =
110             CgenLexicalEnvironment6.CgenEmptyLexicalEnvironment6.create();
            this.ccode = getCEAST().compile6(lexenv, common);

            this.compiled = true;

115     } catch (Throwable e) {
            this.compilationFailure = e;
        }
    }

120 /** Ex?cution du programme compil?: */

    @Override
    public void runCompiled() {
        try {
125         assert this.compiled;
            assert this.cFile != null;
            assert this.compileThenRunScript != null;
            FileTool.stuffFile(this.cFile, this.ccode);

130         // Optionnel: mettre en forme le programme:
            String indentProgram = "indent -npcs " + this.cFile.getAbsolutePath();
            ProgramCaller pcindent = new ProgramCaller(indentProgram);
            pcindent.run();

135         // et le compiler:
            String program = "bash "
                + this.compileThenRunScript.getAbsolutePath() + " "
                + " +gc "
                + this.cFile.getAbsolutePath()
140             + " C/ilpObj.o ";
            ProgramCaller pc = new ProgramCaller(program);
            pc.setVerbose();
            pc.run();
            this.executionPrinting = pc.getStdout().trim();
145
            this.executed = ( pc.getExitValue() == 0 );

        } catch (Throwable e) {
            this.executionFailure = e;
150     }
    }

}
```

Java/src/fr/upmc/ilp/ilp6/test/ProcessTest.java

```
1  package fr.upmc.ilp.ilp6.test;

   import java.io.IOException;
   import java.util.Collection;

6  import org.junit.Before;
   import org.junit.runner.RunWith;

   import fr.upmc.ilp.ilp1.test.AbstractProcessTest;
   import fr.upmc.ilp.ilp6.Process;
11 import fr.upmc.ilp.tool.File;
   import fr.upmc.ilp.tool.Parameterized;
   import fr.upmc.ilp.tool.Parameterized.Parameters;

   @RunWith(value=Parameterized.class)
```



```
16 public class ProcessTest
   extends fr.upmc.ilp.ilp4.test.ProcessTest {

       /** Le constructeur du test sur un fichier. */

21     public ProcessTest (final File file) {
           super(file);
       }

       @Before
26     @Override
       public void setUp () throws IOException {
           this.setProcess(new Process(finder));
           getProcess().setVerbose(options.verbose);
       }
31
       @Parameters
       public static Collection<File[]> data() throws Exception {
           initializeFromOptions();
           AbstractProcessTest.staticSetUp(samplesDir, "u\\d+-[1-46]");
36         // Pour un (ou plusieurs) test(s) en particulier:
           //AbstractProcessTest.staticSetUp("u35-1");
           return AbstractProcessTest.collectData();
       }
   }
```

Java/src/fr/upmc/ilp/ilp6/test/WholeTestSuite.java

```
   package fr.upmc.ilp.ilp6.test;

   import org.junit.runner.RunWith;
   import org.junit.runners.Suite;
5  import org.junit.runners.Suite.SuiteClasses;

   /** Regroupement de classes de tests pour le paquetage ilp6. */

   @RunWith(value=Suite.class)
10 @SuiteClasses(value={
           // Tous les fichiers de tests un par un:
           fr.upmc.ilp.ilp6.test.ProcessTest.class
   })
   public class WholeTestSuite {}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6Factory.java

```
   package fr.upmc.ilp.ilp6.interfaces;

   import fr.upmc.ilp.ilp4.interfaces.IAST4Factory;
   import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
5  import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
   import fr.upmc.ilp.ilp4.interfaces.IAST4globalFunctionVariable;
   import fr.upmc.ilp.ilp4.interfaces.IAST4variable;

   public interface IAST6Factory
10 extends IAST4Factory {
       IAST6classDefinition newClassDefinition(
               String className,
               String superClassName,
               String[] fieldNames,
15             IAST6methodDefinition[] methodDefinitions );
       IAST6methodDefinition newMethodDefinition(
               IAST4functionDefinition function);
       IAST6methodDefinition newMethodDefinition(
               String methodName,
20             IAST4globalFunctionVariable gfv,
               IAST4variable[] variables,
               IAST4expression body  );
       IAST6instantiation newInstantiation(
               String className,
25             IAST4expression[] arguments );
       IAST6readField newReadField(
               String fieldName,
               IAST4expression object );
       IAST6writeField newWriteField(
30             String fieldName,
               IAST4expression object,
```

```
            IAST4expression value );
    IAST6self newSelf(IAST4variable variable);
    IAST6send newSend(
            String message,
            IAST4expression receiver,
            IAST4expression[] arguments );
    IAST6super newSuper();
    IAST6program newProgram(IAST4functionDefinition[] definitions,
                            IAST6classDefinition[] clazzes,
                            IAST4expression body);
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6classDefinition.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;

public interface IAST6classDefinition
extends IAST4, IAST6visitable {

    // Les caracteristiques propres a la classe

    /** Le nom de la classe. */

    String getName ();

    /** Le nom de la super-classe. */

    String getSuperClassName ();

    /** Les noms des champs propres introduits par la classe
     * et non pas herites. */

    String[] getProperFieldNames ();

    /** Les noms des methodes propres definies par la classe
     * et non heritees. */

    String[] getProperMethodNames ();

    /** Les definitions des methodes propres introduites par
     * la classe et non heritees. */

    IAST6methodDefinition[] getProperMethodDefinitions ();

    // Les caracteristiques heritees necessite la presence d'une
    // structure de memorisation.

    /** Les noms de tous les champs propres ou herites. */

    String[] getFieldNames (IClassEnvironment common);

    /** La definition d'une methode nommee. */

    IAST6methodDefinition getMethodDefinition (
            String name,
            IClassEnvironment common );

    int fieldSize (IClassEnvironment common);

    int inheritedFieldSize (IClassEnvironment common);

    int getFieldOffset (String fieldName, IClassEnvironment common);

    int getMethodOffset (String methodName, IClassEnvironment common)
    throws CgenerationException;

    int getNumberOfInheritedMethods (IClassEnvironment common);

    int getMethodsCount (IClassEnvironment common);

    // Compilation

    void compileHeader (StringBuffer buffer,
```



```
                        ICgenLexicalEnvironment lexenv,
                        ICgenEnvironment common )
    throws CgenerationException;

    void compile (StringBuffer buffer,
                  ICgenLexicalEnvironment lexenv,
                  ICgenEnvironment common )
    throws CgenerationException;

    /** Compilation de toutes les methodes. */

    void compileMethodsTable (StringBuffer buffer,
                              ICgenEnvironment common,
                              IAST6classDefinition clazz)
    throws CgenerationException;

    IAST6classDefinition normalize (
            INormalizeLexicalEnvironment lexenv,
            INormalizeGlobalEnvironment common,
            IAST6Factory factory )
      throws NormalizeException;
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6expression.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;

public interface IAST6expression extends IAST4expression {
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6instantiation.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;

public interface IAST6instantiation
extends IAST6visitable, IAST6expression {
    String getClassName ();
    IAST4expression[] getArguments ();
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6methodDefinition.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;

public interface IAST6methodDefinition
extends IAST4functionDefinition, IAST6visitable {

    String getMethodName ();
    void setDefiningClass (IAST6classDefinition classDefinition);
    int getRealArity ();
}

// end of IAST6methodDefinition.java
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6program.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4program;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
```

```
import fr.upmc.ilp.ilp6.runtime.ICommon;

public interface IAST6program
extends IAST4program {

    IAST6classDefinition[] getClassDefinitions ();

    Object eval6 (ILexicalEnvironment lexenv, ICommon common)
    throws EvaluationException;

    String compile6 (ICgenLexicalEnvironment lexenv,
                     ICgenEnvironment common )
    throws CgenerationException;

    IAST6program normalize6 (
            final INormalizeLexicalEnvironment lexenv,
            final INormalizeGlobalEnvironment common,
            final IAST6Factory factory )
      throws NormalizeException;
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6readField.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;

public interface IAST6readField
extends IAST6visitable, IAST6expression {
    IAST4expression getObject ();
    String getFieldName ();
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6self.java

```
package fr.upmc.ilp.ilp6.interfaces;

public interface IAST6self
extends IAST6expression, IAST6visitable {
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6send.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;

public interface IAST6send extends IAST6visitable, IAST6expression {
    String getMethodName ();
    IAST4expression getReceiver ();
    IAST4expression[] getArguments ();
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6super.java

```
package fr.upmc.ilp.ilp6.interfaces;

public interface IAST6super extends IAST6visitable, IAST6expression {
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6visitable.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4visitable;

public interface IAST6visitable extends IAST4visitable {

    <Data, Result, Exc extends Throwable> Result accept (
            IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc;
}
```


Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6visitor.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;

public interface IAST6visitor<Data, Result, Exc extends Throwable>
 extends IAST4visitor<Data, Result, Exc> {
    Result visit (IAST6classDefinition classDefinition, Data data) throws Exc;
    Result visit (IAST6methodDefinition methodDefinition, Data data) throws Exc;
    Result visit (IAST6instantiation expression, Data data) throws Exc;
    Result visit (IAST6send expression, Data data) throws Exc;
    Result visit (IAST6readField expression, Data data) throws Exc;
    Result visit (IAST6writeField expression, Data data) throws Exc;
    Result visit (IAST6self expression, Data data) throws Exc;
    Result visit (IAST6super expression, Data data) throws Exc;
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IAST6writeField.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;

public interface IAST6writeField
extends IAST6visitable, IAST6expression {
    IAST4expression getObject ();
    String getFieldName ();
    IAST4expression getValue ();
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/ICgenEnvironment.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;

/** L'interface décrivant l'environnement des opérateurs prédéfinis du
 * langage à compiler vers C. Il est l'analogue de runtime/ICommon
 * pour le paquetage cgen. */

public interface ICgenEnvironment
extends fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment,
        IClassEnvironment {

  /** Rechercher l'index associé à une méthode. */

  public int getMethodOffset (String className, String methodName)
    throws CgenerationException;

  /** Rechercher la classe ayant introduit un champ.
   *
   * NOTA: C'est pour assurer qu'elle est unique que l'on restreint
   * les noms des champs. */

  public IAST6classDefinition findDefiningClassDefinition (String fieldName)
    throws CgenerationException;

  /** Mémoriser si un appel approprié à ILP_GenerateClass a déjà été
   * engendré. */

  public boolean alreadyGeneratedClassMacro (int methodCount);

  /** Engendrer l'objet représentant une méthode si elle n'a pas déjà
   * été engendrée! */

  public boolean alreadyGeneratedMethodObject (String methodName);

}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IClassEnvironment.java

```
package fr.upmc.ilp.ilp6.interfaces;

/** Cette interface precise qu'un environnement travaillant avec des
 * classes doit pouvoir les memoriser et les rechercher.  */

public interface IClassEnvironment {

    /** Enregistrer une définition de classe. */

    public void addClassDefinition (IAST6classDefinition cd);

    /** Rechercher la définition d'une classe. Comme une classe peut manquer
     * aussi bien pendant la normalisation que la compilation (ou toute autre
     * phase) on signale ce probleme par une RuntimeException. */

    public IAST6classDefinition findClassDefinition (String className)
      throws RuntimeException;

}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/INormalizeGlobalEnvironment.java

```
package fr.upmc.ilp.ilp6.interfaces;

/** Normaliser les variables globales veut dire utiliser un unique
 * objet pour toutes les références à une variable globale. Il est
 * ainsi possible de partager simplement de l'information sur cette
 * variable globale depuis tous les endroits où elle est référencée.
 */

public interface INormalizeGlobalEnvironment
extends fr.upmc.ilp.ilp4.interfaces.INormalizeGlobalEnvironment,
        IClassEnvironment {
}
```

Java/src/fr/upmc/ilp/ilp6/interfaces/IParser6.java

```
package fr.upmc.ilp.ilp6.interfaces;

import fr.upmc.ilp.ilp4.interfaces.IParser;

public interface IParser6 extends IParser {
    IAST6Factory getFactory ();
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEAST6.java

```
package fr.upmc.ilp.ilp6.ast;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.interfaces.IAST4;
import fr.upmc.ilp.ilp4.interfaces.IAST4Factory;
import fr.upmc.ilp.ilp4.interfaces.IAST4program;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.runtime.ICommon;

public abstract class CEAST6
extends fr.upmc.ilp.ilp4.ast.CEAST
implements IAST4 {

    /**
     * Pratique en Eclipse! Ainsi, dans la perspective de mise au point,
     * la valeur d'un CEASTprogram s'affichera de maniere plus lisible. Il
     * egalement possible de positionner (menu contextuel: edit detail
```



```
     * formatter) sur la variable Process.ceast qu'on veut la voir s'afficher
     * avec: "return new XMLwriter().process(this);". Cette meme astuce doit
     * fonctionner avec toute instance d'IAST4.
     */
    @Override
    public String toString () {
        try {
            if ( xmlwriter == null ) {
                xmlwriter = new XMLwriter();
            }
            return xmlwriter.process(this);
        } catch (Throwable t) {
            return super.toString();
        }
    }
    private static XMLwriter xmlwriter;

    @Override
    public Object eval (final ILexicalEnvironment lexenv,
                        final fr.upmc.ilp.ilp2.interfaces.ICommon common)
    throws EvaluationException {
        return eval6(
                lexenv,
                CEAST6.narrowToICommon(common) );
    }
    public abstract Object eval6 (final ILexicalEnvironment lexenv,
                                  final ICommon common)
    throws EvaluationException;

    public void compile (
            final StringBuffer buffer,
            final ICgenLexicalEnvironment lexenv,
            final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common )
      throws CgenerationException {
        compile6(buffer,
                 lexenv,
                 CEAST6.narrowToICgenEnvironment(common) );
    }
    public void compile (
            final StringBuffer buffer,
            final ICgenLexicalEnvironment lexenv,
            final ICgenEnvironment common )
      throws CgenerationException {
        compile6(buffer, lexenv, common);
    }
    public abstract void compile6 (final StringBuffer buffer,
                                   final ICgenLexicalEnvironment lexenv,
                                   final ICgenEnvironment common )
    throws CgenerationException;

    public void compileHeader (
            final StringBuffer buffer,
            final ICgenLexicalEnvironment lexenv,
            final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common )
      throws CgenerationException {
        compileHeader6(buffer,
                       lexenv,
                       CEAST6.narrowToICgenEnvironment(common) );
    }
    public void compileHeader (
            final StringBuffer buffer,
            final ICgenLexicalEnvironment lexenv,
            final ICgenEnvironment common )
    throws CgenerationException {
        compileHeader6(buffer, lexenv, common);
    }
    public abstract void compileHeader6 (
            final StringBuffer buffer,
            final ICgenLexicalEnvironment lexenv,
            final ICgenEnvironment common )
    throws CgenerationException;

    // Quelques retrecisseurs:

    public static <Data, Result, Exc extends Throwable> IAST6visitor<Data, Result, Exc>
    narrowToIAST6visitor (IAST4visitor<Data, Result, Exc> visitor) {
        if ( visitor instanceof IAST6visitor<?,?,?> ) {
```

```
            return (IAST6visitor<Data, Result, Exc>) visitor;
        } else {
            final String msg = "Not an IAST6visitor: " + visitor;
            throw new ClassCastException(msg);
        }
    }
    public static IAST6methodDefinition narrowToIAST6methodDefinition (
            IAST4 o ) {
        if ( o instanceof IAST6methodDefinition ) {
            return (IAST6methodDefinition) o;
        } else {
            final String msg = "Not an IAST6methodDefinition: " + o;
            throw new ClassCastException(msg);
        }
    }
    public static ICommon narrowToICommon (
            fr.upmc.ilp.ilp2.interfaces.ICommon o) {
        if ( o instanceof ICommon ) {
            return (ICommon) o;
        } else {
            final String msg = "Not an ICommon6: " + o;
            throw new ClassCastException(msg);
        }
    }
    public static IAST6Factory narrowToIAST6Factory (
            IAST4Factory f ) {
        if ( f instanceof IAST6Factory ) {
            return (IAST6Factory) f;
        } else {
            final String msg = "Not an IAST6Factory: " + f;
            throw new ClassCastException(msg);
        }
    }
    public static IAST6program narrowToIAST6program (IAST4program f ) {
        if ( f instanceof IAST6program) {
            return (IAST6program) f;
        } else {
            final String msg = "Not an IAST6program: " + f;
            throw new ClassCastException(msg);
        }
    }
    public static INormalizeGlobalEnvironment
        narrowToINormalizeGlobalEnvironment (
            fr.upmc.ilp.ilp4.interfaces.INormalizeGlobalEnvironment o) {
        if ( o instanceof INormalizeGlobalEnvironment) {
            return (INormalizeGlobalEnvironment) o;
        } else {
            final String msg = "Not an INormalizeGlobalEnvironment6: " + o;
            throw new ClassCastException(msg);
        }
    }
    public static ICgenLexicalEnvironment narrowToICgenLexicalEnvironment (
            fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment o) {
        return o;
    }
    public static ICgenEnvironment narrowToICgenEnvironment (
            fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment o) {
        if ( o instanceof ICgenEnvironment) {
            return (ICgenEnvironment) o;
        } else {
            final String msg = "Not an ICgenEnvironment6: " + o;
            throw new ClassCastException(msg);
        }
    }
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEAST6Factory.java

```
package fr.upmc.ilp.ilp6.ast;

import fr.upmc.ilp.ilp4.ast.CEASTFactory;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalFunctionVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
```



```
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6instantiation;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.IAST6readField;
import fr.upmc.ilp.ilp6.interfaces.IAST6self;
import fr.upmc.ilp.ilp6.interfaces.IAST6send;
import fr.upmc.ilp.ilp6.interfaces.IAST6super;
import fr.upmc.ilp.ilp6.interfaces.IAST6writeField;

public class CEAST6Factory extends CEASTFactory
implements IAST6Factory {

    public IAST6program newProgram(
            IAST4functionDefinition[] definitions,
            IAST6classDefinition[] clazzes,
            IAST4expression body ) {
        return new CEASTprogram(definitions, clazzes, body);
    }

    public IAST6classDefinition newClassDefinition(
            String className,
            String superClassName,
            String[] fieldNames,
            IAST6methodDefinition[] methodDefinitions) {
        return new CEASTclassDefinition(
                className, superClassName, fieldNames, methodDefinitions);
    }

    public IAST6instantiation newInstantiation(
            String className,
            IAST4expression[] arguments) {
        return new CEASTinstantiate(className, arguments);
    }

    public IAST6methodDefinition newMethodDefinition(
            IAST4functionDefinition function) {
        return new CEASTmethodDefinition(function);
    }
    public IAST6methodDefinition newMethodDefinition(
            String methodName,
            IAST4globalFunctionVariable gfv,
            IAST4variable[] variables,
            IAST4expression body  ) {
        return new CEASTmethodDefinition(methodName, gfv, variables, body);
    }

    public IAST6readField newReadField(
            String fieldName, IAST4expression object) {
        return new CEASTreadField(fieldName, object);
    }

    public IAST6self newSelf(IAST4variable variable) {
        return new CEASTself(variable);
    }

    public IAST6send newSend(
            String message,
            IAST4expression receiver,
            IAST4expression[] arguments) {
        return new CEASTsend(message, receiver, arguments);
        }

    public IAST6super newSuper() {
        return new CEASTsuper();
    }

    public IAST6writeField newWriteField(
            String fieldName,
            IAST4expression object,
            IAST4expression value) {
        return new CEASTwriteField(fieldName, object, value);
    }
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEAST6expression.java

```
package fr.upmc.ilp.ilp6.ast;
```

```
import java.util.Set;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.cgen.NoDestination;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4Factory;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6expression;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.runtime.ICommon;

/** Cette classe abstraite ne sert qu'a partager cette methode
 * que l'on n'aurait jamais du heriter. */

public abstract class CEAST6expression
extends fr.upmc.ilp.ilp4.ast.CEASTexpression
implements IAST6expression {

    /**
     * Pratique en Eclipse! Ainsi, dans la perspective de mise au point,
     * la valeur d'un CEASTprogram s'affichera de maniere plus lisible. Il
     * egalement possible de positionner (menu contextuel: edit detail
     * formatter) sur la variable Process.ceast qu'on veut la voir s'afficher
     * avec: "return new XMLwriter().process(this);". Cette meme astuce doit
     * fonctionner avec toute instance d'IAST4.
     */
    @Override
    public String toString () {
        try {
            if ( xmlwriter == null ) {
                xmlwriter = new XMLwriter();
            }
            return xmlwriter.process(this);
        } catch (Throwable t) {
            return super.toString();
        }
    }
    private static XMLwriter xmlwriter;

    @Override
    public void findGlobalVariables (
            final Set<IAST2variable> globalvars,
            final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
        return;
    }

    @Override
    public IAST4expression normalize (
            final INormalizeLexicalEnvironment lexenv,
            final fr.upmc.ilp.ilp4.interfaces.INormalizeGlobalEnvironment common,
            final IAST4Factory factory )
    throws NormalizeException {
        return normalize6(lexenv,
                CEAST6.narrowToINormalizeGlobalEnvironment(common),
                CEAST6.narrowToIAST6Factory(factory) );
    }

    public abstract IAST4expression normalize6 (
            final INormalizeLexicalEnvironment lexenv,
            final INormalizeGlobalEnvironment common,
            final IAST6Factory factory )
    throws NormalizeException;

    @Override
    public Object eval (final ILexicalEnvironment lexenv,
                        final fr.upmc.ilp.ilp2.interfaces.ICommon common)
    throws EvaluationException {
        return eval6(
                lexenv,
                CEAST6.narrowToICommon(common) );
    }
    public abstract Object eval6 (final ILexicalEnvironment lexenv,
                                  final ICommon common)
    throws EvaluationException;

    @Override
    public void compile (final StringBuffer buffer,
                         final ICgenLexicalEnvironment lexenv,
                         final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common,
                         final IDestination destination)
      throws CgenerationException {
        compile6(buffer,
                lexenv,
                CEAST6.narrowToICgenEnvironment(common),
                destination );
    }
    public abstract void compile6 (final StringBuffer buffer,
                                   final ICgenLexicalEnvironment lexenv,
                                   final ICgenEnvironment common,
                                   final IDestination destination )
    throws CgenerationException;

    @Override
    @Deprecated
    public void compileExpression (final StringBuffer buffer,
                                   final ICgenLexicalEnvironment lexenv,
                                   final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common,
                                   final IDestination destination )
    throws CgenerationException {
        this.compile6(buffer,
                      lexenv,
                      CEAST6.narrowToICgenEnvironment(common),
                      NoDestination.create());
    }

    @Override
    @Deprecated
    public void compileInstruction (final StringBuffer buffer,
                                    final ICgenLexicalEnvironment lexenv,
                                    final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common,
                                    final IDestination destination)
    throws CgenerationException {
        this.compile6(buffer,
                lexenv,
                CEAST6.narrowToICgenEnvironment(common),
                destination);
    }

    //public void accept (IAST4visitor visitor) {
    //    CEAST6.narrowToIAST6visitor(visitor).visit(this);
    // On ne visite pas les CEAST6expression!
    //}

}
```


Java/src/fr/upmc/ilp/ilp6/ast/CEASTParser.java

```
package fr.upmc.ilp.ilp6.ast;

import org.w3c.dom.Document;
import org.w3c.dom.Element;

import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.IParser6;

/** Transformer un document XML en un CEAST. */

public class CEASTParser
extends fr.upmc.ilp.ilp4.ast.CEASTParser
implements IParser6 {

    @Override
    public IAST6Factory getFactory () {
        return CEAST6.narrowToIAST6Factory(super.getFactory());
```

```
    }

    public CEASTParser (IAST6Factory factory) {
        super(factory);
        addParser("definitionClasse", CEASTclassDefinition.class);
        addParser("creationObjet", CEASTinstantiate.class);
        addParser("lectureChamp", CEASTreadField.class);
        addParser("envoiMessage", CEASTsend.class);
        addParser("ecritureChamp", CEASTwriteField.class);
        addParser("moi", CEASTself.class);
        addParser("appelSuper", CEASTsuper.class);
    }

    @Override
    public IAST6program parse (final Document d)
    throws CEASTparseException {
        final Element e = d.getDocumentElement();
        return CEASTprogram.parse(e, this);
    }
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTclassDefinition.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.List;
import java.util.Set;
import java.util.Vector;

import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEASTexpression;
import fr.upmc.ilp.ilp4.ast.CEASTglobalFunctionVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IClassEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPClass;
import fr.upmc.ilp.ilp6.runtime.ILPmethod;

/** Définir une classe pour ILP6. */

public class CEASTclassDefinition
extends CEAST6
implements IAST6classDefinition {

    // Ce constructeur est invoqué par parse():
  public CEASTclassDefinition (final String className,
                               final String superClassName,
                               final String[] fieldNames,
                               final IAST6methodDefinition[] methods ) {
      this.className = className;
      this.superClassName = superClassName;
      this.fieldNames = fieldNames;
```



```
      this.methods = this.adjoinSelfToMethods(methods);
      this.methodNames = new String[methods.length];
      for ( int i=0 ; i<methods.length ; i++ ) {
          this.methodNames[i] = this.methods[i].getFunctionName();
      }
  }
  private final String className;
  private final String superClassName;
  // les champs propres:
  private final String[] fieldNames;
  // Les noms originaux des méthodes propres:
  private final String[] methodNames;
  // Les définitions élaborées des fonctions implantant les méthodes
  // propres. Attention: elles ont changé de nom global et contiennent
  // self en première variable.
  private final IAST6methodDefinition[] methods;

  public String getName () {
    return this.className;
  }
  public String getSuperClassName () {
    return this.superClassName;
  }
  public String[] getProperMethodNames () {
    return this.methodNames;
  }
  @ILPexpression(isArray=true)
  public IAST6methodDefinition[] getProperMethodDefinitions () {
    return this.methods;
  }
  public String[] getProperFieldNames () {
    return this.fieldNames;
  }

  public static IAST6classDefinition parse (
          final Element e, final IParser6 parser)
  throws CEASTparseException {
      final String className = e.getAttribute("nom");
      final String superClassName = e.getAttribute("parent");
      try {
          final XPathExpression fieldPath =
              xPath.compile("./champs/champ");
          final NodeList nlFields = (NodeList)
              fieldPath.evaluate(e, XPathConstants.NODESET);
          final List<String> fieldNames = new Vector<>();
          for ( int i=0 ; i<nlFields.getLength() ; i++ ) {
              final Element n = (Element) nlFields.item(i);
              fieldNames.add(n.getAttribute("nom"));
          }

          final XPathExpression methodPath =
              xPath.compile("./methodes/methode");
          final NodeList nlMethods = (NodeList)
              methodPath.evaluate(e, XPathConstants.NODESET);
          final List<IAST6methodDefinition> methodDefinitions = new Vector<>();
          for ( int i=0 ; i<nlMethods.getLength() ; i++ ) {
              final Element method = (Element) nlMethods.item(i);
              final IAST6methodDefinition m =
                  CEASTmethodDefinition.parse(method, parser);
              methodDefinitions.add(m);
          }
          return parser.getFactory().newClassDefinition(
                  className,
                  superClassName,
                  fieldNames.toArray(EMPTY_STRING_ARRAY),
                  methodDefinitions.toArray(EMPTY_FUNCTION_ARRAY)
                  );
      } catch (XPathExpressionException e1) {
        throw new CEASTparseException(e1);
    }
  }
  private static final XPath xPath = XPathFactory.newInstance().newXPath();
  private static final IAST6methodDefinition[] EMPTY_FUNCTION_ARRAY =
      new IAST6methodDefinition[0];

  // Transformer une methode en une fonction prenant self en premiere variable.
```

```
    private IAST6methodDefinition[] adjoinSelfToMethods (
            final IAST6methodDefinition[] methods ) {
        IAST6methodDefinition[] methods_ = new IAST6methodDefinition[methods.length];
        for ( int i=0 ; i<methods.length ; i++ ) {
            methods_[i] = adjoinSelfToMethod(methods[i]);
        }
        return methods_;
    }

    private IAST6methodDefinition adjoinSelfToMethod (
            final IAST6methodDefinition method ) {
        // Self a peut-etre deja ete ajoute ?
        IAST4variable[] vars = method.getVariables();
        if ( vars.length > 0
          && vars[0] instanceof CEASTself.CEASTselfVariable ) {
            return method;
        }
        CEASTself.CEASTselfVariable selfVariable = new CEASTself.CEASTselfVariable();
        IAST4variable[] varsPlusSelf = new IAST4variable[1+vars.length];
        varsPlusSelf[0] = selfVariable;
        for ( int j=0 ; j<vars.length ; j++ ) {
            varsPlusSelf[j+1] = vars[j];
        }
        return new CEASTmethodDefinition(
                  method.getMethodName(),
                  method.getDefinedVariable(),
                  varsPlusSelf,
                  method.getBody() );
    }

    // Plus d'acces direct aux champs a partir d'ici.

    //public @OrNull IAST6classDefinition findSuperClass (
    //        final IClassEnvironment common) {
    //    return common.findClassDefinition(getName());
    //} // Simplifie le code ???

    public String[] getFieldNames (final IClassEnvironment common) {
      final List<String> result = new Vector<>();
      // recuperer (dans l'ordre) les champs hérités:
      if ( ! "Object".equals(getSuperClassName()) ) {
        final IAST6classDefinition superCD =
          common.findClassDefinition(getSuperClassName());
        for ( String fieldName : superCD.getFieldNames(common) ) {
            result.add(fieldName);
        }
      }
      // y ajouter les champs propres:
      for ( String fieldName : getProperFieldNames() ) {
        result.add(fieldName);
      }
      return (String[]) result.toArray(EMPTY_STRING_ARRAY);
    }
    private static final String[] EMPTY_STRING_ARRAY = new String[0];

    /** Chercher la définition d'une méthode (propre ou héritée). */

    public IAST6methodDefinition getMethodDefinition (
            final String name,
            final IClassEnvironment common) {
        final IAST6methodDefinition[] methods = getProperMethodDefinitions();
        final String[] methodNames = getProperMethodNames();
        // Chercher dans les méthodes propres:
        for ( int i = 0 ; i<methods.length ; i++ ) {
            if ( name.equals(methodNames[i]) ) {
                return methods[i];
            }
        }
        // ou dans la superclasse (récursivement):
        if ( "Object".equals(getSuperClassName()) ) {
            // Sont-ce les méthodes prédéfinies ?
            if ( "print".equals(name) ) {
                return printMethod;
            } else if ( "classOf".equals(name) ) {
                return classOfMethod;
            } else {
                final String msg = "No such method " + name;
                throw new RuntimeException(msg);
            }
```



```
        } else {
            final IAST6classDefinition superCD =
                common.findClassDefinition(getSuperClassName());
            return superCD.getMethodDefinition(name, common);
        }
    }
    protected static final IAST6methodDefinition printMethod;
    static {
        final CEASTself self = new CEASTself();
        printMethod = new CEASTmethodDefinition(
                "print",
                new CEASTglobalFunctionVariable("ILP_print"),
                new IAST4variable[]{ self.getLocalVariable() },
                self ); // corps inutile
    }
    protected static final IAST6methodDefinition classOfMethod;
    static {
        final IAST4variable self = new CEASTself().getLocalVariable();
        classOfMethod = new CEASTmethodDefinition(
                "classOf",
                new CEASTglobalFunctionVariable("ILP_classOf"),
                new IAST4variable[] { self },
                CEASTexpression.voidExpression() );  // corps inutile
    }

    /** Calculer l'offset d'un champ (hérité ou propre). */

    public int getFieldOffset (final String fieldName,
                               final IClassEnvironment common) {
        String[] fieldNames = getProperFieldNames();
        for ( int i = 0 ; i<fieldNames.length ; i++ ) {
            if ( fieldNames[i].equals(fieldName) ) {
                return i + this.inheritedFieldSize(common);
            }
        }
        if ( "Object".equals(getSuperClassName()) ) {
            throw new RuntimeException("No such field");
        } else {
            final IAST6classDefinition superCD =
                common.findClassDefinition(getSuperClassName());
            return superCD.getFieldOffset(fieldName, common);
        }
    }

    /** Nombre de champs propres ainsi qu'hérités de la classe. */

    public int fieldSize (final IClassEnvironment common) {
      return getProperFieldNames().length + this.inheritedFieldSize(common);
    }

    /** Nombre de champs hérités de la classe. */

    public int inheritedFieldSize (final IClassEnvironment common) {
        if ( "Object".equals(getSuperClassName()) ) {
            return 0;
        } else {
            final IAST6classDefinition superCD =
                common.findClassDefinition(getSuperClassName());
            return superCD.fieldSize(common);
        }
    }

    /** Calculer l'offset d'une méthode. À tout message est associé un
     * entier non unique pour le programme tout entier. On recherche si
     * cet entier a déjà été attribué sinon on en utilise un nouveau. */

    public int getMethodOffset (final String methodName,
                                final IClassEnvironment common)
    throws CgenerationException {
        // MOCHE ces constantes devraient être calculées!
        if ( "print".equals(methodName) ) {
            return 0;
        } else if ( "classOf".equals(methodName) ) {
            return 1;
        }
        // On recherche d'abord parmi les méthodes héritées:
        try {
            if ( "Object".equals(getSuperClassName()) ) {
                final String msg = "Nope";
```

```
            throw new CgenerationException(msg);
        } else {
            final IAST6classDefinition superCD =
                common.findClassDefinition(getSuperClassName());
            return superCD.getMethodOffset(methodName, common);
        }
    } catch (CgenerationException e) {
        // Pas héritée! On continue en séquence:
    }
    // Attribuer un nouveau numéro:
    String[] methodNames = getProperMethodNames();
    for ( int i = 0 ; i<methodNames.length ; i++ ) {
        if ( methodNames[i].equals(methodName) ) {
            return i + getNumberOfInheritedMethods(common);
        }
    }
    final String msg = "No such method " + methodName
        + " for " + getName();
    throw new CgenerationException(msg);
}

/** Calculer le nombre de méthodes héritées. */

public int getNumberOfInheritedMethods (final IClassEnvironment common) {
    if ( "Object".equals(getSuperClassName()) ) {
        // Peu robuste! cette constante devrait être calculée      FIXME
        return 2; // print et classOf
    } else {
        final IAST6classDefinition superCD =
            common.findClassDefinition(getSuperClassName());
        return superCD.getProperMethodNames().length
             + superCD.getNumberOfInheritedMethods(common);
    }
}

/** Calculer le nombre total de méthodes propres ainsi qu'héritées. */

public int getMethodsCount (final IClassEnvironment common) {
    return getProperMethodNames().length
         + getNumberOfInheritedMethods(common);
}

/** Interprétation. */

@Override
public Object eval6 (final ILexicalEnvironment lexenv,
                     final ICommon common)
throws EvaluationException {
    final ILPClass superClass = common.findClass(getSuperClassName());
    final IAST4functionDefinition[] methods = getProperMethodDefinitions();
    final ILPmethod[] functions = new ILPmethod[methods.length];
    for ( int i = 0 ; i<methods.length ; i++ ) {
        functions[i] = (ILPmethod) methods[i].eval(lexenv, common);
    }
    ILPClass clazz = new ILPClass(getName(),
                                  superClass,
                                  getProperFieldNames(),
                                  getProperMethodNames(),
                                  functions );
    common.addClass(clazz);
    return clazz;
}

/** Compilation du vecteur des méthodes. */

public void compileMethodsTable (final StringBuffer buffer,
                                 final ICgenEnvironment common,
                                 final IAST6classDefinition clazz)
  throws CgenerationException {
  // Citer les fonctions implantant les méthodes héritées:
  if ( "Object".equals(getSuperClassName()) ) {
      // Peu robuste! ces noms de méthodes devraient être calculées     FIXME
      compileMethodTableEntry(buffer, common, clazz, "print");
      compileMethodTableEntry(buffer, common, clazz, "classOf");
  } else {
      final IAST6classDefinition superCD =
          common.findClassDefinition(getSuperClassName());
      superCD.compileMethodsTable(buffer, common, clazz);
  }
```



```
  // puis les fonctions implantant les méthodes propres nouvelles:
  String[] methodNames = getProperMethodNames();
  for ( int i = 0 ; i<methodNames.length ; i++ ) {
      final int offset = this.getMethodOffset(methodNames[i], common);
      if ( offset >= this.getNumberOfInheritedMethods(common) ) {
          compileMethodTableEntry(buffer, common, clazz, methodNames[i]);
      }
  }
}

private void compileMethodTableEntry (final StringBuffer buffer,
                                      final ICgenEnvironment common,
                                      final IAST6classDefinition clazz,
                                      final String methodName)
  throws CgenerationException {
  IAST4functionDefinition function =
    clazz.getMethodDefinition(methodName, common);
  buffer.append(function.getDefinedVariable().getMangledName());
  buffer.append(",     /* ");
  buffer.append(methodName);
  buffer.append(" */\n   ");
}

/** En C la macro ILP_GenerateClass engendre le type d'une classe
 avec un certain nombre de méthodes. Ce type ne doit être spécifié
 qu'une seule fois. Il faut donc mémoriser si on a déjà émis ce code
 ou pas. */

private void compileGenerateClassMacro (final StringBuffer buffer,
                                        final ICgenEnvironment common)
  throws CgenerationException {
  if ( !common.alreadyGeneratedClassMacro(getMethodsCount(common)) ) {
    // Engendrer le type approprié de la classe à engendrer:
    buffer.append("\nILP_GenerateClass(");
    buffer.append(getMethodsCount(common));
    buffer.append(");\n");
  }
}

/** Pour les besoins de citation mutuelles, on déclare les classes,
 * champs et méthodes qui apparaissent en C sous forme de variables
 * globales. */

@Override
public void compileHeader6 (final StringBuffer buffer,
                            final ICgenLexicalEnvironment lexenv,
                            final ICgenEnvironment common)
  throws CgenerationException {
  this.compileGenerateClassMacro(buffer, common);
  // Déclarer la classe:
  buffer.append("extern struct ILP_Class");
  buffer.append(getMethodsCount(common));
  buffer.append(" ILP_object_");
  buffer.append(getName());
  buffer.append("_class;\n");
  // Déclarer ses champs propres:
  for ( String fieldName : getProperFieldNames() ) {
    buffer.append("extern struct ILP_Field ILP_object_");
    buffer.append(fieldName);
    buffer.append("_field;\n");
  }
  // Déclarer ses méthodes propres (si ce n'est déjà fait):
  for ( IAST4functionDefinition method : getProperMethodDefinitions() ) {
      method.compileHeader(buffer, lexenv, common);
  }
}

/** Compiler une définition de classe. */
@Override
public void compile6 (final StringBuffer buffer,
                      final ICgenLexicalEnvironment lexenv,
                      final ICgenEnvironment common )
  throws CgenerationException {
  // Engendrer le code des méthodes propres:
  final String[] methodNames = getProperMethodNames();
  final IAST6methodDefinition[] methods = getProperMethodDefinitions();
  for ( int i = 0 ; i<methods.length ; i++ ) {
    buffer.append("\n/* Classe ");
    buffer.append(getName());
```

```
      buffer.append(", méthode ");
      buffer.append(methodNames[i]);
      buffer.append(": */\n");
      methods[i].compile(buffer, lexenv, common);
    }
    // Engendrer les champs propres:
    String lastFieldName = "NULL";
    if ( ! "Object".equals(getSuperClassName()) ) {
      final IAST6classDefinition superCD =
        common.findClassDefinition(getSuperClassName());
      final String[] allSuperFields = superCD.getFieldNames(common);
      if ( allSuperFields.length >= 1 ) {
          // Que s'il y a au moins un champ herite! <Francois.Nizou@etu.upmc.fr>
          lastFieldName = "&ILP_object_"
              + allSuperFields[allSuperFields.length-1] + "_field";
      }
    }
    String[] fieldNames = getProperFieldNames();
    for ( int i = 0 ; i<fieldNames.length ; i++ ) {
      buffer.append("\nstruct ILP_Field ILP_object_");
      buffer.append(fieldNames[i]);
      buffer.append("_field = {\n  &ILP_object_Field_class,\n     { { ");
      buffer.append("(ILP_Class) &ILP_object_");
      buffer.append(getName());
      buffer.append("_class,\n   ");
      buffer.append(lastFieldName);
      buffer.append(",\n   \"");
      buffer.append(fieldNames[i]);
      buffer.append("\",\n    ");
      buffer.append(i + inheritedFieldSize(common));
      buffer.append(" } }\n};\n");
      lastFieldName = "&ILP_object_" + fieldNames[i] + "_field";
    }
    // Engendrer la classe:
    buffer.append("\nstruct ILP_Class");
    buffer.append(getMethodsCount(common));
    buffer.append(" ILP_object_");
    buffer.append(getName());
    buffer.append("_class = {\n  &ILP_object_Class_class,\n  { { ");
    buffer.append("(ILP_Class) &ILP_object_");
    buffer.append(getSuperClassName());
    buffer.append("_class,\n         \"");
    buffer.append(getName());
    buffer.append("\",\n         ");
    buffer.append(fieldSize(common));
    buffer.append(",\n         ");
    buffer.append(lastFieldName);
    buffer.append(",\n         ");
    buffer.append(getMethodsCount(common));
    buffer.append(",\n { ");
    this.compileMethodsTable(buffer, common, this);
    buffer.append(" } } }\n};\n");
    // Engendrer les objets correspondant aux méthodes propres:
    for ( int i = 0 ; i<methods.length ; i++ ) {
      if ( !common.alreadyGeneratedMethodObject(methodNames[i]) ) {
        buffer.append("\nstruct ILP_Method ILP_object_");
        buffer.append(methodNames[i]);
        buffer.append("_method = {\n  &ILP_object_Method_class,\n  { { ");
        buffer.append("(struct ILP_Class*) &ILP_object_");
        buffer.append(getName());
        buffer.append("_class,\n  \"");
        buffer.append(methodNames[i]);
        buffer.append("\",\n  ");
        buffer.append(methods[i].getVariables().length);
        buffer.append(",  /* arité */\n  ");
        buffer.append(getMethodOffset(methodNames[i], common));
        buffer.append(" /* offset */ \n    } }\n};\n");
      }
    }
  }

  @Override
  public void findGlobalVariables (
          final Set<IAST2variable> globalvars,
          final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
      IAST4functionDefinition[] methods = getProperMethodDefinitions();
      for ( int i = 0 ; i<methods.length ; i++ ) {
          methods[i].findGlobalVariables(globalvars, lexenv);
      }
  }
```



```
  public IAST6classDefinition normalize (
          final INormalizeLexicalEnvironment lexenv,
          final INormalizeGlobalEnvironment common,
          final IAST6Factory factory )
    throws NormalizeException {
    final IAST6methodDefinition[] methods = getProperMethodDefinitions();
    final IAST6methodDefinition[] methods_ =
        new IAST6methodDefinition[methods.length];
    for ( int i = 0 ; i<methods.length ; i++ ) {
        methods_[i] = CEAST6.narrowToIAST6methodDefinition(
                methods[i].normalize(lexenv, common, factory) );
    }
    final IAST6classDefinition newClass =
        factory.newClassDefinition(
                getName(),
                getSuperClassName(),
                getProperFieldNames(),
                methods_);
    return newClass;
  }

  public <Data, Result, Exc extends Throwable> Result
  accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return visitor.visit(this, data);
  }
  public <Data, Result, Exc extends Throwable> Result
  accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
  }
  // NOTE: double methode surchargee.
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTinstantiate.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.Set;

import org.w3c.dom.Element;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTlocalVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.cgen.AssignDestination;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6instantiation;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPClass;
import fr.upmc.ilp.ilp6.runtime.ILPinstance;

/** Créer un objet c'est-à-dire, en jargon, instantier une classe. */

public class CEASTinstantiate
extends CEAST6expression
implements IAST6instantiation {

  public CEASTinstantiate (final String className,
                           final IAST4expression[] arguments) {
    this.className = className;
```

```
    this.arguments = arguments;
  }
  private final String className;
  private final IAST4expression[] arguments;

  public String getClassName () {
      return this.className;
  }
  @ILPexpression(isArray=true)
  public IAST4expression[] getArguments () {
      return this.arguments;
  }

  public static IAST6instantiation parse (final Element e, final IParser6 parser)
  throws CEASTparseException {
      final String className = e.getAttribute("classe");
      final IAST4expression[] arguments =
          parser.parseList(e.getChildNodes())
          .toArray(new IAST4expression[0]);
      return parser.getFactory().newInstantiation(
              className, arguments);
  }

  // Plus d'acces direct aux champs a partir d'ici.

  @Override
  public Object eval6 (final ILexicalEnvironment lexenv,
                       final ICommon common)
    throws EvaluationException {
    final ILPClass clazz = common.findClass(getClassName());
    IAST4expression[] arguments = getArguments();
    final Object[] values = new Object[arguments.length];
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      values[i] = arguments[i].eval(lexenv, common);
    }
    return new ILPinstance(clazz, values);
  }

  @Override
  public void compile6 (final StringBuffer buffer,
                        final ICgenLexicalEnvironment lexenv,
                        final ICgenEnvironment common,
                        final IDestination destination)
    throws CgenerationException {
    buffer.append("{ ");
    final IAST4variable tempInstance = CEASTlocalVariable.generateVariable();
    tempInstance.compileDeclaration(buffer, lexenv, common);
    final IAST4expression[] arguments = getArguments();
    final IAST4variable[] tempvar = new IAST4variable[arguments.length];
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      tempvar[i] = CEASTlocalVariable.generateVariable();
      tempvar[i].compileDeclaration(buffer, lexenv, common);
    }
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      arguments[i].compile(buffer, lexenv, common,
                           new AssignDestination(tempvar[i]));
      buffer.append(";\n");
    }
    buffer.append(tempInstance.getMangledName());
    buffer.append(" = ILP_make_instance((ILP_Class) &ILP_object_");
    buffer.append(getClassName());
    buffer.append("_class);\n");
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      buffer.append(tempInstance.getMangledName());
      buffer.append("->_content.asInstance.field[");
      buffer.append(i);
      buffer.append("] = ");
      buffer.append(tempvar[i].getMangledName());
      buffer.append(";\n");
    }
    destination.compile(buffer, lexenv, common);
    buffer.append(tempInstance.getMangledName());
    buffer.append(";\n}\n");
  }

  @Override
  public void findGlobalVariables (
          final Set<IAST2variable> globalvars,
```



```
          final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
      final IAST4expression[] arguments = getArguments();
      for ( int i = 0 ; i<arguments.length ; i++ ) {
          arguments[i].findGlobalVariables(globalvars, lexenv);
      }
  }

  @Override
  public IAST6instantiation normalize6 (
          final INormalizeLexicalEnvironment lexenv,
          final INormalizeGlobalEnvironment common,
          final IAST6Factory factory )
    throws NormalizeException {
      final IAST4expression[] arguments = getArguments();
      IAST6classDefinition cd = common.findClassDefinition(getClassName());
      if ( arguments.length != cd.fieldSize(common) ) {
          final String msg = "Wrong constructor arity for " + getClassName()
          + " expected: " + cd.fieldSize(common)
          + " obtained: " + arguments.length;
          throw new NormalizeException(msg);
      }
    IAST4expression[] arguments_ = new IAST4expression[arguments.length];
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      arguments_[i] = CEAST.narrowToIAST4expression(
              arguments[i].normalize(lexenv, common, factory) );
    }
    return factory.newInstantiation(getClassName(), arguments_);
  }

  public <Data, Result, Exc extends Throwable> Result
  accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return visitor.visit(this, data);
  }
  public <Data, Result, Exc extends Throwable> Result
  accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
  }
  // NOTE: double methode surchargee.

  @Override
  public void compileExpression(StringBuffer buffer,
        ICgenLexicalEnvironment lexenv,
        fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common,
        IDestination destination) throws CgenerationException {
      compile6(buffer,
               lexenv,
               CEAST6.narrowToICgenEnvironment(common),
               destination );
  }

  @Override
  public void compileInstruction(StringBuffer buffer,
        ICgenLexicalEnvironment lexenv,
        fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common,
        IDestination destination) throws CgenerationException {
      compile6(buffer,
               lexenv,
               CEAST6.narrowToICgenEnvironment(common),
               destination );
  }
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTmethodDefinition.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.Set;

import org.w3c.dom.Element;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.annotation.OrNull;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.cgen.ReturnDestination;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
```

```java
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTfunctionDefinition;
import fr.upmc.ilp.ilp4.ast.CEASTglobalFunctionVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4Factory;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalFunctionVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4localVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.cgen.CgenMethodLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IClassEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPmethod;
import fr.upmc.ilp.tool.CStuff;

public class CEASTmethodDefinition
extends CEAST6
implements IAST6methodDefinition {

    // Ce constructeur est invoqué par parse():
    public CEASTmethodDefinition (IAST4functionDefinition delegate) {
        this.methodName = delegate.getFunctionName();
        IAST4variable[] vars = delegate.getVariables();
        // self n'est pas encore en tete des variables:
        this.selfVariable = null;
        // toutes les fonctions sous-jacentes aux methodes doivent avoir
        // des noms differents:
        CEASTglobalFunctionVariable gfv =
            new CEASTglobalFunctionVariable("ilpMETHOD_" + getCounter());
        this.delegate = new CEASTfunctionDefinition(
                gfv,
                vars,
                delegate.getBody() );
    }
    private final CEASTfunctionDefinition delegate;
    private final IAST4variable selfVariable;
    private final String methodName;

    protected synchronized int getCounter () {
        return counter++;
    }
    private static int counter = 1;

    // Ce constructeur est invoqué par CEASTclassDefinition.adjoinSelfToMethod
    // et par normalize().
    public CEASTmethodDefinition (
            String methodName,
            IAST4globalFunctionVariable gfv,
            IAST4variable[] variables,
            IAST4expression body ) {
        this.methodName = methodName;
        // On n'ajoute pas self en tete, c'est deja fait!
        this.selfVariable = variables[0];
        this.delegate = new CEASTfunctionDefinition(gfv, variables, body);
    }

    public String getFunctionName () {
        return this.methodName;
    }
    public String getMethodName () {
        return this.methodName;
    }
    public IAST4globalFunctionVariable getDefinedVariable () {
        return this.delegate.getDefinedVariable();
    }
    public IAST4variable[] getVariables() {
        return this.delegate.getVariables();
```



```java
    }
    @ILPexpression
    public IAST4expression getBody () {
        return this.delegate.getBody();
    }
    // On n'elimine aucune methode, on les considere toutes comme recursives.
    public boolean isRecursive () {
        return true;
    }

    /** Rendre l'arite de la fonction sous-jacente (self est compte comme
     * une variable). */
    public int getRealArity () {
        return this.delegate.getVariables().length;
    }

    public static IAST6methodDefinition parse (
            final Element e, final IParser6 parser)
    throws CEASTparseException {
        IAST4functionDefinition fun = CEASTfunctionDefinition.parse(e, parser);
        return parser.getFactory().newMethodDefinition(fun);
    }

    // Etablir un lien depuis la methode vers la classe la definissant.
    // Attention, ce lien n'est, pour l'heure, etabli qu'a la compilation.

    public void setDefiningClass (final IAST6classDefinition clazz) {
        this.definingClassDefinition = clazz;
    }
    private IAST6classDefinition definingClassDefinition;

    public int getMethodOffset (final IClassEnvironment common)
    throws CgenerationException {
        return this.definingClassDefinition.getMethodOffset(
                this.methodName, common );
    }

    public @OrNull IAST6methodDefinition findSuperMethod (
            final IClassEnvironment common )
    throws CgenerationException {
        IAST6classDefinition clazz = this.definingClassDefinition;
        String superClassName = clazz.getSuperClassName();
        while ( true ) {
            if ( "Object".equals(superClassName) ) {
                // Il n'y a que print et classOf comme methodes sur Object:
                if ( "print".equals(getMethodName()) ) {
                    return CEASTclassDefinition.printMethod;
                } else if ( "classOf".equals(getMethodName()) ) {
                    return CEASTclassDefinition.classOfMethod;
                } else {
                    // Pas de super-methode:
                    return null;
                }
            }
            clazz = common.findClassDefinition(superClassName);
            final IAST6methodDefinition[] methods =
                clazz.getProperMethodDefinitions();
            for ( IAST6methodDefinition m : methods ) {
                if ( getMethodName().equals(m.getMethodName()) ) {
                    return m;
                }
            }
            superClassName = clazz.getSuperClassName();
        }
    }

    @Override
    public IAST4functionDefinition normalize (
            final INormalizeLexicalEnvironment lexenv,
            final fr.upmc.ilp.ilp4.interfaces.INormalizeGlobalEnvironment common,
            final IAST4Factory factory)
      throws NormalizeException {
        return normalize6(
                lexenv,
                CEAST6.narrowToINormalizeGlobalEnvironment(common),
                CEAST6.narrowToIAST6Factory(factory) );
    }
```

```
public IAST4functionDefinition normalize6 (
        final INormalizeLexicalEnvironment lexenv,
        final INormalizeGlobalEnvironment common,
        final IAST6Factory factory )
  throws NormalizeException {
    final IAST4globalFunctionVariable gfv =
        CEAST.narrowToIAST4globalFunctionVariable(
                getDefinedVariable().normalize(lexenv, common, factory));
    INormalizeLexicalEnvironment bodyLexenv = lexenv;
    final IAST4variable[] variables = getVariables();
    final IAST4variable[] variables_ = new IAST4variable[variables.length];
    variables_[0] = this.selfVariable;
    bodyLexenv = bodyLexenv.extend(this.selfVariable);
    for ( int i = 1 ; i<variables.length ; i++ ) {
        variables_[i] = factory.newLocalVariable(variables[i].getName());
        bodyLexenv = bodyLexenv.extend(variables_[i]);
    }
    final IAST4expression body_ =
        getBody().normalize(bodyLexenv, common, factory);
    return factory.newMethodDefinition(
            getMethodName(),
            gfv,
            variables_,
            body_ );
}

@Override
public void compile6 (
        final StringBuffer buffer,
        final ICgenLexicalEnvironment lexenv,
        final ICgenEnvironment common )
throws CgenerationException {
    ICgenLexicalEnvironment methodLexenv =
        new CgenMethodLexicalEnvironment(this, lexenv);
    // Émettre en commentaire les fonctions appelées:
    if ( getInvokedFunctions().size() > 0 ) {
        buffer.append("/* Fonctions globales invoquées: ");
        for ( IAST4globalFunctionVariable gv : getInvokedFunctions() ) {
            buffer.append(gv.getMangledName());
            buffer.append(" ");
        }
        buffer.append(" */\n");
    }
    // Émettre la définition de la fonction:
    buffer.append("\nILP_Object\n");
    buffer.append(getDefinedVariable().getMangledName());
    this.delegate.compileVariableList(buffer);
    buffer.append("\n{\n");
    // Lien vers l'objet ILP_Method:
    buffer.append("static ILP_Method ilp_CurrentMethod = &ILP_object_");
    buffer.append(getFunctionName());
    buffer.append("_method;\n");
    // Calcul de la super methode:
    IAST6methodDefinition superMethod = findSuperMethod(common);
    buffer.append("static ILP_general_function ilp_SuperMethod = ");
    if ( superMethod == null ){
        buffer.append("NULL");
    } else {
        buffer.append(superMethod.getDefinedVariable().getMangledName());
    }
    buffer.append(";\n");
    // Sauvegarder des arguments pour super():
    buffer.append("ILP_Object ilp_CurrentArguments[");
    buffer.append(getVariables().length);
    // Pas de tableau de taille nulle en ISO C
    buffer.append("];\n");
    for ( int i=1 ; i<getVariables().length ; i++ ) {
        buffer.append(" ilp_CurrentArguments[");
        buffer.append(i);
        buffer.append("] = ");
        buffer.append(getVariables()[i].getMangledName());
        buffer.append(";\n");
    }
    final ICgenLexicalEnvironment bodyLexenv =
        CEAST6.narrowToICgenLexicalEnvironment(
                this.delegate.extendWithFunctionVariables(methodLexenv) );
    getBody().compile(buffer, bodyLexenv, common, ReturnDestination.create());
    buffer.append(";\n}");
```



```
}

@Override
public void compileHeader6 (
        final StringBuffer buffer,
        final ICgenLexicalEnvironment lexenv,
        final ICgenEnvironment common )
throws CgenerationException {
    // Pas genant d'engendrer une directive de trop:
    buffer.append("extern struct ILP_Method ILP_object_");
    buffer.append(getFunctionName());
    buffer.append("_method;\n");
}

@Override
public Object eval6 (ILexicalEnvironment lexenv, ICommon common)
        throws EvaluationException {
    final Object function =
        new ILPmethod(getFunctionName(),
                      getVariables(),
                      getBody() );
    common.updateGlobal(getFunctionName(), function);
    return function;
}

@Override
public void findGlobalVariables (
        final Set<IAST2variable> globalvars,
        final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv) {
    this.delegate.findGlobalVariables(globalvars, lexenv);
}

public <Data, Result, Exc extends Throwable> Result
accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
    return visitor.visit(this, data);
}
public <Data, Result, Exc extends Throwable> Result
accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
    return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
}
// NOTE: double methode surchargee.

@Override
public IAST4localVariable[] getLocalVariables() {
    throw new RuntimeException("not yet implemented!");
}

@Override
public String getMangledFunctionName() {
    return CStuff.mangle(this.methodName);
}
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTprogram.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.List;
import java.util.Set;
import java.util.Vector;

import org.w3c.dom.Element;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2;
import fr.upmc.ilp.ilp2.interfaces.IAST2functionDefinition;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTexpression;
import fr.upmc.ilp.ilp4.ast.CEASTfunctionDefinition;
import fr.upmc.ilp.ilp4.ast.CEASTglobalFunctionVariable;
import fr.upmc.ilp.ilp4.ast.FindingInvokedFunctionsException;
```

```
import fr.upmc.ilp.ilp4.ast.InliningException;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4;
import fr.upmc.ilp.ilp4.interfaces.IAST4Factory;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalFunctionVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4program;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;

/** La classe d'un programme composé de fonctions globales et
 * d'instructions. Ce n'est pas une expression ni une instruction mais
 * un programme. */

public class CEASTprogram
extends CEAST6
implements IAST6program {

  protected CEASTprogram (final IAST4functionDefinition[] definitions,
                          final IAST6classDefinition[] clazzes,
                          final IAST4expression body ) {
      this.delegate = new fr.upmc.ilp.ilp4.ast.CEASTprogram(definitions, body);
      this.clazzes = clazzes;
  }
  private final fr.upmc.ilp.ilp4.ast.CEASTprogram delegate;
  private final IAST6classDefinition[] clazzes;

  public fr.upmc.ilp.ilp4.ast.CEASTprogram getDelegate () {
      return this.delegate;
  }

  @ILPexpression(isArray=true)
  public IAST6classDefinition[] getClassDefinitions () {
      return this.clazzes;
  }
  @ILPexpression
  public IAST4expression getBody () {
    return this.getDelegate().getBody();
  }
  @ILPexpression(isArray=true)
  public IAST4functionDefinition[] getFunctionDefinitions () {
      IAST2functionDefinition<CEASTparseException>[] fds =
          this.getDelegate().getFunctionDefinitions();
      IAST4functionDefinition[] result =
          new IAST4functionDefinition[fds.length];
      System.arraycopy(fds, 0, result, 0, fds.length);
      return result;
  }

  /** Le constructeur analysant syntaxiquement un DOM. */

  public static IAST6program parse (final Element e, final IParser6 parser)
          throws CEASTparseException {
      List<IAST2<CEASTparseException>> itemsAsList =
          parser.parseList(e.getChildNodes());
      IAST4[] items = itemsAsList.toArray(new IAST4[0]);
      final List<IAST4functionDefinition> functionDefinitions = new Vector<>();
      final List<IAST6classDefinition> classDefinitions = new Vector<>();
      final List<IAST4expression> instructions = new Vector<>();
      for ( IAST4 item : items ) {
          if ( item instanceof IAST4functionDefinition ) {
              functionDefinitions.add((IAST4functionDefinition) item);
          } else if ( item instanceof IAST6classDefinition ) {
              classDefinitions.add((IAST6classDefinition) item);
          } else if ( item instanceof IAST4expression ) {
              instructions.add((IAST4expression) item);
          } else {
```



```
              final String msg = "Should never occur!";
              assert false : msg;
              throw new CEASTparseException(msg);
          }
      }
      IAST4functionDefinition[] defs =
          functionDefinitions.toArray(new IAST4functionDefinition[0]);
      IAST6classDefinition[] clazzes =
          classDefinitions.toArray(new IAST6classDefinition[0]);
      IAST4expression instrs = parser.getFactory().newSequence(
              instructions.toArray(CEASTexpression.EMPTY_EXPRESSION_ARRAY));
      return parser.getFactory().newProgram(defs, clazzes, instrs);
  }
  // NOTE: utiliser Xpath pour trier classes/fonctions/instructions.

  /** Interpréter un programme tout entier. On évalue les fonctions
   * globales ce qui enrichit au passage l'environnement global
   * (common) puis on évalue le corps du programme c'est-à-dire les
   * expressions qui ne sont pas dans des fonctions globales. */

  @Override
  public Object eval6 (final ILexicalEnvironment lexenv,
                       final ICommon common)
    throws EvaluationException {
      IAST6classDefinition[] clazzes = getClassDefinitions();
      for ( int i = 0 ; i<clazzes.length ; i++ ) {
          clazzes[i].eval(lexenv, common);
      }
      return getDelegate().eval(lexenv, common);
  }

  // Compilation

  @Override
  public String compile(
        final ICgenLexicalEnvironment lexenv,
        final fr.upmc.ilp.ilp2.interfaces.ICgenEnvironment common)
    throws CgenerationException {
      return compile6(
              lexenv,
              CEAST6.narrowToICgenEnvironment(common));
  }

  public String compile6 (final ICgenLexicalEnvironment lexenv,
                          final ICgenEnvironment common )
    throws CgenerationException {
    final StringBuffer buffer = new StringBuffer(4095);
    this.compile6(buffer, lexenv, common);
    return buffer.toString();
  }

  /** Compiler un programme tout entier. */

  @Override
  public void compile6 (final StringBuffer buffer,
                        final ICgenLexicalEnvironment lexenv,
                        final ICgenEnvironment common )
    throws CgenerationException {
      buffer.append("#include <stdio.h>\n");
      buffer.append("#include <stdlib.h>\n");
      buffer.append("#include <time.h>\n");
      buffer.append("\n");
      buffer.append("#include \"ilpObj.h\"\n");
      buffer.append("\n");
      // Déclarer les variables globales:
      buffer.append("/* Variables ou prototypes globaux: */\n");
      for ( IAST2variable var : getGlobalVariables() ) {
          IAST4globalVariable v = CEAST.narrowToIAST4globalVariable(var);
          v.compileGlobalDeclaration(buffer, lexenv, common);
      }
      IAST4functionDefinition[] definitions = getFunctionDefinitions();
      for ( IAST4functionDefinition fun : definitions ) {
          fun.compileHeader(buffer, lexenv, common);
      }
      // Déclarer les classes aussi:
      IAST6classDefinition[] clazzes = getClassDefinitions();
      for ( IAST6classDefinition clazz : clazzes ) {
          common.addClassDefinition(clazz);
      }
```

```
    for ( IAST6classDefinition clazz : clazzes ) {
        clazz.compileHeader(buffer, lexenv, common);
        // on en profite pour etablir le lien methode -> classe
        for ( IAST6methodDefinition method : clazz.getProperMethodDefinitions() ) {
            method.setDefiningClass(clazz);
        }
    }
    // Engendrer les classes et leurs methodes:
    buffer.append("\n/* Classes: */\n");
    for ( IAST6classDefinition clazz : clazzes ) {
        clazz.compile(buffer, lexenv, common);
    }
    // Engendrer le code des fonctions globales:
    buffer.append("\n/* Fonctions globales: */\n");
    for ( IAST4functionDefinition fun : definitions ) {
        fun.compile(buffer, lexenv, common);
    }
    // Émettre les instructions regroupées dans une fonction:
    IAST4globalFunctionVariable program =
        new CEASTglobalFunctionVariable(
                fr.upmc.ilp.ilp4.ast.CEASTprogram.PROGRAM);
    buffer.append("\n/* Code hors fonction: */");
    IAST4functionDefinition bodyAsFunction =
        new CEASTfunctionDefinition(
                program,
                new IAST4variable[0],
                getBody() );
    bodyAsFunction.compile(buffer, lexenv, common);
    buffer.append("\n");
    buffer.append("static ILP_Object ilp_caught_program () {\n");
    buffer.append("  struct ILP_catcher* current_catcher = ILP_current_catcher;\n");
    buffer.append("  struct ILP_catcher new_catcher;\n");
    buffer.append("\n");
    buffer.append("  if ( 0 == setjmp(new_catcher._jmp_buf) ) {\n");
    buffer.append("    ILP_establish_catcher(&new_catcher);\n");
    buffer.append("    return ilp_program();\n");
    buffer.append("  };\n");
    buffer.append("  /* Une exception est survenue. */\n");
    buffer.append("  return ILP_current_exception;\n");
    buffer.append("}\n");
    buffer.append("\n");
    buffer.append("int main (int argc, char *argv[]) {\n");
    buffer.append("  ILP_START_GC;\n");
    buffer.append("  ILP_print(ilp_caught_program());\n");
    buffer.append("  ILP_newline();\n");
    buffer.append("  return EXIT_SUCCESS;\n");
    buffer.append("}\n\n");
    buffer.append("\n/* fin */\n");
}

@Override
public void compileHeader6 (
        final StringBuffer buffer,
        final ICgenLexicalEnvironment lexenv,
        final ICgenEnvironment common )
throws CgenerationException {
    // rien!
}

/** Normaliser un programme dans un environnement lexical et global
 * particuliers. */

@Override
public IAST4program normalize (
        final INormalizeLexicalEnvironment lexenv,
        final fr.upmc.ilp.ilp4.interfaces.INormalizeGlobalEnvironment common,
        final IAST4Factory factory )
  throws NormalizeException {
    return normalize6(lexenv,
          CEAST6.narrowToINormalizeGlobalEnvironment(common),
          CEAST6.narrowToIAST6Factory(factory) );
}

public IAST6program normalize6 (
        final INormalizeLexicalEnvironment lexenv,
        final INormalizeGlobalEnvironment common,
        final IAST6Factory factory )
  throws NormalizeException {
    // Introduire d'abord toutes les variables globales nommant les
```



```
    // fonctions globales:
    IAST4functionDefinition[] definitions = getFunctionDefinitions();
    for ( int i = 0 ; i<definitions.length ; i++ ) {
        IAST4globalFunctionVariable gfv =
            factory.newGlobalFunctionVariable(
                    definitions[i].getDefinedVariable().getName());
        gfv.setFunctionDefinition(definitions[i]);
        common.add(gfv);
    }
    // ainsi que les definitions a normaliser des classes:
    // (les classes normalisent leurs methodes au passage et créent les
    // fonctions associées)
    IAST6classDefinition[] clazzes = getClassDefinitions();
    IAST6classDefinition[] clazzes_ =
        new IAST6classDefinition[clazzes.length];
    for ( int i = 0 ; i<clazzes.length ; i++ ) {
        clazzes_[i] = (IAST6classDefinition) clazzes[i]
            .normalize(lexenv, common, factory);
        common.addClassDefinition(clazzes_[i]);
    }
    // Normaliser les fonctions globales (et les méthodes):
    final IAST4functionDefinition[] definitions_ =
        new IAST4functionDefinition[definitions.length + 1];
    for ( int i = 0 ; i<definitions.length ; i++ ) {
        definitions_[i] = definitions[i].normalize(lexenv, common, factory);
    }
    // Empaqueter le code hors fonction en une fonction globale:
    final IAST4expression oldBody =
            getBody().normalize(lexenv, common, factory);
    final IAST4globalFunctionVariable program =
        CEASTglobalFunctionVariable.generateGlobalFunctionVariable(factory);
    common.add(program);
    final IAST4functionDefinition bodyAsFunction =
        factory.newFunctionDefinition(program, new IAST4variable[0], oldBody);
    program.setFunctionDefinition(bodyAsFunction);
    definitions_[definitions.length] = CEAST.narrowToIAST4functionDefinition(
        bodyAsFunction.normalize(lexenv, common, factory));
    final IAST4expression body_ =
        factory.newGlobalInvocation(program, new CEAST6expression[0]);
    return factory.newProgram(definitions_, clazzes_, body_);
}

@Override
public void computeInvokedFunctions ()
throws FindingInvokedFunctionsException {
    this.getDelegate().computeInvokedFunctions();
}

@Override
public void inline (IAST4Factory factory) throws InliningException {
    this.getDelegate().inline(factory);
}

/** Recensement des variables globales. */

// NOTE: on utilise ilp6.cgen.CgenEnvironment
@Override
public void computeGlobalVariables () {
    globals = GlobalCollector6.getGlobalVariables(this);
}
@Deprecated
public void computeGlobalVariables (final ICgenLexicalEnvironment lexenv) {
    // Cette methode est heritee mais son argument ne sert plus a rien car
    // on a change de mode de calcul des variables globales.
    computeGlobalVariables();
}
public IAST4globalVariable[] getGlobalVariables () {
    return this.globals;
}
private IAST4globalVariable[] globals = new IAST4globalVariable[0];
public void setGlobalVariables (IAST4globalVariable[] globals) {
    this.globals = globals;
}

/** Visiteur */

public <Data, Result, Exc extends Throwable> Result
accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
    return visitor.visit(this, data);
```

```
  }
  public <Data, Result, Exc extends Throwable> Result
  accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
  }
  // NOTE: double methode surchargee.

  @Override
  public void findGlobalVariables (final Set<IAST2variable> globalvars,
          final ICgenLexicalEnvironment lexenv ) {
      throw new RuntimeException("Should not occurr!");
  }
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTreadField.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.Set;

import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTlocalVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.cgen.AssignDestination;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6readField;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPinstance;

/** Lire un champ d'un objet. */

public class CEASTreadField
extends CEAST6expression
implements IAST6readField {

  public CEASTreadField (final String fieldName,
                         final IAST4expression object) {
    this.fieldName = fieldName;
    this.object = object;
  }
  private final String fieldName;
  private final IAST4expression object;

  @ILPexpression
  public IAST4expression getObject () {
      return this.object;
  }
  public String getFieldName () {
      return this.fieldName;
  }

  public static IAST6readField parse (final Element e, final IParser6 parser)
```



```
throws CEASTparseException {
    final String fieldName = e.getAttribute("champ");
    try {
      final XPathExpression targetPath = xPath.compile("./cible/*");
        final Node nTarget = (Node)
            targetPath.evaluate(e, XPathConstants.NODE);
        final IAST4expression target = CEAST.narrowToIAST4expression(
                parser.parse(nTarget) );
        return parser.getFactory().newReadField(fieldName, target);
  } catch (XPathExpressionException e1) {
      final String msg = "Should never occur!";
      assert false : msg;
      throw new CEASTparseException(msg);
  }
}
private static final XPath xPath = XPathFactory.newInstance().newXPath();

@Override
public Object eval6 (final ILexicalEnvironment lexenv,
                     final ICommon common)
  throws EvaluationException {
  final Object target = getObject().eval(lexenv, common);
  if ( target instanceof ILPinstance ) {
      ILPinstance o = (ILPinstance) target;
      return o.read(getFieldName());
  } else {
      throw new EvaluationException("Wrong class");
  }
}

@Override
public void compile6 (final StringBuffer buffer,
                      final ICgenLexicalEnvironment lexenv,
                      final ICgenEnvironment common,
                      final IDestination destination)
  throws CgenerationException {
  final IAST4variable tempvar = CEASTlocalVariable.generateVariable();
  buffer.append("{ ");
  tempvar.compileDeclaration(buffer, lexenv, common);
  getObject().compile(buffer, lexenv, common,
                      new AssignDestination(tempvar));
  buffer.append(";\n");
  buffer.append("if ( ILP_IsA(");
  buffer.append(tempvar.getMangledName());
  buffer.append(", (ILP_Class) &ILP_object_");
  final IAST6classDefinition cd =
    common.findDefiningClassDefinition(getFieldName());
  buffer.append(cd.getName());
  buffer.append("_class) ) {\n");
  destination.compile(buffer, lexenv, common);
  buffer.append(" ");
  buffer.append(tempvar.getMangledName());
  buffer.append("->_content.asInstance.field[");
  buffer.append(cd.getFieldOffset(getFieldName(), common));
  buffer.append("];\n} else {\n");
  destination.compile(buffer, lexenv, common);
  buffer.append(" ILP_UnknownFieldError(\"");
  buffer.append(getFieldName());
  buffer.append("\", ");
  buffer.append(tempvar.getMangledName());
  buffer.append(");\n}\n}\n");
}

@Override
public void findGlobalVariables (
        final Set<IAST2variable> globalvars,
        final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
  getObject().findGlobalVariables(globalvars, lexenv);
}

@Override
public IAST6readField normalize6 (
        final INormalizeLexicalEnvironment lexenv,
        final INormalizeGlobalEnvironment common,
        final IAST6Factory factory )
  throws NormalizeException {
  IAST4expression object_ = getObject().normalize(lexenv, common, factory);
  return factory.newReadField(getFieldName(), object_);
```

```
    }

    public <Data, Result, Exc extends Throwable> Result
    accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return visitor.visit(this, data);
    }
    public <Data, Result, Exc extends Throwable> Result
    accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
    }
    // NOTE: double methode surchargee.
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTself.java

```
package fr.upmc.ilp.ilp6.ast;

import org.w3c.dom.Element;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEASTvariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4localVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6self;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;

public class CEASTself
extends CEAST6expression
implements IAST6self {

    public CEASTself () {
        this(new CEASTselfVariable());
    }
    private final IAST4variable variable;

    public CEASTself (IAST4variable variable) {
        this.variable = variable;
    }

    public IAST4variable getLocalVariable () {
        return this.variable;
    }

    public static class CEASTselfVariable extends CEASTvariable
    implements IAST4localVariable {
        public CEASTselfVariable () {
            super("ilp_Self");
        }
    }

    public static IAST6self parse (final Element e, final IParser6 parser)
    throws CEASTparseException {
        return parser.getFactory().newSelf(new CEASTselfVariable());
    }

    @Override
    public Object eval6 (final ILexicalEnvironment lexenv,
                         final ICommon common)
    throws EvaluationException {
        return lexenv.lookup(getLocalVariable());
    }

    @Override
```



```
    public void compile6 (final StringBuffer buffer,
                          final ICgenLexicalEnvironment lexenv,
                          final ICgenEnvironment common,
                          final IDestination destination)
    throws CgenerationException {
        destination.compile(buffer, lexenv, common);
        buffer.append("ilp_Self;\n");
    }

    @Override
    public IAST6self normalize6 (
            final INormalizeLexicalEnvironment lexenv,
            final INormalizeGlobalEnvironment common,
            IAST6Factory factory )
    throws NormalizeException {
        return factory.newSelf(lexenv.isPresent(this.getLocalVariable()));
    }

    @Deprecated
    IAST4expression normalize (INormalizeLexicalEnvironment lexenv,
                               INormalizeGlobalEnvironment common )
    throws NormalizeException {
        return new CEASTself(lexenv.isPresent(this.getLocalVariable()));
    }

    public <Data, Result, Exc extends Throwable> Result
    accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return visitor.visit(this, data);
    }
    public <Data, Result, Exc extends Throwable> Result
    accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
    }
    // NOTE: double methode surchargee.
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTsend.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.Set;

import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTlocalVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.cgen.AssignDestination;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6send;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPinstance;

/** Envoi de message c'est-à-dire invocation d'une méthode. */
```

```
public class CEASTsend
extends CEAST6expression
implements IAST6send {

  public CEASTsend (final String methodName,
                    final IAST4expression receiver,
                    final IAST4expression[] arguments) {
    this.methodName = methodName;
    this.receiver = receiver;
    this.arguments = arguments;
  }
  private final String methodName;
  private final IAST4expression receiver;
  private final IAST4expression[] arguments;

  public String getMethodName () {
      return this.methodName;
  }
  @ILPexpression
  public IAST4expression getReceiver () {
      return this.receiver;
  }
  @ILPexpression(isArray=true)
  public IAST4expression[] getArguments () {
      return this.arguments;
  }

  public static IAST6send parse (final Element e, final IParser6 parser)
  throws CEASTparseException  {
      final String message = e.getAttribute("message");
      try {
          final XPathExpression receiverPath = xPath.compile("./receveur/*");
          final Node nReceiver = (Node)
              receiverPath.evaluate(e, XPathConstants.NODE);
          final IAST4expression receiver = CEAST.narrowToIAST4expression(
                  parser.parse(nReceiver) );

          final IAST4expression[] arguments =
              parser.findThenParseChildAsList(e, "arguments")
              .toArray(new IAST4expression[0]);

              return parser.getFactory().newSend(
                      message, receiver, arguments);
      } catch (XPathExpressionException e1) {
          final String msg = "Should never occur!";
          assert false : msg;
          throw new CEASTparseException(msg);
      }
  }
  private static final XPath xPath = XPathFactory.newInstance().newXPath();

  @Override
  public Object eval6 (final ILexicalEnvironment lexenv,
                       final ICommon common)
    throws EvaluationException {
    final Object target = getReceiver().eval(lexenv, common);
    IAST4expression[] arguments = getArguments();
    final Object[] value = new Object[arguments.length];
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      value[i] = arguments[i].eval(lexenv, common);
    }
    if ( target instanceof ILPinstance ) {
        ILPinstance o = (ILPinstance) target;
        return o.send(getMethodName(), value, common);
    } else {
        final String msg = "No such method " + getMethodName();
        throw new EvaluationException(msg);
    }
  }

  @Override
  public void compile6 (final StringBuffer buffer,
                        final ICgenLexicalEnvironment lexenv,
                        final ICgenEnvironment common,
                        final IDestination destination)
    throws CgenerationException {
    buffer.append("{ ");
```



```
    final IAST4variable tempInstance = CEASTlocalVariable.generateVariable();
    tempInstance.compileDeclaration(buffer, lexenv, common);
    final IAST4variable tempMethod = CEASTlocalVariable.generateVariable();
    buffer.append("  ILP_general_function ");
    buffer.append(tempMethod.getName());
    buffer.append(";\n");
    IAST4expression[] arguments = getArguments();
    final IAST4variable[] tempvar = new IAST4variable[arguments.length];
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      tempvar[i] = CEASTlocalVariable.generateVariable();
      tempvar[i].compileDeclaration(buffer, lexenv, common);
    }
    // getReceiver() et non this.receiver. Vu par Hiep Luu <htr999@gmail.com>
    getReceiver().compile(buffer, lexenv, common,
                          new AssignDestination(tempInstance));
    buffer.append(";\n");
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      arguments[i].compile(buffer, lexenv, common,
                           new AssignDestination(tempvar[i]));
      buffer.append(";\n");
    }
    // Déterminer la méthode séparément (mieux pour gdb):
    buffer.append(tempMethod.getName());
    buffer.append(" = ILP_find_method(");
    buffer.append(tempInstance.getMangledName());
    buffer.append(", &ILP_object_");
    buffer.append(getMethodName());
    buffer.append("_method, ");
    buffer.append(1 + arguments.length);
    buffer.append(");\n");
    // Invoquer la méthode:
    destination.compile(buffer, lexenv, common);
    buffer.append(tempMethod.getName());
    buffer.append("(");
    buffer.append(tempInstance.getMangledName());
    for ( int i = 0 ; i<arguments.length ; i++ ) {
      buffer.append(", ");
      buffer.append(tempvar[i].getMangledName());
    }
    buffer.append(");\n}\n");
  }

  @Override
  public void findGlobalVariables (
          final Set<IAST2variable> globalvars,
          final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
      IAST4expression[] arguments = getArguments();
      getReceiver().findGlobalVariables(globalvars, lexenv);
      for ( int i = 0 ; i<arguments.length ; i++ ) {
          arguments[i].findGlobalVariables(globalvars, lexenv);
      }
  }

  @Override
  public IAST6send normalize6 (
          final INormalizeLexicalEnvironment lexenv,
          final INormalizeGlobalEnvironment common,
          IAST6Factory factory )
    throws NormalizeException {
      IAST4expression receiver_ = CEAST.narrowToIAST4expression(
              getReceiver().normalize(lexenv, common, factory) );
      IAST4expression[] arguments = getArguments();
      IAST4expression[] arguments_ = new IAST4expression[arguments.length];
      for ( int i = 0 ; i<arguments.length ; i++ ) {
          arguments_[i] = CEAST.narrowToIAST4expression(
                  arguments[i].normalize(lexenv, common, factory) );
      }
      return factory.newSend(getMethodName(), receiver_, arguments_);
  }

  public <Data, Result, Exc extends Throwable> Result
  accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return visitor.visit(this, data);
  }
  public <Data, Result, Exc extends Throwable> Result
  accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
      return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
  }
  // NOTE: double methode surchargee.
```

```
}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTsuper.java

```
package fr.upmc.ilp.ilp6.ast;

import org.w3c.dom.Element;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.cgen.CgenMethodLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6super;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPmethod;

public class CEASTsuper
extends CEAST6expression
implements IAST6super {

    public CEASTsuper() {}

    public static IAST6super parse (final Element e, final IParser6 parser)
    throws CEASTparseException {
        return parser.getFactory().newSuper();
    }

    @Override
    public Object eval6 (final ILexicalEnvironment lexenv,
                         final ICommon common)
    throws EvaluationException {
        ILPmethod currentMethod = (ILPmethod) lexenv.lookup(ILPmethod.cmv);
        return currentMethod.callSuper(lexenv, common);
    }

    @Override
    public void compile6 (final StringBuffer buffer,
                          final ICgenLexicalEnvironment lexenv,
                          final ICgenEnvironment common,
                          final IDestination destination)
    throws CgenerationException {
        IAST6methodDefinition currentMethod = null;
        ICgenLexicalEnvironment le = lexenv;
        while ( ! le.isEmpty() ) {
            if ( le instanceof CgenMethodLexicalEnvironment ) {
                CgenMethodLexicalEnvironment cmle =
                    (CgenMethodLexicalEnvironment) le;
                currentMethod = cmle.getMethodDefinition();
                break;
            } else {
                le = le.getNext();
            }
        }
        if ( currentMethod != null ) {
            destination.compile(buffer, lexenv, common);
            buffer.append("ILP_FindAndCallSuperMethod(");
            buffer.append(currentMethod.getRealArity());
            buffer.append(");\n");
        } else {
            final String msg = "No supermethod!";
            throw new CgenerationException(msg);
        }
    }
```



```
    @Override
    public IAST6super normalize6 (
            final INormalizeLexicalEnvironment lexenv,
            final INormalizeGlobalEnvironment common,
            final IAST6Factory factory )
    throws NormalizeException {
        return factory.newSuper();
    }
    // NOTE: Raisonnable car pas de classe imbriquees.

    public <Data, Result, Exc extends Throwable> Result
    accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return visitor.visit(this, data);
    }
    public <Data, Result, Exc extends Throwable> Result
    accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
    }
    // NOTE: double methode surchargee.

}
```

Java/src/fr/upmc/ilp/ilp6/ast/CEASTwriteField.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.Set;

import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import fr.upmc.ilp.annotation.ILPexpression;
import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
import fr.upmc.ilp.ilp2.interfaces.IDestination;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp4.ast.CEAST;
import fr.upmc.ilp.ilp4.ast.CEASTlocalVariable;
import fr.upmc.ilp.ilp4.ast.NormalizeException;
import fr.upmc.ilp.ilp4.cgen.AssignDestination;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp4.interfaces.IAST4visitor;
import fr.upmc.ilp.ilp4.interfaces.INormalizeLexicalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IAST6Factory;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.IAST6writeField;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;
import fr.upmc.ilp.ilp6.interfaces.IParser6;
import fr.upmc.ilp.ilp6.runtime.ICommon;
import fr.upmc.ilp.ilp6.runtime.ILPinstance;

/** Modifier un champ dans un objet. */

public class CEASTwriteField
extends CEAST6expression
implements IAST6writeField {

  public CEASTwriteField (final String fieldName,
                          final IAST4expression object,
                          final IAST4expression value) {
    this.fieldName = fieldName;
    this.object = object;
    this.value = value;
  }
  private final String fieldName;
```

```
    private final IAST4expression object;
    private final IAST4expression value;

    @ILPexpression
    public IAST4expression getObject () {
        return this.object;
    }
    @ILPexpression
    public IAST4expression getValue () {
        return this.value;
    }
    public String getFieldName () {
        return this.fieldName;
    }

    public static IAST6writeField parse (
            final Element e, final IParser6 parser)
    throws CEASTparseException {
        final String fieldName = e.getAttribute("champ");
        try {
            final XPathExpression targetPath = xPath.compile("./cible/*");
            final Node nTarget = (Node)
                targetPath.evaluate(e, XPathConstants.NODE);
            final IAST4expression target = CEAST.narrowToIAST4expression(
                    parser.parse(nTarget) );

            final XPathExpression valuePath = xPath.compile("./valeur/*");
            final Node nValue = (Node)
                valuePath.evaluate(e, XPathConstants.NODE);
            final IAST4expression value = CEAST.narrowToIAST4expression(
                    parser.parse(nValue) );

            return parser.getFactory().newWriteField(fieldName, target, value);
        } catch (XPathExpressionException e1) {
            final String msg = "Should never occur!";
            assert false : msg;
            throw new CEASTparseException(msg);
        }
    }
    private static final XPath xPath = XPathFactory.newInstance().newXPath();
    // NOTE: partager ce XPath.

    @Override
    public Object eval6 (final ILexicalEnvironment lexenv,
                         final ICommon common)
      throws EvaluationException {
      final Object target = getObject().eval(lexenv, common);
      final Object newValue = getValue().eval(lexenv, common);
      if ( target instanceof ILPinstance ) {
          ILPinstance o = (ILPinstance) target;
          return o.write(getFieldName(), newValue);
      } else {
          throw new EvaluationException("Wrong class");
      }
    }

    @Override
    public void compile6 (final StringBuffer buffer,
                          final ICgenLexicalEnvironment lexenv,
                          final ICgenEnvironment common,
                          final IDestination destination)
      throws CgenerationException {
      final IAST4variable tempvar = CEASTlocalVariable.generateVariable();
      final IAST4variable tempValue = CEASTlocalVariable.generateVariable();
      buffer.append("{ ");
      tempvar.compileDeclaration(buffer, lexenv, common);
      tempValue.compileDeclaration(buffer, lexenv, common);
      getObject().compile(buffer, lexenv, common, new AssignDestination(tempvar));
      buffer.append(";\n");
      getValue().compile(buffer, lexenv, common, new AssignDestination(tempValue));
      buffer.append(";\n");
      buffer.append("if ( ILP_IsA(");
      buffer.append(tempvar.getMangledName());
      buffer.append(", (ILP_Class) &ILP_object_");
      final IAST6classDefinition cd =
        common.findDefiningClassDefinition(getFieldName());
      buffer.append(cd.getName());
```



```
      buffer.append("_class) ) {\n");
      destination.compile(buffer, lexenv, common);
      buffer.append(" (");
      buffer.append(tempvar.getMangledName());
      buffer.append("->_content.asInstance.field[");
      buffer.append(cd.getFieldOffset(getFieldName(), common));
      buffer.append("] = ");
      buffer.append(tempValue.getMangledName());
      buffer.append(");\n} else {\n");
      destination.compile(buffer, lexenv, common);
      buffer.append(" ILP_UnknownFieldError(\"");
      buffer.append(getFieldName());
      buffer.append("\", ");
      buffer.append(tempvar.getMangledName());
      buffer.append(");\n}\n}\n");
    }

    @Override
    public void findGlobalVariables (
            final Set<IAST2variable> globalvars,
            final fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment lexenv ) {
        getObject().findGlobalVariables(globalvars, lexenv);
        getValue().findGlobalVariables(globalvars, lexenv);
    }

    @Override
    public IAST6writeField normalize6 (
            final INormalizeLexicalEnvironment lexenv,
            final INormalizeGlobalEnvironment common,
            IAST6Factory factory )
      throws NormalizeException {
      IAST4expression object_ = getObject().normalize(lexenv, common, factory);
      IAST4expression value_ =  getValue().normalize(lexenv, common, factory);
      return factory.newWriteField(getFieldName(), object_, value_);
    }

    public <Data, Result, Exc extends Throwable> Result
    accept (IAST6visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return visitor.visit(this, data);
    }
    public <Data, Result, Exc extends Throwable> Result
    accept (IAST4visitor<Data, Result, Exc> visitor, Data data) throws Exc {
        return CEAST6.narrowToIAST6visitor(visitor).visit(this, data);
    }
    // NOTE: double methode surchargee.
}
```

Java/src/fr/upmc/ilp/ilp6/ast/GlobalCollector6.java

```
package fr.upmc.ilp.ilp6.ast;

import fr.upmc.ilp.ilp4.ast.GlobalCollector;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4program;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6instantiation;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6readField;
import fr.upmc.ilp.ilp6.interfaces.IAST6self;
import fr.upmc.ilp.ilp6.interfaces.IAST6send;
import fr.upmc.ilp.ilp6.interfaces.IAST6super;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.IAST6writeField;

public class GlobalCollector6 extends GlobalCollector
implements IAST6visitor<Object, Object, RuntimeException> {

    public GlobalCollector6 () {
        super();
    }

    public static IAST4globalVariable[] getGlobalVariables (IAST4program ast) {
        final GlobalCollector6 visitor = new GlobalCollector6();
        ast.accept(visitor, null);
        return visitor.globals.toArray(new IAST4globalVariable[0]);
    }
```

```
    @Override
    public Object visit(IAST6classDefinition iast, Object nothing) {
        for (IAST6methodDefinition md : iast.getProperMethodDefinitions()) {
            md.accept(this, nothing);
        }
        return null;
    }

    @Override
    public Object visit(IAST6methodDefinition iast, Object nothing) {
        iast.getBody().accept(this, nothing);
        return null;
    }

    @Override
    public Object visit(IAST6instantiation iast, Object nothing) {
        for (IAST4expression e : iast.getArguments()) {
            e.accept(this, nothing);
        }
        return null;
    }

    @Override
    public Object visit(IAST6send iast, Object nothing) {
        iast.getReceiver().accept(this, nothing);
        for (IAST4expression e : iast.getArguments()) {
            e.accept(this, nothing);
        }
        return null;
    }

    @Override
    public Object visit(IAST6readField iast, Object nothing) {
        iast.getObject().accept(this, nothing);
        return null;
    }

    @Override
    public Object visit(IAST6writeField iast, Object nothing) {
        iast.getObject().accept(this, nothing);
        iast.getValue().accept(this, nothing);
        return null;
    }

    @Override
    public Object visit(IAST6self iast, Object nothing) {
        return null;
    }

    @Override
    public Object visit(IAST6super iast, Object nothing) {
        return null;
    }


}
```

Java/src/fr/upmc/ilp/ilp6/ast/NormalizeGlobalEnvironment.java

```
package fr.upmc.ilp.ilp6.ast;

import java.util.HashMap;
import java.util.Map;

import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.INormalizeGlobalEnvironment;

/** Une implantation d'environnement global pour la normalisation des
 * expressions. */

public class NormalizeGlobalEnvironment
extends fr.upmc.ilp.ilp4.ast.NormalizeGlobalEnvironment
implements INormalizeGlobalEnvironment {

  public NormalizeGlobalEnvironment () {
      super();
      this.classMap = new HashMap<>();
  }
  private final Map<String,IAST6classDefinition> classMap;

  // On enregistre les classes afin de verifier l'arite des instantiations.

  public IAST6classDefinition findClassDefinition (final String className) {
    IAST6classDefinition result = this.classMap.get(className);
    if ( result != null ) {
      return result;
    } else {
      final String msg = "No such class " + className;
      throw new RuntimeException(msg);
    }
  }

  public void addClassDefinition (final IAST6classDefinition clazz) {
    this.classMap.put(clazz.getName(), clazz);
  }

}
```

Java/src/fr/upmc/ilp/ilp6/ast/XMLwriter.java

```
package fr.upmc.ilp.ilp6.ast;

import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Element;

import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4functionDefinition;
import fr.upmc.ilp.ilp4.interfaces.IAST4globalFunctionVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6instantiation;
import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
import fr.upmc.ilp.ilp6.interfaces.IAST6program;
import fr.upmc.ilp.ilp6.interfaces.IAST6readField;
import fr.upmc.ilp.ilp6.interfaces.IAST6self;
import fr.upmc.ilp.ilp6.interfaces.IAST6send;
import fr.upmc.ilp.ilp6.interfaces.IAST6super;
import fr.upmc.ilp.ilp6.interfaces.IAST6visitor;
import fr.upmc.ilp.ilp6.interfaces.IAST6writeField;

public class XMLwriter
extends fr.upmc.ilp.ilp4.ast.XMLwriter
implements IAST6visitor<Object, Element, RuntimeException> {

    public XMLwriter ()
    throws ParserConfigurationException {
        super();
    }

    public Element visit(IAST6classDefinition iast, Object data) {
       final Element result = this.document.createElement(
                    iast.getClass().getName() );
       if ( this.memory.containsKey(iast) ) {
            final Element old = this.memory.get(iast);
            result.setAttribute("idref", old.getAttribute("id"));
       } else {
            result.setAttribute("id", "" + getCounter());
            this.memory.put(iast, result);
            // Serialisation:
            result.setAttribute("name", iast.getName());
            result.setAttribute("super", iast.getSuperClassName());
            StringBuffer sb = new StringBuffer();
            for (String fieldName : iast.getProperFieldNames()) {
                sb.append(fieldName).append(" ");
            }
            result.setAttribute("properFieldNames", sb.toString());
            final Element methods = this.document.createElement("properMethods");
            result.appendChild(methods);
            Element lastVisitedElement = null;
            for (IAST6methodDefinition md : iast.getProperMethodDefinitions()) {
                lastVisitedElement = md.accept(this, data);
                methods.appendChild(lastVisitedElement);
```

```
        }
    }
    return result;
}
public Element visit(IAST6instantiation iast, Object data) {
       final Element result = this.document.createElement(
               iast.getClass().getName() );
    if ( this.memory.containsKey(iast) ) {
         final Element old = this.memory.get(iast);
         result.setAttribute("idref", old.getAttribute("id"));
    } else {
         result.setAttribute("id", "" + getCounter());
         this.memory.put(iast, result);
         // Serialisation:
         result.setAttribute("className", iast.getClassName());
         Element lastVisitedElement;
         for ( IAST4expression e : iast.getArguments()) {
             lastVisitedElement = e.accept(this, null);
             result.appendChild(lastVisitedElement);
         }
    }
    return result;
}
public Element visit(IAST6methodDefinition iast, Object data) {
       final Element result = this.document.createElement(
               iast.getClass().getName() );
    if ( this.memory.containsKey(iast) ) {
         final Element old = this.memory.get(iast);
         result.setAttribute("idref", old.getAttribute("id"));
    } else {
         result.setAttribute("id", "" + getCounter());
         this.memory.put(iast, result);
         // Serialisation:
         result.setAttribute("methodName", iast.getMethodName());
         result.setAttribute("realArity", ""+iast.getRealArity());
         final Element definition = this.document.createElement("function");
         result.appendChild(definition);

         definition.setAttribute("name", iast.getFunctionName());
         definition.setAttribute("recursive",
                 Boolean.toString(iast.isRecursive()) );

         Element lastVisitedElement = iast.getDefinedVariable().accept(this, null);
         definition.appendChild(lastVisitedElement);

         final Element funs =
             this.document.createElement("invokedFunctions");
         definition.appendChild(funs);
         for ( IAST4globalFunctionVariable gfv : iast.getInvokedFunctions() ) {
             lastVisitedElement = gfv.accept(this, null);
             funs.appendChild(lastVisitedElement);
         }

         final Element vars = this.document.createElement("variables");
         definition.appendChild(vars);
         for ( IAST4variable lv : iast.getVariables() ) {
             lastVisitedElement = lv.accept(this, null);
             vars.appendChild(lastVisitedElement);
         }

         final Element body = this.document.createElement("body");
         definition.appendChild(body);
         lastVisitedElement = iast.getBody().accept(this, null);
         body.appendChild(lastVisitedElement);
    }
    return result;
}
public Element visit(IAST6program iast, Object data) {
       final Element result = this.document.createElement(
               iast.getClass().getName() );
    if ( this.memory.containsKey(iast) ) {
         final Element old = this.memory.get(iast);
         result.setAttribute("idref", old.getAttribute("id"));
    } else {
         result.setAttribute("id", "" + getCounter());
         this.memory.put(iast, result);
         // Serialisation:
         Element lastVisitedElement;
```



```
         final Element classes = this.document.createElement("classes");
         result.appendChild(classes);
         for ( IAST6classDefinition cd : iast.getClassDefinitions() ) {
             lastVisitedElement = cd.accept(this, null);
             classes.appendChild(lastVisitedElement);
         }
         final Element functions = this.document.createElement("functions");
         result.appendChild(functions);
         for ( IAST4functionDefinition fd : iast.getFunctionDefinitions() ) {
             lastVisitedElement = fd.accept(this, null);
             functions.appendChild(lastVisitedElement);
         }
    }
    return result;
}

public Element visit(IAST6readField iast, Object data) {
    final Element result = this.document.createElement(
                iast.getClass().getName() );
    if ( this.memory.containsKey(iast) ) {
         final Element old = this.memory.get(iast);
         result.setAttribute("idref", old.getAttribute("id"));
    } else {
         result.setAttribute("id", "" + getCounter());
         this.memory.put(iast, result);
         // Serialisation:
         result.setAttribute("fieldname", iast.getFieldName());
         Element lastVisitedElement = iast.getObject().accept(this, null);
         result.appendChild(lastVisitedElement);
    }
    return result;
}

public Element visit(IAST6self iast, Object data) {
     final Element result = this.document.createElement(
             iast.getClass().getName() );
     if ( this.memory.containsKey(iast) ) {
         final Element old = this.memory.get(iast);
         result.setAttribute("idref", old.getAttribute("id"));
     } else {
         result.setAttribute("id", "" + getCounter());
         this.memory.put(iast, result);
     }
     return result;
}

public Element visit(IAST6send iast, Object data) {
       final Element result = this.document.createElement(
               iast.getClass().getName() );
       if ( this.memory.containsKey(iast) ) {
            final Element old = this.memory.get(iast);
            result.setAttribute("idref", old.getAttribute("id"));
       } else {
            result.setAttribute("id", "" + getCounter());
            this.memory.put(iast, result);
            // Serialisation:
            result.setAttribute("methodName", iast.getMethodName());
            Element lastVisitedElement = iast.getReceiver().accept(this, null);
            result.appendChild(lastVisitedElement);
            for ( IAST4expression arg : iast.getArguments() ) {
                lastVisitedElement = arg.accept(this, null);
                result.appendChild(lastVisitedElement);
            }
       }
       return result;
}

public Element visit(IAST6super iast, Object data) {
    final Element result = this.document.createElement(
            iast.getClass().getName() );
    if ( this.memory.containsKey(iast) ) {
        final Element old = this.memory.get(iast);
        result.setAttribute("idref", old.getAttribute("id"));
    } else {
        result.setAttribute("id", "" + getCounter());
        this.memory.put(iast, result);
    }
    return result;
```

```
    }

    public Element visit(IAST6writeField iast, Object data) {
       final Element result = this.document.createElement(
                iast.getClass().getName() );
       if ( this.memory.containsKey(iast) ) {
            final Element old = this.memory.get(iast);
            result.setAttribute("idref", old.getAttribute("id"));
       } else {
            result.setAttribute("id", "" + getCounter());
            this.memory.put(iast, result);
            // Serialisation:
            result.setAttribute("fieldname", iast.getFieldName());
            Element lastVisitedElement = iast.getObject().accept(this, null);
            result.appendChild(lastVisitedElement);
            lastVisitedElement = iast.getValue().accept(this, null);
            result.appendChild(lastVisitedElement);
       }
       return result;
    }
}
```

Java/src/fr/upmc/ilp/ilp6/runtime/CommonPlus.java

```
package fr.upmc.ilp.ilp6.runtime;

import java.util.HashMap;
import java.util.Map;

import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp4.ast.CEASTglobalVariable;
import fr.upmc.ilp.ilp4.ast.CEASTprimitiveInvocation;
import fr.upmc.ilp.ilp4.interfaces.IAST4expression;
import fr.upmc.ilp.ilp4.interfaces.IAST4variable;
import fr.upmc.ilp.ilp6.ast.CEASTself;

/** Cette classe implante les caractéristiques
 * générales d'un interprète du langage ILP6.
 */

public class CommonPlus
extends fr.upmc.ilp.ilp4.runtime.CommonPlus
implements ICommon {

    public CommonPlus () {
        super();
        this.classMap = new HashMap<>();
        this.fillClassMap();
    }

  /** Rechercher une classe par son nom. */

  public ILPClass findClass (final String name)
  throws EvaluationException {
      ILPClass result = classMap.get(name);
      if ( result != null ) {
          return result;
      } else {
          throw new EvaluationException("No such class " + name);
      }
  }
  private final Map<String,ILPClass> classMap;

  /** Ajouter une classe. */

  public void addClass (final ILPClass clazz) {
      classMap.put(clazz.getName(), clazz);
  }

  /** Créer la classe Object et ses deux méthodes prédéfinies. la
   * première méthode est o.print() implantée par un appel à la
   * primitive print(value). La seconde méthode est o.classOf(), elle
   * n'est pas implantée car il faut créer toutes les autres classes
   * dans la bibliothèque d'exécution.
   */

  private void fillClassMap () {
    ILPmethod[] object_methods = new ILPmethod[2];
```



```
    CEASTself self = new CEASTself();
    object_methods[0] =
      new ILPmethod(
            "print",
            new IAST4variable[] { self.getLocalVariable() },
            new CEASTprimitiveInvocation(
                  new CEASTglobalVariable("print"),
                  new IAST4expression[] { self }));
    // On réalloue une variable self (pour les besoins de la normalisation):
    self = new CEASTself();
    object_methods[1] =
      new ILPmethod(
            "classOf",
            new IAST4variable[] { self.getLocalVariable() },
            new CEASTprimitiveInvocation(
                  new CEASTglobalVariable("classOf"),
                  new IAST4expression[] { self }));
    classMap.put("Object",
                 new ILPClass("Object",
                              null,           // pas de superclasse
                              new String[0],  // pas de champ
                              new String[] { "print", "classOf" },
                              object_methods));
  }

}
```

Java/src/fr/upmc/ilp/ilp6/runtime/ICommon.java

```
package fr.upmc.ilp.ilp6.runtime;

import fr.upmc.ilp.ilp1.runtime.EvaluationException;

public interface ICommon
extends fr.upmc.ilp.ilp2.interfaces.ICommon {

    ILPClass findClass (final String name)
    throws EvaluationException;

    void addClass (final ILPClass clazz);
}
```

Java/src/fr/upmc/ilp/ilp6/runtime/ILPClass.java

```
package fr.upmc.ilp.ilp6.runtime;

import java.util.HashMap;
import java.util.Map;

import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.interfaces.ICommon;

/** Une classe pour ILP6. */

public class ILPClass {

  public ILPClass (final String className,
                   final ILPClass superClass,
                   final String[] fieldName,
                   final String[] methodName,
                   final ILPmethod[] method) {
    this.className = className;
    this.superClass = superClass;
    if ( superClass != null ) {
      this.fieldName = new String[superClass.fieldName.length + fieldName.length];
      for ( int i = 0 ; i<superClass.fieldName.length ; i++ ) {
        this.fieldName[i] = superClass.fieldName[i];
      }
      for ( int i = 0 ; i<fieldName.length ; i++ ) {
        this.fieldName[superClass.fieldName.length + i] = fieldName[i];
      }
    } else {
      // Il s'agit de créer la class Object!
      this.fieldName = fieldName;
    }
```

```
    assert method.length == methodName.length;
    this.method = new HashMap<>();
    for ( int i = 0 ; i<method.length ; i++ ) {
      this.method.put(methodName[i], method[i]);
      method[i].setDefiningClass(this);
    }
  }
  private final String className;
  private final ILPClass superClass;
  // Tous les champs y compris ceux hérités:
  private final String[] fieldName;
  // Seules les méthodes propres:
  private final Map<String,ILPmethod> method;

  /** Renvoyer le nom de la classe. */

  public String getName () {
    return this.className;
  }
  public ILPClass getSuperClass () {
      return this.superClass;
  }

  /** Renvoyer le nom du rank-ieme champ. */

  public String fieldName (int rank) {
    return this.fieldName[rank];
  }

  /** Indiquer le nombre total de champs (propres ainsi qu'hérité)
   * d'une instance de cette classe. */

  public int fieldSize () {
    return this.fieldName.length;
  }

  /** Calculer le rang d'un champ. */

  public int fieldRank (final String name)
    throws EvaluationException {
    for ( int i = 0 ; i<fieldName.length ; i++ ) {
      if ( fieldName[i].equals(name) ) {
        return i;
      }
    }
    throw new EvaluationException("No such field " + name);
  }

  /** Retourner la methode propre ayant un certain nom. */

  public ILPmethod getMethodByName (final String methodName)
  throws EvaluationException {
      ILPmethod method = this.method.get(methodName);
      if ( method != null ) {
          return method;
      } else {
          final String msg = "No such method " + methodName;
          throw new EvaluationException(msg);
      }
  }

  /** Envoyer un message à une instance. */

  public Object send (final ILPinstance self,
                      final String message,
                      final Object[] argument,
                      final ICommon common)
    throws EvaluationException {
    ILPmethod m = this.method.get(message);
    if ( m != null ) {
      final Object[] newArgument = new Object[1 + argument.length];
      newArgument[0] = self;
      for ( int i = 0 ; i<argument.length ; i++ ) {
        newArgument[i+1] = argument[i];
      }
      return m.invoke(newArgument, common);
    } else {
```



```
      // Pas de méthode propre de ce nom!
      if ( superClass != null ) {
        return superClass.send(self, message, argument, common);
      } else {
        // On est sur Object
        throw new EvaluationException("No such method " + message);
      }
    }
  }

  /** Envoyer un message à un bloc d'arguments. */

  public Object send (final String message,
                      final Object[] argument,
                      final ICommon common)
  throws EvaluationException {
      ILPmethod m = this.method.get(message);
      if ( m != null ) {
          return m.invoke(argument, common);
      } else {
          // Pas de méthode propre de ce nom!
          if ( superClass != null ) {
              return superClass.send(message, argument, common);
          } else {
              // On est sur Object
              throw new EvaluationException("No such method " + message);
          }
      }
  }

  /** Fabriquer un vecteur de champs (représentés par des chaînes) en
   * ajoutant de nouveaux champs à ceux de la classe. */

  protected String[] extend (final String[] properFieldName) {
    final int count = fieldName.length + properFieldName.length;
    final String[] result = new String[count];
    for ( int i = 0 ; i<fieldName.length ; i++ ) {
      result[i] = fieldName[i];
    }
    for ( int i = 0 ; i<properFieldName.length ; i++ ) {
      result[fieldName.length + i] = properFieldName[i];
    }
    return result;
  }
}
```

Java/src/fr/upmc/ilp/ilp6/runtime/ILPinstance.java

```
package fr.upmc.ilp.ilp6.runtime;

import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.interfaces.ICommon;

/** Une instance pour ILP6. */

public class ILPinstance {

  public ILPinstance (final ILPClass clazz, final Object[] argument) {
    this.clazz = clazz;
    this.field = argument;
  }
  private final ILPClass clazz;
  private final Object[] field;

  /** Rendre la classe pour ILP6 de cette instance. */

  public ILPClass classOf () {
    return clazz;
  }

  /** Lire un champ de cette instance. */

  public Object read (final String fieldName)
    throws EvaluationException {
    return field[clazz.fieldRank(fieldName)];
  }
```

```
  /** Écrire un champ de cette instance. */

  public Object write (final String fieldName, final Object value)
    throws EvaluationException {
    field[clazz.fieldRank(fieldName)] = value;
    return Boolean.FALSE;
  }

  /** Invoquer une méthode sur cette instance. */

  public Object send (final String message,
                      final Object[] argument,
                      final ICommon common)
    throws EvaluationException {
    return clazz.send(this, message, argument, common);
  }

  /** Imprimer une instance (et tous ses champs). */
  @Override
  public String toString () {
    final StringBuffer sb = new StringBuffer();
    sb.append("<");
    sb.append(clazz.getName());
    for ( int i = 0 ; i<field.length ; i++ ) {
      sb.append(":");
      sb.append(clazz.fieldName(i));
      sb.append("=");
      sb.append(field[i]);
    }
    sb.append(">");
    return sb.toString();
  }

}
```

Java/src/fr/upmc/ilp/ilp6/runtime/ILPmethod.java

```
package fr.upmc.ilp.ilp6.runtime;

import fr.upmc.ilp.ilp1.runtime.EvaluationException;
import fr.upmc.ilp.ilp2.ast.CEASTparseException;
import fr.upmc.ilp.ilp2.interfaces.IAST2instruction;
import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
import fr.upmc.ilp.ilp2.interfaces.ICommon;
import fr.upmc.ilp.ilp2.interfaces.ILexicalEnvironment;
import fr.upmc.ilp.ilp2.runtime.LexicalEnvironment;
import fr.upmc.ilp.ilp2.runtime.UserFunction;
import fr.upmc.ilp.ilp4.ast.CEASTlocalVariable;
import fr.upmc.ilp.ilp4.interfaces.IAST4localVariable;

public class ILPmethod
// on n'herite pas de UserGlobalFunction
extends UserFunction {

    public ILPmethod (final String name,
                      final IAST2variable[] variable,
                      final IAST2instruction<CEASTparseException> body) {
        super(variable,
              body,
              LexicalEnvironment.EmptyLexicalEnvironment.create() );
        this.name = name;
    }
    protected final String name;

    public String getMethodName () {
        return this.name;
    }

    // On retrouve la methode courante par cette pseudo-variable.
    public static IAST4localVariable cmv =
        new CEASTlocalVariable("ilp_CurrentMethod");
    protected static IAST4localVariable cmargs =
        new CEASTlocalVariable("ilp_CurrentArguments");

    public void setDefiningClass (ILPClass clazz) {
        this.definingClass = clazz;
```



```
    }
    private ILPClass definingClass;

    public ILPClass getDefiningClass () {
        return this.definingClass;
    }

    public Object callSuper (final ILexicalEnvironment lexenv,
                             final ICommon common )
    throws EvaluationException {
        Object[] arguments = (Object[]) lexenv.lookup(cmargs);
        return getDefiningClass().getSuperClass()
            .send(getMethodName(), arguments, common);
    }

    @Override
    public Object invoke (final Object[] arguments,
                          final ICommon common)
      throws EvaluationException {
        IAST2variable[] variables = getVariables();
        if ( variables.length != arguments.length ) {
            final String msg = "Wrong arity";
            throw new EvaluationException(msg);
        }
        ILexicalEnvironment lexenv = getEnvironment()
            .extend(cmv, this)
            .extend(cmargs, arguments);
        for ( int i = 0 ; i<variables.length ; i++ ) {
            lexenv = lexenv.extend(variables[i], arguments[i]);
        }
        return getBody().eval(lexenv, common);
    }

}
```

Java/src/fr/upmc/ilp/ilp6/cgen/CgenEnvironment6.java

```
package fr.upmc.ilp.ilp6.cgen;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import fr.upmc.ilp.ilp1.cgen.CgenerationException;
import fr.upmc.ilp.ilp6.interfaces.IAST6classDefinition;
import fr.upmc.ilp.ilp6.interfaces.ICgenEnvironment;

/** La représentation de l'environnement des opérateurs prédéfinis. Il
 * définit comment les compiler. C'est un peu l'analogue de
 * runtime/Common pour le paquetage cgen. */

public class CgenEnvironment6
extends fr.upmc.ilp.ilp2.cgen.CgenEnvironment
implements ICgenEnvironment {

  public CgenEnvironment6 () {
      super();
      alreadyGeneratedClassMacroSet = new HashSet<>();
      alreadyGeneratedMethodObjectSet = new HashSet<>();
      alreadyGeneratedMethodObjectSet.add("print");
      alreadyGeneratedMethodObjectSet.add("classOf");
  }

  /** Enregistrer une définition de classe ainsi que ses champs propres. */

  public void addClassDefinition (final IAST6classDefinition cd) {
    classMap.put(cd.getName(), cd);
    final String[] fieldName = cd.getProperFieldNames();
    for ( int i = 0 ; i<fieldName.length ; i++ ) {
      fieldMap.put(fieldName[i], cd);
    }
  }
  protected Map<String,IAST6classDefinition> classMap = new HashMap<>();
  protected Map<String,IAST6classDefinition> fieldMap = new HashMap<>();

  /** Rechercher la définition d'une classe. */
```

```
41
    public IAST6classDefinition findClassDefinition (final String className) {
      IAST6classDefinition result = classMap.get(className);
      if ( result != null ) {
        return result;
46    } else {
        final String msg = "No such class " + className;
        throw new RuntimeException(msg);
      }
    }
51
    /** Rechercher l'offset associé à une méthode.
     *
     * Attention, l'offset peut être le même pour des méthodes n'ayant
     * rien à voir mais étant défini dans des sous-classes s?urs. */
56
    public int getMethodOffset (final String className,
                                final String methodName)
      throws CgenerationException {
      final IAST6classDefinition cd = classMap.get(className);
61    if ( cd == null ) {
        final String msg = "No such class " + className;
        throw new CgenerationException(msg);
      } else {
        return cd.getMethodOffset(methodName, this);
66    }
    }

    /** Rechercher la classe ayant introduit un champ. */
71
    public IAST6classDefinition findDefiningClassDefinition (String fieldName)
      throws CgenerationException {
      final IAST6classDefinition cd = fieldMap.get(fieldName);
      if ( cd != null ) {
        return cd;
76    } else {
        final String msg = "No defining class for " + fieldName;
        throw new CgenerationException(msg);
      }
    }
81
    /** Engendrer la définition d'un type de classe avec un certain
     * nombre de méthodes si elle n'a pas déjà été engendrée! */

    public boolean alreadyGeneratedClassMacro (final int methodCount) {
86        return !alreadyGeneratedClassMacroSet.add(methodCount);
    }
    private final Set<Integer> alreadyGeneratedClassMacroSet;

    /** Engendrer l'objet représentant une méthode si elle n'a pas déjà
91   * été engendrée! */

    public boolean alreadyGeneratedMethodObject (final String methodName) {
      return !alreadyGeneratedMethodObjectSet.add(methodName);
    }
96  private final Set<String> alreadyGeneratedMethodObjectSet;

}
```

Java/src/fr/upmc/ilp/ilp6/cgen/CgenLexicalEnvironment6.java

```
1   package fr.upmc.ilp.ilp6.cgen;

    import fr.upmc.ilp.ilp1.cgen.CgenerationException;
    import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
    import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
6
    /** Les environnements lexicaux pour la compilation. Ils sont semblables
     * a ceux d'ilp2 sauf qu'ils permettent en plus de savoir dans quelle
     * methode l'on est lorsque l'on compile une methode. C'est utile pour
     * compiler l'appel a la super-methode. */
11
    public class CgenLexicalEnvironment6
    extends fr.upmc.ilp.ilp2.cgen.CgenLexicalEnvironment {

      public CgenLexicalEnvironment6 (final IAST2variable variable,
16                                    final ICgenLexicalEnvironment next) {
```



```
        super(variable, next);
      }
      public CgenLexicalEnvironment6 (final IAST2variable variable,
                                      final String cname,
21                                    final ICgenLexicalEnvironment next) {
        super(variable, cname, next);
      }

      // On utilise maintenant l'egalité physique.
26
      @Override
      public String compile (final IAST2variable variable)
        throws CgenerationException {
        if ( this.variable == variable ) {
31        return this.variable.getMangledName();
        } else {
          return getNext().compile(variable);
        }
      }
36
      @Override
      public boolean isPresent (IAST2variable variable) {
        if ( this.variable == variable ) {
          return true;
41      } else {
          return next.isPresent(variable);
        }
      }

46    @Override
      public ICgenLexicalEnvironment extend (final IAST2variable variable) {
        return new CgenLexicalEnvironment6(variable, this);
      }
      @Override
51    public ICgenLexicalEnvironment extend (final IAST2variable variable,
                                             final String cname) {
        return new CgenLexicalEnvironment6(variable, cname, this);
      }

56    /** ============================================
       */
      public static class CgenEmptyLexicalEnvironment6
      extends fr.upmc.ilp.ilp2.cgen.CgenLexicalEnvironment.CgenEmptyLexicalEnvironment {

61      public CgenEmptyLexicalEnvironment6 () {}
        private static final CgenEmptyLexicalEnvironment6
            THE_EMPTY_LEXICAL_ENVIRONMENT;
        static {
            THE_EMPTY_LEXICAL_ENVIRONMENT = new CgenEmptyLexicalEnvironment6();
66      }

        public static ICgenLexicalEnvironment create () {
            return CgenEmptyLexicalEnvironment6.THE_EMPTY_LEXICAL_ENVIRONMENT;
        }
71
        @Override
        public ICgenLexicalEnvironment extend (final IAST2variable variable) {
            return new CgenLexicalEnvironment6(variable, this);
        }
76      @Override
        public ICgenLexicalEnvironment extend (final IAST2variable variable,
                                               final String cname) {
            return new CgenLexicalEnvironment6(variable, cname, this);
        }
81    }

}
```

Java/src/fr/upmc/ilp/ilp6/cgen/CgenMethodLexicalEnvironment.java

```
1   package fr.upmc.ilp.ilp6.cgen;

    import fr.upmc.ilp.annotation.OrNull;
    import fr.upmc.ilp.ilp1.cgen.CgenerationException;
    import fr.upmc.ilp.ilp2.interfaces.IAST2variable;
6   import fr.upmc.ilp.ilp2.interfaces.ICgenLexicalEnvironment;
    import fr.upmc.ilp.ilp6.interfaces.IAST6methodDefinition;
```

```
public class CgenMethodLexicalEnvironment
implements ICgenLexicalEnvironment {

    public CgenMethodLexicalEnvironment (final IAST6methodDefinition m,
                                         final ICgenLexicalEnvironment next) {
        this.methodDefinition = m;
        this.next = next;
    }
    protected final IAST6methodDefinition methodDefinition;
    protected final ICgenLexicalEnvironment next;

    public IAST6methodDefinition getMethodDefinition () {
        return this.methodDefinition;
    }
    public ICgenLexicalEnvironment getNext () {
        return this.next;
    }
    // Il n'y a pas de variable associee:
    public IAST2variable getVariable () {
        throw new RuntimeException("Should never be invoked!");
    }
    public boolean isEmpty () {
        return false;
    }

    public String compile (IAST2variable variable)
    throws CgenerationException {
        return this.next.compile(variable);
    }

    public ICgenLexicalEnvironment extend (final IAST2variable variable) {
        return new CgenLexicalEnvironment6(variable, this);
    }
    public ICgenLexicalEnvironment extend (final IAST2variable variable,
                                           final String cname) {
        return new CgenLexicalEnvironment6(variable, cname, this);
    }

    public boolean isPresent (IAST2variable variable) {
        return this.next.isPresent(variable);
    }

    @Override
    public @OrNull ICgenLexicalEnvironment shrink(IAST2variable variable) {
        return this.next.shrink(variable);
    }

}
```