Bull - IMAG
Rapport Bull-IMAG n.4
Décembre 1990

Exemples de programmes en langage Guide

Michel Riveill et Cécile Roisin

Unité mixte Bull - IMAG
Z.I. de Mayencin - 2, avenue de Vignate
F-38610 Gières - FRANCE
e-mail : (riveill, roisin)@imag.imag.fr

Résumé :
Ce document est un complément au manuel du langage Guide. Il contient des exemples de programmes Guide de différentes complexités : des exemples simples pour illustrer les traits du langage, des exemples issus de la bibliothèque Guide pour pouvoir utiliser les types et les classes fournis en standard (notamment la définition des types List et Catal), et enfin quelques programmes d'application simples. Cependant, les applications qui servent pour les démonstrations du système Guide ne sont pas données ici car elles sont de taille trop importante pour ce document.

Remerciements :
Les sources de ces exemples sont issus du travail de nombreuses personnes du projet Guide : en plus des deux auteurs de ce rapport, on peut citer : François Exertier, Hervé Jamrozik, Serge Lacourte, Patrick Le Dot, Xavier Rousset de Pina.
Tous les programmes ci-dessous ont été compilés et exécutés avec la version 1.5 du système Guide disponible au 15 décembre 1990.

1 Exemples simples

On trouvera dans cette section une suite de petits exemples simples permettant de comprendre les principaux traits du langage. Ils proviennent en partie de ceux du manuel.

1.1 Définition d'un type

```
TYPE Boite IS
  SYNONYM Name = String [80];          // synonymes
  proprietaire : Name;                 // variable d'état visible
  METHOD mettre (IN REF Element);      // méthodes
    // ajout d'un element dans la boite
  METHOD prendre (OUT REF Element);
    // retrait d'un élément de la boîte
END Boite.

TYPE Cercle IS
  SYNONYM BitMap = Array [1000,1000] OF Boolean;
```

```
  SYNONYM MailBoxList = List OF REF MailBox;
  size : Integer = 10;               // variable visible initialisée
  toto : REF Cercle = "my_circle";
  CONST titi : REF Cercle = "my_circle"; // var. visible constante
  rayon : Integer;                   // variable visible
  METHOD Surface : Integer;
END Cercle.
```

1.2 Définition d'une classe

```
CLASS PetiteBoite
IMPLEMENTS BoiteAObjets IS     // BoiteAObjets est le type implementé
                               // par la classe PetiteBoite
  // boite ne pouvant pas contenir plus de 5 éléments;
  ATTRIBUTE UNMOVABLE;       // Caractéristique des objets de la classe

  CONST taille : Integer = 5;   // variables d'état (non visibles)
  nb_elt, dernier : Integer;
  contenu : Array [taille] OF REF Element;
  premier : Integer = 0;
  person : REF Personne = NIL;  // référence initialisée a` NIL
  moi: REF Personne = "cecile"; // référence initialisée

  METHOD concat (IN boite: REF BoiteAObjets) :REF BoiteAObjets;
    BEGIN
    //...
  END concat ;
  METHOD mettre (IN elt : REF Element);
    BEGIN  IF nb_elt < taille THEN
    dernier := (dernier+1) MOD taille ;
    nb_elt := nb_elt + 1 ;
    ELSE
      // ...
      ;
    END;
  END mettre ;
  METHOD prendre (OUT elt : REF Element);
    BEGIN
    //...
  END prendre;
  METHOD Init;
    BEGIN
    output.WriteInteger(moi.nom);
    output.WriteString("\n");
  END Init;
END PetiteBoite.
```

1.3 Sous-typage et héritage

1.3.1 Exemple de relation de sous-typage portant sur les variables visibles

```
TYPE Personne IS
  nom : String[15];
END Personne.

TYPE Etudiant SUBTYPE OF Personne IS
  universite : String[20];  // héritage de la variable visible nom
                       // et ajout de la variable visible universite
END Etudiant.
```

1.3.2 Exemple de relation de sous-typage portant sur les méthodes

On définit ici un type Livret_A qui est un sous-type du type Compte : il possède les mêmes propriétés (variables visibles et méthodes d'accès) que le type Compte, avec en plus des propriétés spécifiques (la méthode calcul_interets).

```
TYPE Compte IS
  solde : Integer ;                        // solde du compte bancaire;
  METHOD crediter (IN val : Integer);     // augmentation du solde;
  METHOD debiter (IN val : Integer);
END Compte.

TYPE Livret_A SUBTYPE OF Compte IS
  // héritage des méthodes et de la variable visible du type Compte;
  // ajout d'une nouvelle méthode : calcul_interets;
  METHOD calcul_interets (IN taux : Integer); // calcul des intérêts
                                          // et mise à jour du solde;
END Livret_A.
```

1.3.3 Exemple de relation d'héritage

Pour qu'une classe soit définie comme une sous-classe d'une autre classe, il faut que le type qu'elle implémente soit sous-type du type implémenté par la super-classe. Ici, le type Reflivre est sous-type du type Refbib.

La classe Reflivre définie ci-dessous est une sous-classe de la classe Refbib : le modèle d'état est surchargé et la méthode Imprimer est redéfinie. De plus on a ajouté une nouvelle méthode (elle apparaît dans le type).

```
TYPE Reflivre SUBTYPE OF Refbib IS
  METHOD Special_livre;
END Reflivre.

CLASS Refbib IMPLEMENTS Refbib IS
  numref : Integer;
  nom : String[15];
  METHOD imprimer ;
    BEGIN
```

```
    output.WriteInteger (numref);
    output.WriteString (nom);
  END imprimer ;
END Refbib.

CLASS Reflivre SUBCLASS OF Refbib
IMPLEMENTS Reflivre IS
  editeur : String[10]; // surcharge de l'état;
  METHOD imprimer ;     // surcharge de la méthode imprimer;
    BEGIN
    SUPER.imprimer;   // impression des champs définis dans la classe
                  // Refbib en appelant la méthode imprimer de la
                  // super-classe (Refbib);
    output.WriteString (editeur);
  END imprimer ;
  METHOD Special_livre;
    BEGIN
  END Special_livre;
END Reflivre.
```

1.4 Utilisation de la conformité

1.4.1 Conformité induite par une relation de sous-typage

Pour cet exemple, on utilise les types Comptes et Livret_A définis en section 1.3.2.

```
CLASS Compte IMPLEMENTS Compte IS
  METHOD crediter (IN val : Integer);
    BEGIN
  END crediter;
  METHOD debiter (IN val : Integer);
    BEGIN
  END debiter;
END Compte.

CLASS Livret_A SUBCLASS OF Compte
IMPLEMENTS  Livret_A IS
  METHOD Calcul_interets (IN taux : Integer);
    BEGIN
  END Calcul_interets;
END Livret_A.

TYPE Testconf1 IS
  METHOD Imprimer_solde (IN cpt : REF Compte);
  METHOD Init ;
END Testconf1.

CLASS Testconf1 IMPLEMENTS Testconf1 IS
  METHOD Imprimer_solde (IN cpt : REF Compte);
```

```
    // la méthode Imprimer_solde accepte en entrée tout
    // paramètre de type Compte ou conforme à Compte
    BEGIN
    output.WriteInteger(cpt.solde);
    output.WriteString("\n");
  END Imprimer_solde;

// exemple d'appel
  METHOD Init ;
    mon_compte : REF Compte;
    mon_livret_A : REF Livret_A;
    BEGIN
    mon_compte := Compte.New;
    mon_livret_A := Livret_A.New;
    SELF.Imprimer_solde(mon_compte);
    SELF.Imprimer_solde(mon_livret_A);
    // appel correct car la référence mon_livret_A est d'un type
      // conforme au type du paramètre formel cpt (de type Compte) de
      // de la méthode M
  END Init;
END Testconf1.
```

1.4.2 Conformité correspondant à une relation de restriction

Dans cet exemple, la relation de conformité est utilisée entre deux types quelconques (n'ayant une relation explicite de sous-typage entre eux). En fait, la relation entre les types correspond à une relation de "restriction" que le langage aurait pu imposer de rendre explicite (ce n'est pas le cas actuellement) : dans l'exemple ci-dessous, on a défini d'une part, un type Bibliotheque qui spécifie toutes les opérations que l'on peut effectuer sur les objets bibliothèques ; on a défini d'autre part le type Usager_bibliotheque qui ne contient qu'un sous-ensemble des opérations définies dans le type Bibliotheque. C'est une façon d'interdire aux usagers l'accès à certaines opérations de gestion de la bibliothèque.

Ainsi deux références (bib dans la classe Administrateur, et u_bib dans la classe Usager) de types différents (Bibliotheque et Usager_bibliotheque) vont désigner un même objet (de nom symbolique "bibliotheque_Guide") de type Bibliotheque. Ceci est rendu possible par l'affectation conforme d'une référence de type Bibliotheque (pointant sur l'objet "bibliotheque_Guide") à une référence de type Usager_bibliotheque.

```
TYPE Document IS
END Document.

CLASS Document IMPLEMENTS Document IS
END Document.

TYPE Livre SUBTYPE OF Document IS
END Livre.

CLASS Livre SUBCLASS OF Document IMPLEMENTS Livre IS
END Livre.
```

```
TYPE Emprunteur IS
END Emprunteur.

CLASS Emprunteur IMPLEMENTS Emprunteur IS
END Emprunteur.

TYPE Bibliotheque IS
  METHOD Emprunter(IN doc : REF Document; IN qui : REF Emprunteur;
                   OUT status : Integer);
  METHOD Rendre (IN doc : REF Document; IN qui : REF Emprunteur;
                 OUT status : Integer);
  METHOD Ajouter_doc (IN titre, auteur, edition : String;
                      OUT doc : REF Document);
END Bibliotheque.

CLASS Bibliotheque IMPLEMENTS Bibliotheque IS
  // état et code des méthodes
  METHOD Emprunter(IN doc : REF Document; IN qui : REF Emprunteur;
                   OUT status : Integer);
    BEGIN
    output.WriteString("emprunter\n");
  END Emprunter;

  METHOD Rendre (IN doc : REF Document; IN qui : REF Emprunteur;
                 OUT status : Integer);
    BEGIN  output.WriteString("rendre\n");
  END Rendre;

  METHOD Ajouter_doc (IN titre, auteur, edition : String;
                      OUT doc : REF Document);
    BEGIN
    output.WriteString("Ajouter_doc\n");
    doc := Document.New;
  END Ajouter_doc;
END Bibliotheque.

TYPE Usager_bibliotheque IS
  METHOD Emprunter (IN livre : REF Livre; IN qui : REF Emprunteur;
                    OUT status : Integer);
  METHOD Rendre (IN livre : REF Livre; IN qui : REF Emprunteur;
                 OUT status : Integer);
END Usager_bibliotheque.
```

Le type Usager_bibliotheque est une restriction du type Bibliotheque: on ne permet aux usagers que les opérations d'emprunt et de remise. De plus, les seuls documents qu'ils peuvent emprunter sont des livres. D'après la définition de la relation de conformité, le type Bibliotheque est conforme au type Usager_bibliotheque. Ce qui permet d'accéder à la même bibliothèque avec des interfaces différentes.

```
TYPE Administrateur IS
  METHOD Init;
END Administrateur.

CLASS Administrateur IMPLEMENTS Administrateur IS
  bib : REF Bibliotheque;
  moi : REF Emprunteur;
       // l'administrateur accède à la bibliotheque avec une référence
       // de type Bibliotheque (toutes les opérations sont permises)

  METHOD Init;
    commande : String[15];
    d : REF Document;
    ok : Integer;
    BEGIN
    moi := Emprunteur.New;
    bib := ASSERTYPE(catal.Search("bibliotheque_Guide"));
    IF bib = NIL THEN
      bib := Bibliotheque.New;
      catal.Insert("bibliotheque_Guide", bib);
    END;
    REPEAT
      commande := input.ReadString;
      CASE commande OF
        "nouveau_doc" : bib.Ajouter_doc ("toto","titi","gf",d); END;
        "consulter_doc" : bib.Emprunter(d,moi,ok); END;
         //...;
      END;
    UNTIL commande = "fini";
  END Init;
END Administrateur.

TYPE Usager IS
  METHOD Init;
END Usager.

CLASS Usager IMPLEMENTS Usager IS
  moi : REF Emprunteur;
  u_bib : REF Usager_bibliotheque;
          // un usager accède à la bibliothèque avec une référence
          // de type Usager_bibliotheque (accès restreint)
  METHOD Init ;  commande : String[16];
    bib : REF Bibliotheque;
    roman : REF Livre;
    ok : Integer;
    BEGIN
    moi := Emprunteur.New;
    bib := ASSERTYPE(catal.Search("bibliotheque_Guide"));
    IF bib = NIL THEN RETURN; END;
    u_bib := bib;   // affectation valide car le type de
```

```
                    // bib est conforme à u_bib
    REPEAT
      commande := input.ReadString;
      CASE commande OF
        "consulter_livre" : roman := Livre.New;
                            u_bib.Emprunter(roman,moi,ok);
                            END;
        //...;
      END;
    UNTIL commande = "fini";
  END Init;
END Usager.
```

1.5 Parallélisme et synchronisation

1.5.1 Gestion d'un tampon

Cet exemple illustre l'expression des conditions d'accès aux objets à l'aide de la clause de contrôle associée à chaque méthode. Par exemple, dans le cas ci-dessous, on exprime que l'ajout d'un élément dans le tampon n'est possible que s'il n'est pas plein et que s'il n'y a pas d'autres ajouts en cours.

Pour tester le fonctionnement de ce tampon, on a défini les classes Putter, Getter et Testbuffer permettant l'appel des opérations Put et Get du tampon en parallèle : ce parallélisme est exprimé par l'instruction CO_BEGIN CO_END.

On montre aussi dans cet exemple comment les conditions de synchronisation sont héritées ; la règle est que si une méthode est héritée, la condition de synchronisation qui lui est associée est automatiquement héritée sauf si une nouvelle clause est définie dans la sous-classe pour cette méthode (la nouvelle clause remplace celle de la super-classe). De plus, l'appel explicite d'une méthode de la super-classe (à l'aide du mot-clé SUPER) est considéré comme un appel de procédure (pas d'évaluation de la condition de synchronisation).

```
// test de la synchro avec héritage et CO_BEGIN
// ----------------------------------------

TYPE ProducerConsumer IS
  METHOD Get(OUT m:Char);
  METHOD Put(IN m:Char);
END ProducerConsumer.

CLASS FixedSizeBuffer
 IMPLEMENTS ProducerConsumer IS
  CONST size : Integer = 4;
  buffer : Array [size] OF Char;
  dernier, premier: Integer = 0;
  METHOD Put(IN m: Char);
    i : Integer;
    BEGIN  buffer[dernier]:=m;
    dernier:=(dernier+1) MOD size;
```

```
    FOR i := 0 TO size-1 DO
      output.WriteChar(buffer[i]);
      output.WriteString("  ");
    END;
    output.WriteString(" Put\n");
  END Put;

  METHOD Get(OUT m: Char);
    i : Integer;
    BEGIN
    m:=buffer[premier];
    buffer[premier] := ' ';
    premier:= (premier +1) MOD size;
    FOR i := 0 TO size-1 DO
      output.WriteChar(buffer[i]);
      output.WriteString("  ");
    END;
    output.WriteString("Get\n");
  END Get;

  CONTROL
  Put: (completed(Put) - completed (Get)<size) AND current(Put)=0;
  Get: (completed (Put) > completed (Get)) AND current(Get)=0;
END FixedSizeBuffer.

TYPE Testbuffer IS
  METHOD Init;
END Testbuffer.

CLASS Testbuffer IMPLEMENTS Testbuffer IS
 METHOD Init;
    b : REF ProducerConsumer;
     p : REF Putter;g : REF Getter;
    BEGIN
     b := FixedSizeBuffer.New;
                    // en changeant la classe de creation de b,
                    // on peut tester trois cas d'heritage :
                    // classes Exclusifbuffer(1, 2 ou 3)
    p := Putter.New;
    g := Getter.New;
    CO_BEGIN
      a1 : p.activer(5,b);
      a2 : g.activer(6,b);
    CO_END ALL;
    output.WriteString ("fin testbuffer");
  END Init;
END Testbuffer.

TYPE Putter IS
  METHOD activer(IN n : Integer; IN b : REF  ProducerConsumer);
```

```
END Putter.

TYPE Getter IS
  METHOD activer(IN n : Integer; IN b : REF  ProducerConsumer);
END Getter.

CLASS Putter IMPLEMENTS Putter IS
  METHOD activer(IN n : Integer; IN b : REF  ProducerConsumer);
    i : Integer;
    BEGIN
    FOR i := 1 TO n DO
      b.Put('a');
      SYSTEM("FCT_sleep(FCT_random() % 13);");
    END;
  END activer;
END Putter.

CLASS Getter IMPLEMENTS Getter IS
  METHOD activer(IN n : Integer; IN b : REF  ProducerConsumer);
    i : Integer;
    x : Char;
    BEGIN
    FOR i := 1 TO n DO
      b.Get(x);
      SYSTEM("FCT_sleep(FCT_random() % 17);");
    END;
  END activer;
END Getter.

//exemples avec héritage
//----------------------

// surcharge des méthodes et héritage des conditions de
// synchronisation

CLASS Exclusifbuffer1 SUBCLASS OF FixedSizeBuffer
IMPLEMENTS ProducerConsumer IS
  METHOD Put(IN m: Char);
    BEGIN
    SUPER.Put(m);          // revient à un appel de procédure
                           // car l'appel se fait sur le même objet,
                           // donc pas d'evaluation de la synchro
                           // de la classe mère
  END Put;
  METHOD Get(OUT m: Char);
    BEGIN
    SUPER.Get(m);
  END Get;
                         // les conditions de synchro sont héritées
END  Exclusifbuffer1.
```

```
// surcharge des méthodes et redéfinition des conditions
// de synchronisation

CLASS Exclusifbuffer SUBCLASS OF FixedSizeBuffer
IMPLEMENTS ProducerConsumer IS
  METHOD Put(IN m: Char);
    BEGIN
    SUPER.Put(m);           // revient à un appel de procédure
                            // donc pas d'évaluation de la synchro
                            // de la classe mère
  END Put;
  METHOD Get(OUT m: Char);
    BEGIN
    SUPER.Get(m);
  END Get;
  CONTROL                      // les conditions de synchro sont
                               // redéfinies complètement.
    Put : (completed(Put) - completed (Get)<size) AND
            (current(Put) =0) AND (current(Get) =0) ;
    Get : (completed (Put) > completed (Get)) AND
            (current(Put) =0) AND (current(Get) =0);
END Exclusifbuffer2.

// heritage des méthodes et redéfinition de la synchronisation

CLASS Exclusifbuffer3 SUBCLASS OF FixedSizeBuffer
IMPLEMENTS ProducerConsumer IS
  CONTROL
    Put : (completed(Put) - completed (Get)<size) AND
           (current(Put) =0) AND (current(Get) =0);
    Get : (completed (Put) > completed (Get)) AND
           (current(Put) =0) AND (current(Get) =0);
END Exclusifbuffer3.
```

1.5.2 Control

Le but de ce programme est de tester différentes stratégies de synchronisation pour un modèle de lecteurs/rédacteurs.

```
TYPE Control IS
  METHOD Read ( IN j : Integer );
  METHOD Write ( IN j : Integer );
END Control.

TYPE NoPri SUBTYPE OF Control IS
  METHOD GetOrder (IN lect : Boolean) : Integer;
  METHOD AskRead (IN request_order : Integer);
  METHOD AskWrite (IN request_order : Integer);
END NoPri.

// lecteurs et rédacteurs sans synchronisation
// ------------------------------------------
```

```
CLASS NoSync
IMPLEMENTS Control IS

  PROCEDURE Message ( IN j : Integer; txt : String);
  str : String[80];
  BEGIN
    CASE j OF
      1: str := " 1"+txt;
        END;
      2: str := "                  2"+txt;
        END;
      3: str := "                                    3"+txt;
        END;
    END;
    output.WriteString( str );
  END Message;

  METHOD Read ( IN j : Integer );
  BEGIN
    Message ( j, " debut Read\n" );
    SYSTEM("FCT_guideSleep(FCT_random() % 4);");
    Message ( j, " fin Read\n" );
  END Read;

  METHOD Write ( IN j : Integer );
  BEGIN
    Message ( j, " debut/fin Write\n" );
  END Write;

END NoSync.

// lecteurs et rédacteurs avec synchronisation en exclusion mutuelle
// ----------------------------------------------------------------

CLASS Exclusive SUBCLASS OF NoSync
IMPLEMENTS Control IS

  // heritage de l'etat et des methodes de la classe NoSync

  CONTROL                      // definition de la clause de controle
   EXCLUSIVE(Read, Write);

END Exclusive.

// synchronisation suivant le modele lecteur/redacteur
// --------------------------------------------------

CLASS ReadWrite SUBCLASS OF NoSync
IMPLEMENTS Control IS

  // heritage de l'etat et des methodes de la classe NoSync

  CONTROL                      // definition de la clause de controle
    WRITER(Write), READER(Read);

END ReadWrite.

// synchronisation avec le modele lecteur/redacteur et priorite
// -----------------------------------------------------------
// aux lecteurs
// ------------

CLASS PriRead SUBCLASS OF NoSync
IMPLEMENTS Control IS
```

```
  CONTROL
    Write : current(Write) + current(Read) + pending(Read) = 0;
    Read : current(Write) = 0;

END PriRead.

// synchronisation avec le modele lecteur/redacteur et priorite
// ------------------------------------------------------------
// aux redacteurs
// --------------

CLASS PriWrite SUBCLASS OF NoSync
IMPLEMENTS Control IS

  CONTROL
    Write : current(Write) + current(Read) = 0;
    Read : current(Write) + pending(Write) = 0;

END PriWrite.

// synchronisation avec le modele lecteur/redacteur et priorite
// ------------------------------------------------------------
// a la premiere demande d'acces (lecteur ou redacteur)
// ----------------------------------------------------

CLASS NoPri SUBCLASS OF NoSync
IMPLEMENTS NoPri IS

  nbRead : Integer = 0;
  nbWrite : Integer = 0;

  METHOD GetOrder (IN read : Boolean) : Integer;
  BEGIN
    IF read THEN
      nbRead := nbRead+1;
      RETURN nbWrite;
    ELSE
      nbWrite := nbWrite+1;
      RETURN nbWrite+nbRead-1;
    END;
  END GetOrder;

  METHOD Read ( IN j : Integer );
  BEGIN
    SELF.AskRead(SELF.GetOrder(TRUE));
    SUPER.Read(j);
  END Read;

  METHOD Write ( IN j : Integer );
  BEGIN
    SELF.AskWrite(SELF.GetOrder(FALSE));
    SUPER.Write(j);
  END Write;

  METHOD AskRead (IN request_order : Integer);
  BEGIN
  END AskRead;

  METHOD AskWrite (IN request_order : Integer);
  BEGIN
  END AskWrite;

  CONTROL
```

```
    EXCLUSIVE(GetOrder);
    AskRead : completed(Write) = request_order;
    AskWrite : completed(Write) + completed(Read) = request_order;

END NoPri.

// test des differents cas definis ci-dessus
// ----------------------------------------

TYPE TestControl IS
  METHOD Init;
  METHOD Scenario ( IN i : Integer ) ;
END TestControl.

CLASS TestControl
IMPLEMENTS TestControl IS

  r : REF Control;

  METHOD Scenario  ( IN i : Integer );
  BEGIN
    r.Read(i); r.Read(i);
    r.Write(i); r.Write(i);
    r.Read(i); r.Read(i); r.Read(i); r.Read(i); r.Read(i);
    r.Write(i); r.Write(i); r.Write(i);
    r.Read(i);
  END Scenario;

  METHOD Init;
  i : Integer;
  BEGIN
    FOR i := 1 TO 6 DO
      CASE i OF
        1: output.WriteString( "ecrivains et lecteurs exclusifs\n");
           r := Exclusive.New;
           END;
        2: output.WriteString( "lecteurs redacteurs\n");
           r := ReadWrite.New;
           END;
        3: output.WriteString( "priorite aux lecteurs\n");
           r := PriRead.New;
           END;
        4: output.WriteString( "priorite aux ecrivains\n");
           r := PriWrite.New;
           END;
        5: output.WriteString( "ecrivains et lecteurs exclusifs\n");
           r := Exclusive.New;
           END;
        6: output.WriteString( "ecrivains et lecteurs avec la meme");
           output.WriteString( "priorite\n");
           r := NoPri.New;
           END;
      END;
      CO_BEGIN
        a : SELF.Scenario(1);
        b : SELF.Scenario(2);
        c : SELF.Scenario(3);
      CO_END a AND b AND c;
    END;
  END Init;

END TestControl.
```

1.6 Persistance, destruction explicite d'objets, destruction d'un domaine

```
// test de la destruction des objets et du domaine courant
// ------------------------------------------------------

TYPE Element IS
  nom : String[10];
END Element.

CLASS Element IMPLEMENTS Element IS
END Element.

TYPE Test1 IS
  METHOD Init;
END Test1.

CLASS Test1 IMPLEMENTS Test1 IS
  a,b,c,d : REF Element;
  METHOD Init;
    BEGIN
    output.WriteString ("creation des objets \n");
    a := Element.New;
    b := Element.New;
    c := Element.New;
    catal.Insert("objet_a", a);
    d := c;
    output.WriteString ("destruction d'un objet \n");
    d.Destroy;
    c.nom := "toto";      // cet appel doit générer une exception
                          // système car l'objet est détruit.
    output.WriteString ("destruction du domaine courant -> arret\n");
    myDomain.Kill;

/* à l'issue de la destruction de l'application, on a :
 - l'objet désigné par a est persistant,
   et est enregistré dans le catalogue,
 - l'objet désigné par b est persistant: il sera détruit au prochain
  ramassage de miettes (sauf si il avait été passé en paramètre),
 - l'objet designe par c et d a ete explicitement detruit. */

  EXCEPT
  ALL FROM SYSTEM: output.WriteString ("exception objet detruit \n");
  END Init;
END Test1.
```

1.7 Instruction ASSERTYPE

```
// test assertype
// --------------

TYPE Refbib IS
  numref : Integer;
  nom : String[15];
  METHOD imprimer ;
END Refbib.

TYPE Refrapport SUBTYPE OF Refbib IS
   universite : String[15];      // surcharge de l'état;
END Refrapport.

TYPE Reflivre SUBTYPE OF Refbib IS
  editeur : String[10];          // surcharge de l'état;
  METHOD Special_livre ;
END Reflivre.

CLASS Refbib IMPLEMENTS Refbib IS
  METHOD imprimer ;
    BEGIN
    output.WriteInteger (numref);
    output.WriteString (nom);
    output.WriteString ("\n");
  END imprimer ;
END Refbib.

CLASS Refrapport SUBCLASS OF Refbib
IMPLEMENTS Refrapport IS
  METHOD imprimer;
  ...
  END imprimer;
END Refrapport.

CLASS Reflivre SUBCLASS OF Refbib
IMPLEMENTS Reflivre IS
  METHOD imprimer ;     // surcharge de la méthode imprimer;
    BEGIN
    SUPER.imprimer;   // impression des champs définis dans la classe
                      // Refbib en appelant la méthode imprimer de la
                      // super-classe Refbib;
    output.WriteString (editeur);
  END imprimer ;
  METHOD Special_livre ;
    BEGIN
    output.WriteString("methode ajoutee dans le type Reflivre \n");
  END Special_livre;
END Reflivre.
```

```
TYPE Test2 IS
  METHOD Init;
END Test2.

CLASS Test2 IMPLEMENTS Test2 IS
  x : REF Refrapport;
  y : REF Reflivre;
  z : REF Testassert;
  METHOD Init;
    BEGIN
    z := Testassert.New;
    x := Refrapport.New;
    x.numref := 5;
    x.nom := "rapport";
    z.M(x);             // appel de M pour tester assertype
                        // avec une référence non conforme
    y:= Reflivre.New;
    y.numref := 10;
    y.nom := "unlivre";
    z.M(y);             // appel de M pour tester assertype
                        // avec une référence conforme (ici égale)
  END Init;
END Test2.

TYPE Testassert IS
  METHOD M (IN a : REF Refbib);
  METHOD Init;
END Testassert.

CLASS Testassert IMPLEMENTS Testassert IS
  METHOD M (IN a : REF Refbib);
    b : REF Reflivre;
    BEGIN
    output.WriteString("appel direct \n");
    a.imprimer;
         // la méthode exécutee sera celle de la classe de l'objet
         // désigné par la référence passée en paramètre, à savoir
         // soit la méthode imprimer de la classe Refrapport,
         // soit celle de la classe Reflivre;
    output.WriteString("appel par assertype \n");
    b := ASSERTYPE (a);
```

A l'issue de cette instruction, la référence b pointe sur l'objet désigné par a uniquement si cet objet est de type Reflivre ou conforme à Reflivre ; si a désigne un objet de type Refrapport, la vérification dynamique de type effectuée lors de l'instruction ASSERTYPE rend faux (pas d'égalité ni de conformité entre les types), ce qui a pour effet de générer l'exception CONFORMANCE_ERROR et d'affecter la valeur NIL à b.

```
    b.imprimer;
    b.Special_livre;
    EXCEPT
      CONFORMANCE_ERROR :
        BEGIN
        output.WriteString("objet d'un mauvais type \n");
        RETURN;
        END;
  END M;
  METHOD Init;
    x: Reflivre;
    y: Refbib;
    BEGIN
    x := Reflivre.New;
    y := Refbib.New;
    SELF.M(x);
    SELF.M(y);
  END Init;
END Testassert.
```

1.8 Instruction BLOCK

```
TYPE T IS
  x : Integer;
  METHOD M (IN i : Integer; IN_OUT io : Integer; OUT o :Integer);
  METHOD Init;
END T.

CLASS C IMPLEMENTS T IS
  y : Integer;
  METHOD M (IN i : Integer; IN_OUT io : Integer; OUT o :Integer);
    z : Integer;
    r : REF D;
    bloc : REF Block;// bloc est une variable référence de type Block
    BEGIN
    //...              // code de la méthode;
    output.WriteString ("creation instance D\n");
    r := D.New;
    z := 2;
    output.WriteString ("creation du bloc\n");
    bloc := BLOCK      // création du bloc et de son contexte;
      t : Integer;     // variable locale au bloc;
      BEGIN
      output.WriteString ("on exectute le bloc\n");
      t := 0;        // permis car t est une variable locale au bloc;
      output.WriteInteger (z);
      output.WriteString ("\n");
      output.WriteInteger (i);
      output.WriteString ("\n");
```

```
      t := z;                                  // permis (z vaut 2)
      z := 0;// permis car z est une variable locale à la méthode M;
      i := i+1;    // permis car i est un paramètre d'entrée de M;
      // o := z;   // interdit car o est un paramètre de sortie de M;
      SELF.x := SELF.x+1;  // permis car x est une variable visible;
    // SELF.y := 0; interdit : y est une variable d'état non visible;
      END;
    //...  // suite du code de la méthode M;
    output.WriteString ("appel de op sur D (bloc passe)\n");
    r.op (bloc);  // appel de la méthode op sur r en passant en
                  // paramètre le bloc créé;
  END M ;
  METHOD Init;
    BEGIN
    SELF.M(4,x,y);
  END Init;
END C.

TYPE D IS
  METHOD op (IN b : REF Block);
END D.

CLASS D IMPLEMENTS D IS
  METHOD op (IN b : REF Block);
    BEGIN
    output.WriteString (" exécution de D : appel du bloc\n");
    b.Value;     // exécution (par appel de la méthode Value) du bloc
                 // dont la référence a ete passee en parametre

    output.WriteString ("fin du l'exécution du bloc : on est dans D\n");
  END op;
END D.
```

1.9 Exceptions

```
TYPE Element IS
END Element.

CLASS Element IMPLEMENTS Element IS
END Element.

TYPE Buffer IS
  METHOD Put (IN elt : REF Element); SIGNALS Full;
  METHOD Get : REF Element;  SIGNALS Empty;
END Buffer.

CLASS Buffer IMPLEMENTS Buffer IS
  CONST max : Integer = 4;
  buffer : Array[max] OF REF Element;
```

```
  n : Integer = 0;
  METHOD Put (IN elt : REF Element); SIGNALS Full;
    BEGIN
    IF (n < max)  THEN
      buffer[n] := elt;
      n := n+1;
    ELSE
      RAISE Full;          //signalement de l'exception  ;
    END;
  END Put;
  METHOD Get : REF Element; SIGNALS Empty;
    BEGIN
    IF (n > 0)  THEN
      n := n-1;
      RETURN buffer[n];
    ELSE
      RAISE Empty;      // signalement de l'exception ;
    END;
  END Get;
END Buffer.

TYPE Buffer_scenario IS
  METHOD Vider_buffer (IN b : REF Buffer);
  METHOD Ajouter_buffer (IN b : REF Buffer);
  METHOD Init ;
END Buffer_scenario.

CLASS Buffer_scenario IMPLEMENTS Buffer_scenario IS
  mon_buffer : REF Buffer;
  y : REF Element;
  METHOD Vider_buffer (IN b : REF Buffer);
    x : REF Element;
    BEGIN
    REPEAT
      x := b.Get;
      x.Destroy;
    UNTIL false;
    EXCEPT
    Empty :
      BEGIN
      output.WriteString("exception Empty, return \n");
      RETURN; // ce retour provoque la fin de la méthode Vider_buffer
      END;
  END Vider_buffer;
  METHOD Ajouter_buffer (IN b : REF Buffer);
    x : REF Element;
    BEGIN
    x := Element.New;
    b.Put(x);
    EXCEPT
```

```
    Full :
      BEGIN
      y := mon_buffer.Get;
      output.WriteString("exception Full, retry \n");
      RETRY;
      END;
  END Ajouter_buffer;
  METHOD Init ;
    i : Integer;
    BEGIN
    mon_buffer := Buffer.New;
    FOR i := 1 TO 4 DO
      SELF.Ajouter_buffer(mon_buffer);
    END;
    output.WriteString("buffer rempli \n");
    SELF.Ajouter_buffer(mon_buffer);
    output.WriteString("insertion en surnombre -> exception \n");
    SELF.Vider_buffer(mon_buffer);
    output.WriteString("buffer vide \n");
  END Init;
END Buffer_scenario.
```

1.10 Généricité

1.10.1 Généricité non contrainte

Nous donnons ci-dessous un exemple de généricité non contrainte avec test de l'interprétation de NIL pour les types élémentaires.

```
TYPE CONSTRUCTOR List1 OF [T] IS
  // T est l'identificateur de substitution. Il pourra être instancié par
  // un paramètre effectif de la forme : Integer, Char, Boolean,
  // LongInt, ShortInt ou REF TypeConstruit.
  METHOD Insert (IN o : T);
  METHOD First : T;
  METHOD Go (IN i : Integer) : T;
END List1.

CLASS CONSTRUCTOR List1 OF [T] IMPLEMENTS List1 IS
  // T est l'identificateur de substitution
  CONST max : Integer = 100;
  list : Array[max] OF T;
  dernier : Integer = 0;
  METHOD Insert (IN o : T);
    BEGIN
    IF o = NIL THEN
      output.WriteString("objet a NIL ou 0");
    END;
    list[dernier] := o;
    dernier := dernier+1;
```

```
  END Insert;
  METHOD First : T;
    BEGIN
    RETURN list[0];
  END First;
  METHOD Go (IN i : Integer) : T;
    BEGIN
    IF i >= 0 AND i < max THEN
      RETURN list[i];
    ELSE
      RETURN NIL;
    END;
  END Go;
END List1.

// exemple d'utilisation

TYPE GenNonContrainte IS
  METHOD Init;
END GenNonContrainte.

CLASS GenNonContrainte IMPLEMENTS GenNonContrainte IS
  METHOD Init;
    r : REF List1 OF Integer;
    c : REF List1 OF Char;
    s : REF List1 OF REF Document;
    BEGIN
    r := List1.New;
    c := List1.New;
    s := List1.New;r.Insert(1);
    r.Insert(0);
    c.Insert('\0');
    s.Insert(NIL);
  END Init;
END GenNonContrainte.
```

1.10.2 Généricité contrainte

Dans cet exemple de généricité contrainte, la classe générique Coll a comme paramètre de généricité une référence vers le type Element. Elle peut donc être utilisée pour créer des classes de collection de références d'Elements et de références de type conforme à Element. Nous donnons ci-dessous deux exemples d'instanciation : la première avec une référence de type Element et la deuxième avec une référence de type Chose (sous-type de Element).

```
TYPE Element IS
  titre : String[20];
END Element.

CLASS Element IMPLEMENTS Element IS
END Element.
```

```
TYPE Chose SUBTYPE OF Element IS
  num : Integer;
END Chose.

CLASS Chose IMPLEMENTS Chose IS
END Chose.

TYPE CONSTRUCTOR Coll OF [ T : REF Element ] IS
  // T est l'identificateur de substitution. Il sera instantié par un
  // paramètre effectif de la forme REF Type, où Type est conforme
  // à Element.
  METHOD Insert( IN x : T) ;
  METHOD Search ( IN nom : String ) : T;
END Coll.

CLASS CONSTRUCTOR Coll OF [ T : REF Element] IMPLEMENTS Coll IS
  // T est l'identificateur de substitution
  CONST max : Integer = 100;
  state : Array[max] OF T;
  dernier : Integer = 0;
  METHOD Insert( IN x : T) ;
    BEGIN
    IF dernier # max     THEN
      state[dernier] := x;
      dernier := dernier+1;
    END;
  END Insert;
  METHOD Search ( IN nom : String ) : T;
    i : Integer;
    BEGIN
    i := 0;
    WHILE (nom # state[i].titre AND i<dernier) DO
      i := i+1;
    END;
    IF i # dernier THEN
      RETURN state[i];
    END;
  END Search;
END Coll.

// le type et la classe ci-dessous utilisent le type et la classe
// générique Coll.

TYPE Gencontrainte IS
  METHOD Init ;
END Gencontrainte.

CLASS Gencontrainte IMPLEMENTS Gencontrainte IS
  METHOD Init;
    coll1 : REF Coll OF [REF Element];
```

```
    coll2 : REF Coll OF [REF Chose];
    e, f : REF Element;
    c : REF Chose;
    BEGIN
    coll1 := Coll.New;
    coll2 := Coll.New;
    e := Element.New;
    c := Chose.New;
    e.titre := "toto";
    c.titre := "titi";
    c.num := 34;
    coll1.Insert(e);
    coll2.Insert(c);
    f := coll1.Search("toto");
    output.WriteString (f.titre);
    output.WriteString ("\n");
  END Init;
END Gencontrainte.
```

1.10.3 Généricité sur les paramètres de dimensionnement

Nous donnons ci-dessous deux exemples de généricité portant sur les paramètres de dimensionnement : la classe Tableau et la classe Chaine.

```
// Exemple de généricité portant sur un parametre type et un paramètre de
// dimensionnement

TYPE CONSTRUCTOR Tableau OF [ T ] IS
  METHOD Put (IN i : Integer; o : T); SIGNALS depass;
  METHOD Get (IN i : Integer) : T; SIGNALS depass;
END Tableau.

CLASS CONSTRUCTOR Tableau [size] OF [ T ]IMPLEMENTS Tableau IS
  // size est un identificateur de substitution ;
  contenu : Array[size] OF T;
  METHOD Put (IN i : Integer; o : T) ; SIGNALS depass;
    BEGIN
    IF i < 0 AND i >= size THEN
      RAISE depass;
    ELSE
      contenu[i] := o;
    END;
  END Put;
  METHOD Get (IN i : Integer) : T ; SIGNALS depass;
    BEGIN    IF i < 0 AND i >= size THEN
      RAISE depass;
    ELSE
      RETURN contenu[i];
    END;
  END Get;
END Tableau.
```

```
// Exemple de généricité ne portant que sur le paramètre de
// dimensionnement

TYPE Chaine IS
  METHOD Put (IN i : Integer; o : Char); SIGNALS depass;
  METHOD Get (IN i : Integer) : Char; SIGNALS depass;
END Chaine.

CLASS CONSTRUCTOR Chaine [size] IMPLEMENTS Chaine IS
  // size est un identificateur de substitution ;
  contenu : String[size];
  METHOD Put (IN i : Integer; o : Char) ; SIGNALS depass;
    BEGIN
    IF i < 0 AND i >= size THEN
      RAISE depass;
    ELSE
      contenu[i] := 0;
    END;
  END Put;
  METHOD Get (IN i : Integer) : Char ; SIGNALS depass;
    BEGIN
    IF i < 0 AND i >= size THEN
      RAISE depass;
    ELSE
      RETURN contenu[i];
    END;
  END Get;
END Chaine.

// Exemple d'utilisation de ces deux classes génériques

TYPE Gendim IS
METHOD Init;
END Gendim.

CLASS Gendim IMPLEMENTS Gendim IS
  METHOD Init;
    // exemples d'utilisation
    r : REF Tableau OF Integer;
    s : REF Chaine;
    i : Integer;
    c : Char;
    BEGIN
    r := Tableau[100].New;
    s := Chaine[100].New;
    r.Put(1,23);
    i := r.Get(1);
    r.Put(100,20);
    s.Put(1,'a');
```

```
    c := s.Get(1);
  END Init;
END Gendim.
```

2 Programmes de la bibliothèque Guide

Afin de ne pas surcharger le document, nous ne donnons que la définition des types pour Catal, List et NS. Comme dans la version actuelle, chacun de ces types n'est réalisé que par une seule classe, le même nom sert à désigner le type et la classe (par exemple le type List est réalisé par la classe List).

Ces types et ces classes peuvent être redéfinies par le programmeur, ou peuvent être hérités pour définir de nouvaux types et classes plus adaptés.

2.1 List

Le type et la classe génériques List permettent de définir des objets listes de n'importe quel type. La forme de généricité utilisée est la généricité non contrainte avec un paramètre de dimensionnement (la taille maximum de la liste). Pour des raisons de simplicité, la réalisation de la liste est faite dans cette version sous forme de tableau.

```
TYPE CONSTRUCTOR List OF [ T ] IS
// T est le paramètre formel de généricité
/* lors de l'accès direct à un élément de la liste, le paramètre
    d'entrée i est le numéro de l'élément (valeur de 1 à max). */
  METHOD First;
  // positionner le pointeur courant sur le premier élément.
  METHOD First : T;
  /* positionner le pointeur courant sur le premier élément et donner
    cet élément. Si la liste est vide, NIL est renvoyé.*/

  METHOD Next;
  // positionner le pointeur courant sur l'élément suivant.
  METHOD Next : T;
  /* positionner le pointeur courant sur l'élément suivant. Si la
    liste est vide, NIL est renvoyé. Si l'élément est le dernier de la
    liste, NIL est renvoyé et le pointeur courant n'est pas déplacé */

  METHOD Current : T;
  // donner l'élément courant.  Si la liste est vide, NIL est renvoyé.

  METHOD Previous;
  /* positionner le pointeur courant sur l'élément précédent. Si
    l'élément est le premier de la liste, le pointeur courant n'est pas
    déplacé */
  METHOD Previous: T;
  /* positionner le pointeur courant sur l'élément précédent. Si la
    liste est vide, NIL est renvoyé. Si l'élément est le premier de la
    liste, NIL est renvoyé et le pointeur courant n'est pas déplacé */

  METHOD Go (IN i : Integer);
  /* positionner le pointeur sur le ième élément. Si la liste est vide
    l'exception EMPTY est signalée. Si le ième élément demandé est
    après le dernier élément de la liste, l'exception OUT est signalée
    et le pointeur courant n'est pas déplacé */
  METHOD Go (IN i : Integer) : T;
  /* positionner le pointeur sur le ième élément et donner cet
    élément. Si la liste est vide, NIL est renvoyé. Si le ième élément
    demandé est après le dernier élément de la liste, NIL est renvoyé
    et le pointeur courant n'est pas déplacé */
```

```
  METHOD Last;
  // positionner le pointeur courant sur le dernier élément.
  METHOD Last : T;
  /* positionner le pointeur courant sur le dernier élément et donner
    cet élément. Si la liste est vide, NIL est renvoyé */

  METHOD NbItems : Integer;
  /* rendre le nombre courant d'éléments de la liste. Si la liste est
    vide, 0 est renvoyé. */

  METHOD Insert (IN o : T);
  /* insérer l'objet à la position donnée par le pointeur courant et
    le pointeur courant pointe l'élément suivant l'élément inséré. Si
    la liste est pleine, l'élément donné n'est pas inséré et le
    pointeur courant n'est pas déplacé */

  METHOD Append (IN o : T);
  /* insérer l'objet après la position donnée par le pointeur courant
    et le pointeur courant pointe l'élément inséré. Si la liste est
    pleine, l'élément donné n'est pas inséré et le pointeur courant
    n'est pas déplacé */

  METHOD Delete;
  /* enlever de la liste l'objet designé par le pointeur courant et
    positionner le pointeur courant sur l'objet suivant l'objet
    supprimé. Si l'objet enlevé est le dernier de la liste, le pointeur
    courant est positionné sur le dernier élément de la liste */
  METHOD Delete : T;
  /* enlever de la liste l'objet désigné par le pointeur courant,
    positionner le pointeur courant sur l'objet suivant l'objet
    supprimé et rendre l'objet pointé. Si la liste est vide, NIL est
    renvoyé. Si l'objet enlevé est le dernier de la liste, le pointeur
    courant est positionné sur le dernier élément de la liste et ce
    dernier élément est renvoyé */
  METHOD Delete(IN i : Integer);
  /* enlever de la liste le ième objet de la liste et positionner le
    pointeur courant sur le ième élément. Si l'objet enlevé est le
    dernier de la liste, le pointeur courant est positionné sur le
    dernier élément de la liste. Si le ième élément demandé est après
    le dernier élément de la liste, le pointeur courant n'est pas
    déplacé */
  METHOD Delete(IN i : Integer) : T;
  /* enlever de la liste le ième objet de la liste, positionner le
    pointeur courant sur le ième élément et rendre cet élément.Si la
    liste est vide, NIL est renvoyé. Si l'objet enlevé est le dernier
    de la liste, le pointeur courant est positionné sur le dernier
    élément de la liste et ce dernier élément est renvoyé. Si le ième
    élément demandé est après le dernier élément de la liste, NIL est
    renvoyé et le pointeur courant n'est pas déplacé */

  METHOD Concat(IN l : List OF [ T ]);
  /* concaténer la liste passée en paramètre après la liste courante,
    le pointeur courant désigne le dernier élément de la nouvelle
    liste. Si l'on ne peut pas concaténer tous les éléments de la liste
    l, la liste courante n'est pas modifiée */

END List.
```

2.2 Catal

Le type Catal et la constante référence catal permettent d'accéder au catalogue des objets : ce catalogue a pour fonction d'associer à tout objet permanent un nom symbolique fourni par le programmeur, de durée de vie indépendante des programmes.

```
TYPE Catal IS
  father : REF Catal = SELF;
  // donne le répertoire de niveau supérieur ('..' pour Unix)
  root : REF Catal = SELF;
  // donne le répertoire racine ('/' pour Unix)
  nbEntries : Integer = 0;
  // donne le nombre d'objets insérés dans le répertoire

  METHOD ChangeOwner(IN newownername, localname : String) : Boolean;
  // changer le propriétaire d'un objet ou d'un répertoire

  METHOD CreateDir (IN localname, ownername : String; OUT dir : REF
    Catal) : Boolean;
  // créer un nouveau répertoire dans le répertoire courant

  METHOD Delete(IN localname : String; OUT object : REF Top) :
    Boolean;
  METHOD Delete(IN localname : String; OUT object : REF Top);
  METHOD Delete(IN localname : String);
  // détruire l'association entre un objet et un nom symbolique

  METHOD DeleteDir(IN localname : String; OUT dir : REF Catal) :
    Boolean;
  // détruire le répertoire de nom "localname" dans le répertoire
  // courant

  METHOD Find(IN object : REF Top) : String;
  METHOD Find(IN object : REF Top; OUT name : String) : Boolean;
  METHOD RecursiveFind(IN object : REF Top; pathname : String;
    pathtable : Array[10] OF Catal): String;
  // retrouver le nom symbolique de l'objet dont la référence est
  // passée en paramètre

  METHOD Init(IN dad, slash : REF Catal);
  // initialiser ou modifier le père et le répertoire du catalogue

  METHOD Insert(IN localname : String; object : REF Top);
  METHOD Insert(IN localname : String; object : REF Top) : Boolean;
  METHOD Insert(IN localname : String; owner : String; object : REF
    Top);
  METHOD Insert(IN localname : String; owner : String; object : REF
    Top) : Boolean;
  // insérer une nouvelle entrée dans le catalogue

  METHOD List;
  METHOD List(IN filter : String);
  // lister le contenu du répertoire courant
  // le filtre permet d'en sélectionner un sous-ensemble

  METHOD SearchClass(IN localname : String) : REF Top;
  // rechercher la référence de la classe dont le nom symbolique
  //  est passé en paramètre

  METHOD Search(IN localname : String) : REF Top;
  // rechercher la référence de l'objet dont le nom symbolique est
  // passé en paramètre
```

```
  METHOD RecursiveSearch(IN localname, pathname : String ; pathtable :
    Array[10] OF REF Catal);
  // lister tous les objets d'une arborescence qui ont le même nom
  // symbolique

  METHOD SearchDir(IN localname : String) : REF Catal;
  // rechercher la référence d'un répertoire local dont le nom
  // symbolique est passé en paramètre.

END Catal.
```

2.3 NS : Name Server

NS est le serveur de noms implémenté dans le système Guide. Il offre un service équivalent à celui d'Unix : création de répertoires hiérarchiques, déplacement dans la hiérarchie des répertoires, recherche sélective, insertion et retrait d'objets.

```
TYPE NS IS
  METHOD ChangeDir(IN pathname : String) : Boolean;
  /* changement du répertoire courant ; le pathname passé en paramètre
    est soit absolu (relatif à "/"), soit relatif au répertoire courant
    ; si c'est une chaîne vide, le répertoire courant est positionné au
    répertoire "home" */

  METHOD Find(IN pathname, name : String);
  /* affichage à l'écran de tous les noms d'un objet ou d'un
    répertoire */

  METHOD Where(IN pathname, name : String);

  METHOD Init(IN nsroot : REF Catal; homepathname : String; ownername
    : String) : Boolean;
  // initialisation du home directory et du nom de propriétaire.

  METHOD Link(IN name, newname : String) : Boolean;
  // création d'un lien entre un nom existant et un nouveau nom.

  METHOD ListDir (IN pathname, filter : String) : Boolean;
  /* affichage des objets et des répertoires contenu dans le
    répertoire spécifié ; le paramètre filtre permet d'en sélectionner
    un sous-ensemble. */

  METHOD LookDir(IN pathname : String; OUT dir : REF Catal) : Boolean;
  /* recherche d'un répertoire dont le pathname est donné en
    paramètre ; sa référence est donnée en retour. */

  METHOD Lookup(IN name : String; OUT object : REF Top) : Boolean;
  /* recherche d'un objet dont le nom est donné en paramètre ; sa
    référence est donnée en retour. */

  METHOD MakeDir(IN pathname : String; OUT dir : REF Catal) : Boolean;
  // création d'un nouveau répertoire.

  METHOD PrintDir; // affichage du répertoire courant.

  METHOD Register(IN name : String; object : REF Top) : Boolean;
  // enregistrement de l'association d'un nom et d'un objet.

  METHOD Remove(IN name : String; OUT object : REF Top) : Boolean;
  // destruction de l'association d'un nom et d'un objet.

  METHOD RemoveDir(IN pathname: String; OUT dir: REF Catal) : Boolean;
  // destruction de l'association d'un nom et d'un répertoire.

  METHOD Rename(IN name, newname : String) : Boolean;
  // changement du nom d'un objet.

  METHOD RenameDir(IN name, newname : String) : Boolean;
  // changement du nom d'un répertoire.
END NS.
```

3 Exemples d'application

Les applications décrites ci-dessous sont de taille modeste de façon à rester lisibles. Ceci est dû entre autres à l'interface utilisateur qui est simplifiée. Il existe pour certaines de ces applications une version utilisant l'interface Motif ce qui permet d'avoir de vrais programmes de démonstration.

3.1 Les tours de Hanoï

Ce programme illustre comment le langage Guide permet d'écrire des applications indépendamment de leur configuration d'exécution (centralisée ou répartie). En effet, ce programme peut aussi bien s'exécuter sur un site (les trois piquets sont affichés sur l'écran d'un même site), deux ou trois sites (chaque piquet est affiché sur un site différent).

Pour permettre cela, on associe à chaque piquet un canal de sortie (désigné par la variable référence `screen`) qui lors de l'initialisation du piquet, est créé sur le site désiré (les noms des sites sont saisis au lancement du programme) : un des paramètres possibles de la méthode New est le nom du site de création de l'objet.

```
screen := Chan.New(site);
```

Cette application est décomposée en trois ensembles d'objets :

- Les disques (type et classe Disk) dont la seule opération est l'affichage à l'écran (à la position courante du curseur),
- Les piquets (type et classe Picket) sur lesquels on peut ajouter un disque (méthode Put) ou enlever le disque situé au sommet (méthode Get),
- L'objet de classe Hanoï qui réalise l'algorithme proprement dit (procédure récursive Hanoi_simulation) ainsi que l'initialisation.

```
// les objets de type Disk sont caractérisés par leur taille
//  définie par la variable d'état visible value (de 0 :disque
//  vide, à 10 : disque le plus large).
TYPE Disk IS
  value: Integer;
  METHOD Print ( IN screen : REF ChanOut );
END Disk.

CLASS Disk
  IMPLEMENTS Disk IS

  METHOD Print( IN screen : REF ChanOut );
    BEGIN
      CASE value OF
      0 : screen.WriteString ("           |           \n"); END;
      1 : screen.WriteString ("         /|\\\n"); END;
      2 : screen.WriteString ("        /o|o\\\n"); END;
      3 : screen.WriteString ("       /oo|oo\\\n"); END;
      4 : screen.WriteString ("      /ooo|ooo\\\n"); END;
      5 : screen.WriteString ("     /oooo|oooo\\\n"); END;
      6 : screen.WriteString ("    /ooooo|ooooo\\\n"); END;
      7 : screen.WriteString ("   /oooooo|oooooo\\\n"); END;
      8 : screen.WriteString ("  /ooooooo|ooooooo\\\n"); END;
      9 : screen.WriteString (" /oooooooo|oooooooo\\\n"); END;
      10 : screen.WriteString("/ooooooooo|ooooooooo\\\n"); END;
```

```
      END;
    END Print;

END Disk.

TYPE Picket IS
  METHOD Init ( IN site : String; num, line, colon: Integer );
  METHOD Put ( IN REF Disk );
  METHOD Get ( OUT REF Disk );
  METHOD Print;
END Picket.

CLASS Picket
  IMPLEMENTS Picket IS

  screen : REF ChanOut;  // canal d'affichage du piquet
  number : Integer;
  top : Integer;
  list_picket : Array [11] OF REF Disk;
  empty : REF Disk;
  init_line : Integer;
  init_colon : Integer;

  METHOD Init ( IN site : String; num, line, colon: Integer );
    BEGIN
      screen := Chan.New( site ); // création du canal d'affichage
                                  // sur le site spécifié
      screen.Clear;
      number := num;
      top := 0;
      list_picket[0] := Disk.New;
      list_picket[0].value := 1;
      empty := Disk.New;
      empty.value := 0;
      init_line := line;
      init_colon := colon;
    END Init;

  METHOD Put ( IN d : REF Disk );  // ajout d'un disque
                                   // et mise à jour du curseur
    BEGIN
      top := top+1;
      list_picket[ top ] :=  d;
      screen.Goto( init_line-top, init_colon );
      d.Print( screen );
    END Put;

  METHOD Get(OUT d : REF Disk); // retrait du disque le plus haut
                                // et mise à jour du curseur
    BEGIN
      d := list_picket[ top ];
      screen.Goto( init_line-top, init_colon );
      top := top-1;
      empty.Print( screen );
    END Get;

  METHOD Print;                  // affichage de tous les disques
    i : Integer;
    d : REF Disk;
    BEGIN
      i := number+1;
      WHILE i # 0
      DO
        screen.Goto( init_line-i, init_colon );
```

```
        IF i <= top
        THEN
          d := list_picket[ i ];
          d.Print( screen );
        ELSE
          empty.Print( screen );
        END;
        i := i-1;
      END;
    END Print;

END Picket.

TYPE Hanoi IS
  METHOD Init;
END Hanoi.

CLASS Hanoi
  IMPLEMENTS Hanoi IS

  chosen_number : Integer;
  picket_1, picket_2, picket_3 : REF Picket;

  PROCEDURE Ask_for_number;  // choix du nombre de disques
    BEGIN
      output.WriteString ("*** Guide Test ***\n\n");
      output.WriteString ("hanoi simulation\n\n");
      WHILE (10 < chosen_number OR chosen_number < 1)
      DO
        output.WriteString ("how many disk ? ");
        chosen_number := input.ReadInteger;
        IF chosen_number <= 0 THEN
          output.WriteString ("\nnumber should  be positive\n");
        END;
        output.WriteString ("\n");
        IF 10 < chosen_number THEN
          output.WriteString("\nnumber should be less than 11\n");
        END;
      END;
    END Ask_for_number;

  // algorithme récursif de déplacement des disques
  PROCEDURE Hanoi_simulation (IN n : Integer; p1, p2, p3 : REF
    Picket);
    d : REF Disk;
    BEGIN
      IF n>1
      THEN
        Hanoi_simulation( n-1, p1, p3, p2 );
        Hanoi_simulation( 1, p1, p2, p3 );
        Hanoi_simulation( n-1, p3, p2, p1 );
      ELSE
        p1.Get( d );
        p2.Put( d );
      END;
    END Hanoi_simulation;

  PROCEDURE Hanoi_init;   // initialisation : choix des sites
                          // création des piquets et des disques
                          // et appel de l'algorithme récursif
    i : Integer;
    d : REF Disk;
    site1, site2, site3 : String[80];
    BEGIN
```

```
   output.WriteString ("Creation site of the first picket?: ");
   site1 := input.ReadString(80);
   IF site1=""
   THEN
     site1 := "uvea";
     output.WriteString( "---> " );
     output.WriteString( site1 );
     output.WriteString( "\n" );
   END;
   output.WriteString("Creation site of the second picket?: ");
   site2 := input.ReadString(80);
   IF site2=""
   THEN
     site2 := site1;
     output.WriteString( "---> " );
     output.WriteString( site2 );
     output.WriteString( "\n" );
   END;
   IF chosen_number = 1
   THEN
     output.WriteString( "---> site1 : " );
     output.WriteString( site1 );
     output.WriteString( "\n" );
     output.WriteString( "---> site2 : " );
     output.WriteString( site2 );
     output.WriteString( "\n" );
     output.WriteString( "Significant optimization :\n" );
     output.WriteString( "two pickets are enough to solve
     output.WriteString( "your problem\n" );
     output.WriteString( "Ok ? " );
     site3 := input.ReadString(80);
     picket_1 := Picket.New( site1 );
     picket_2 := Picket.New( site2 );
     picket_1.Init( site1, chosen_number, chosen_number+6, 0 );
     picket_2.Init( site2, chosen_number, chosen_number+6,24 );
     d := Disk.New;
     d.value := 1;
     picket_1.Put( d );
      picket_1.Print;
      picket_2.Print;
     Hanoi_simulation(chosen_number, picket_1, picket_2,
picket_3);
   ELSE
     output.WriteString("Creation site of the third picket: ");
     site3 := input.ReadString(80);
     IF site3=""
     THEN
       site3 := site2;
       output.WriteString( "---> " );
       output.WriteString( site3 );
       output.WriteString( "\n" );
     END;
     picket_1 := Picket.New( site1 );
     picket_2 := Picket.New( site2 );
     picket_3 := Picket.New( site3 );
     picket_1.Init( site1, chosen_number, chosen_number+6, 0 );
     picket_2.Init( site2, chosen_number, chosen_number+6,24 );
     picket_3.Init( site3, chosen_number, chosen_number+6, 48);
     i := 0;
     WHILE i # chosen_number
     DO
       d := Disk.New;
       d.value := chosen_number-i;
       picket_1.Put( d );
```

```
          i := i+1;
        END;
        picket_1.Print;
        picket_2.Print;
        picket_3.Print;
        Hanoi_simulation( chosen_number, picket_1, picket_2,
    picket_3 );
      END;
    END Hanoi_init;

  METHOD Init;        // lancement de l'application
    BEGIN
      Ask_for_number;
      Hanoi_init;
      output.Goto( chosen_number+7, 0 );
    END Init;

END Hanoi.
```

3.2 Les philosophes et les spaghetti

Cet exemple permet d'illustrer l'utilisation des conditions de synchronisation pour l'accès en exclusion mutuelle aux fourchettes.

```
////////////////////////////////////////////////////////////////
// la table contient 5 fourchettes et les gère de façon à ce
// que le philosophe i prenne les fourchettes i et i+1 si
// elles sont libres
////////////////////////////////////////////////////////////////

TYPE Table IS
  METHOD Init;
  METHOD GetFourchettes(IN i : Integer);
  METHOD FreeFourchettes(IN i : Integer);
END Table.

CLASS Table
IMPLEMENTS Table IS

  CONST NbPhilosophe : Integer = 5;
  Fork : Array [NbPhilosophe] OF Boolean;

  METHOD GetFourchettes(IN i : Integer);
  BEGIN
    Fork[(i + 1) MOD NbPhilosophe] := FALSE;
    Fork[i MOD NbPhilosophe] := FALSE;
  END GetFourchettes;

  METHOD FreeFourchettes(IN i : Integer);
  BEGIN
    Fork[(i + 1) MOD NbPhilosophe] := TRUE;
    Fork[i MOD NbPhilosophe] := TRUE;
  END FreeFourchettes;

  METHOD Init;
  i : Integer;
  BEGIN
    FOR i := 0 TO NbPhilosophe - 1 DO
      Fork[i] := TRUE;
    END;
  END Init;

  CONTROL
    GetFourchettes :
      current(GetFourchettes)=0 AND current(FreeFourchettes)=0
      AND Fork[i MOD NbPhilosophe] AND Fork[(i + 1) MOD NbPhilosophe];
    FreeFourchettes:
      current(GetFourchettes)=0 AND current(FreeFourchettes)=0;

END Table.

////////////////////////////////////////////////////////////////
// canal permettant d'illustrer le comportement des philosophes
// et accédé en mutuelle exclusion
////////////////////////////////////////////////////////////////

TYPE IoMutex IS
  METHOD Init (IN r : REF Table);
  METHOD Illustrer (IN id, MyId : Integer);
END IoMutex.
```

```
CLASS IoMutex
IMPLEMENTS IoMutex IS

  CONST NbPhilosophe : Integer = 5;
  X : Array [NbPhilosophe+1] OF Integer;
  Y : Array [NbPhilosophe+1] OF Integer;
  table : Table;
  spaghetti : Integer=100; // quantite de spaghetti presentes...

  METHOD Init (IN r : REF Table);
  BEGIN
    table := r;
    X[0] := 37; Y[0] := 6;
    X[1] := 26; Y[1] := 12;
    X[2] := 26; Y[2] := 18;
    X[3] := 47; Y[3] := 18;
    X[4] := 47; Y[4] := 12;
    X[5] := 37; Y[5] := 15; // coordonnees de la table
    output.Clear;
    output.Goto (30, 0);
    output.WriteString("     Legende :\n");
    output.WriteString("         je pense : ????\n");
    output.WriteString("         j'attend : WAIT\n");
    output.WriteString("         je mange : MIAM\n");
    output.WriteString("\n\n\n\n");
  END Init;

  METHOD Illustrer (IN id, MyId : Integer);
  i, j : Integer;
  BEGIN
    j := MyId MOD 5;
    FOR i := 0 TO 2 DO
      output.Goto (Y[j] + i, X[j]);
      CASE id OF
        0 : output.WriteString("????"); END;
        1 : output.WriteString("WAIT"); END;
        2 : output.WriteString("MIAM");
            IF spaghetti > 0
            THEN
              spaghetti := spaghetti-1; // on mange un spaghetti
              output.Goto(Y[5],X[5]);
              output.WriteString("    ");
              output.Goto(Y[5],X[5]);
              output.WriteInteger(spaghetti);
            ELSE
              catal.Delete("table");
              catal.Delete("canal");
              output.Goto(40,0);
              myDomain.Kill;
            END;
          END;
      END;
    END;
  END Illustrer;

  CONTROL
    EXCLUSIVE(Illustrer);

END IoMutex.

////////////////////////////////////////////////////////////////
// un philosophe  boucle indéfiniement sur le cycle d'actions :
//    je veux manger
//    je mange
```

```
//    je pense
////////////////////////////////////////////////////////////////

TYPE Philosophe IS
  METHOD Init(IN i : Integer);
END Philosophe.

CLASS Philosophe
IMPLEMENTS Philosophe IS
  MyId : Integer;
  r : REF Table;
  canal : REF IoMutex;

  METHOD Init(IN i : Integer);
  BEGIN
    MyId := i;

    canal := catal.Search("canal");
    IF canal = NIL THEN
      // le canal d'entree/sortie n'existe pas
      myDomain.Kill;
    END;

    r := catal.Search("table");
    IF r = NIL THEN
      // la table n'existe pas...
      myDomain.Kill;
    END;

    WHILE TRUE DO
      canal.Illustrer(0,MyId); // affiche que l'on pense
      SYSTEM("FCT_guideSleep(FCT_random() % 13);");

      canal.Illustrer(1, MyId); // affiche que l'on veut manger
      r.GetFourchettes(MyId);

      canal.Illustrer(2, MyId); // affiche que l'on mange
      SYSTEM("FCT_guideSleep(FCT_random() % 17);");
      r.FreeFourchettes(MyId);
    END;
  END Init;

  CONTROL
    EXCLUSIVE(Init);

END Philosophe.

////////////////////////////////////////////////////////////////
// classe permettant de lancer l'application
////////////////////////////////////////////////////////////////
TYPE TestPhilosophe IS
  METHOD Init;
END TestPhilosophe.

CLASS TestPhilosophe
IMPLEMENTS TestPhilosophe IS

  //_____________________________________________________________
  // Elle crée une table avec cinq fouchettes puis elle
  // invite les cinq philosophes à manger.
  //_____________________________________________________________

  METHOD Init;
  CONST NbPhilosophe : Integer = 5;
```

```
  table : REF Table;
  canal : REF IoMutex;
  i : Integer;
  ph : Array [NbPhilosophe] OF REF Philosophe;
  BEGIN
    // Creation and registration of the table
    table := catal.Search("table");
    IF table = NIL THEN
      table := Table.New;
      table.Init;
      catal.Insert ("table", table);
    END;

    canal := catal.Search("canal");
    IF canal = NIL THEN
      canal := IoMutex.New;
      canal.Init( table );
      catal.Insert ("canal", canal);
    END;

    FOR i := 0 TO NbPhilosophe-1 DO
      ph[i] := Philosophe.New;
    END;

    CO_BEGIN
      b : ph[0].Init( 0 );
      c : ph[1].Init( 1 );
      d : ph[2].Init( 2 );
      k : ph[3].Init( 3 );
      f : ph[4].Init( 4 );
    CO_END ;
  END Init;

END TestPhilosophe.

// Exemple de trace d'exécution
//   ???? pense
//   WAIT veut manger
//   MIAM mange
//
//
//                                      ????
//                                      ????
//                                      ????
//
//
//
//                          MIAM                WAIT
//                          MIAM                WAIT
//                          MIAM                WAIT
//
//
//
//                          WAIT                MIAM
//                          WAIT                MIAM
//                          WAIT                MIAM
//
//
```

Exemples de programmes en langage Guide

1 Exemples simples . . . 1
- 1.1 Définition d'un type . . . 1
- 1.2 Définition d'une classe . . . 2
- 1.3 Sous-typage et héritage . . . 3
  - 1.3.1 Exemple de relation de sous-typage portant sur les variables visibles . . . 3
  - 1.3.2 Exemple de relation de sous-typage portant sur les méthodes . . . 3
  - 1.3.3 Exemple de relation d'héritage . . . 3
- 1.4 Utilisation de la conformité . . . 4
  - 1.4.1 Conformité induite par une relation de sous-typage . . . 4
  - 1.4.2 Conformité correspondant à une relation de restriction . . . 5
- 1.5 Parallélisme et synchronisation . . . 8
  - 1.5.1 Gestion d'un tampon . . . 8
  - 1.5.2 Control . . . 11
- 1.6 Persistance, destruction explicite d'objets, destruction d'un domaine . . . 15
- 1.7 Instruction ASSERTYPE . . . 16
- 1.8 Instruction BLOCK . . . 18
- 1.9 Exceptions . . . 19
- 1.10 Généricité . . . 21
  - 1.10.1 Généricité non contrainte . . . 21
  - 1.10.2 Généricité contrainte . . . 22
  - 1.10.3 Généricité sur les paramètres de dimensionnement . . . 24

2 Programmes de la bibliothèque Guide . . . 27
- 2.1 List . . . 27
- 2.2 Catal . . . 29
- 2.3 NS : Name Server . . . 31

3 Exemples d'application . . . 32
- 3.1 Les tours de Hanoï . . . 32
- 3.2 Les philosophes et les spaghetti . . . 37