Master Informatique 2ème année

Option Programmation Temps-Réel

ADA

Rémi COZOT – cozot@irisa.fr

2005

IRISA

PLAN
Base

- Introduction
  - Historique
  - ADA
  - Utilisation d'ADA aujourd'hui
  - Objet et limites du cours
- Éléments de base du langage
  - Package
  - Type
  - Structure de contrôle
  - Fonction et procédure
  - Autres

PLAN
Temps réel

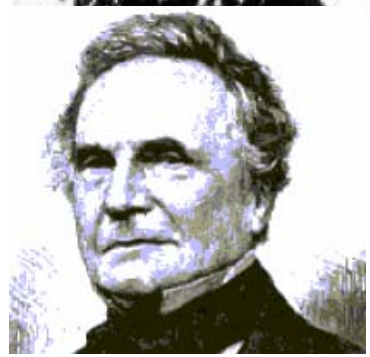
- Éléments temps réel
  - Objet tâche et type tâche
  - Gestion du temps
  - Synchronisation directe par rendez vous
  - Synchronisation sélective
  - Objet protégé
  - Avortement

Historique

- Ministère Américain de la Défense (DoD)
  - Étude 1974
    - Surcoût des projets du aux logiciels
    - Première approche du génie logiciel
- Cahier des charges
  - Langage pour faire du code « propre »
    - Réduire les erreurs
    - Favoriser la réutilisation
  - Temps réel et systèmes embarqués
    - Remplacement des langages et dialectes temps réels
      - JOVIAL j73
      - CORAL 66
      - RTL/2
- A l'époque le langage de référence FORTRAN

INTRODUCTION
ADA

- DoD
  - Choix en mai 1979
  - Normalisation
    - DOD
    - ANSI (American National Standard Institute) en 1983 ADA
    - ISO en 1987
- ADA
  - Augusta Ada Byron, comtesse de Loveplace 1815-1852
    - Assistante de Charles Babbage
      - Machine analytique mécanique
    - Première programmeuse
- ADA 95
  - Révision ADA83
  - Objet



Utilisation d'ADA

- Domaine
  - Aéronautique, spatiale et automobile
  - Temps Réel Critique, Logiciel embarqué
- Entreprises et projets
  - EADS
    - Airbus A380
    - Satellites (SPOT, Helios)
  - Ariane Espace
    - Ariane 5
  - SNECMA
    - Contrôle moteur
- Concurrence
  - C++
  - JAVA

Objet et limites du cours

- Objet
  - Programmation temps réel haut niveau en ADA
  - Aspects temps réel
    - Type **TASK**
    - Rendez vous ADA i.e. synchronisation de tâches
  - Éléments de base du langage
    - ADA 83 plus ADA 95
- Limites
  - Survol du langage, 4 heures de cours
  - Presque rien sur les aspects Objets
  - Recherche personnelle pour approfondir

Package

- Packages permettent un code structuré
  - On regroupe dans le package des choses de même « nature »
  - Ancêtres des objets, Programmation ***Orienté*** Objet
- Packages permettent une meilleure réutilisation
- Packages permettent une compilation séparée
  - Comme en JAVA
  - Compilation des packages
  - Dépendance entre packages
- Packages ont
  - Une interface visible de l'extérieur
  - Un corps (body) caché de l'extérieur
- Packages contiennent
  - Fonctions, Types, Constantes, Variables, ...

Package

- Utilisation des packages
  - Déclaration des packages utilisés

```
with my_package ;
```

  - Utilisation d'une fonction

```
my_package.mpFonction_1 (…) ;
```

  - Ou, si le package est fortement utilisé

```
use my_package ;
mpFonction_1 (…) ;
```

    - Attention cela nuit à la clarté du code, car on ne sait pas d'où vient la fonction

Package

- Codage de l'interface

```
package PILE is                          -- spécification
                                         -- EMPILER et DEPILER visibles
                                         -- de l'extérieur

   procedure EMPILER(X: in Integer) ;

   function DEPILER return Integer ;

end PILE ;
```

Package

- Codage du corps

```
Package body PILE is                          -- corps

   MAX : constant := 100 ;                    -- partie non visible
   P : array (1 .. MAX) of Integer;
   HAUT : Integer range 0..MAX ;

                                              -- partie visible
                                              -- car dans l'interface
   procedure EMPILER(X: in Integer) is
   begin
     HAUT := HAUT + 1;
     P(HAUT) := X;
   end EMPILER;
```

Package

```
function DEPILER return Integer is
   begin
     HAUT := HAUT-1 ;
     return P(HAUT+1) ;
   end DEPILER

begin                                   -- initialisation du package

HAUT := 0 ;

end PILE ;
```

Package générique

- Codage de l'interface

```
generic
   MAX : Integer;
   type Item is private ;

package PILE is                          -- spécification
                                         -- EMPILER et DEPILER visibles
                                         -- de l'extérieur

   procedure EMPILER(X: in Item) ;

   function DEPILER return Item ;

end PILE ;
```

Package générique

- Codage du corps

```
Package body PILE is                                    -- corps

                                                        -- partie non visible
   P : array(1 .. MAX) of Item;
   HAUT : Integer range 0..MAX ;

                                                        -- partie visible
                                                        -- car dans l'interface
   procedure EMPILER(X: in Item) is
   begin
     HAUT := HAUT + 1;
     P(HAUT) := X;
   end EMPILER;
```

Package générique

```
function DEPILER return Item is
   begin
     HAUT := HAUT-1 ;
     return P(HAUT+1) ;
   end DEPILER

begin                                   -- initialisation du package

HAUT := 0 ;

end PILE ;
```

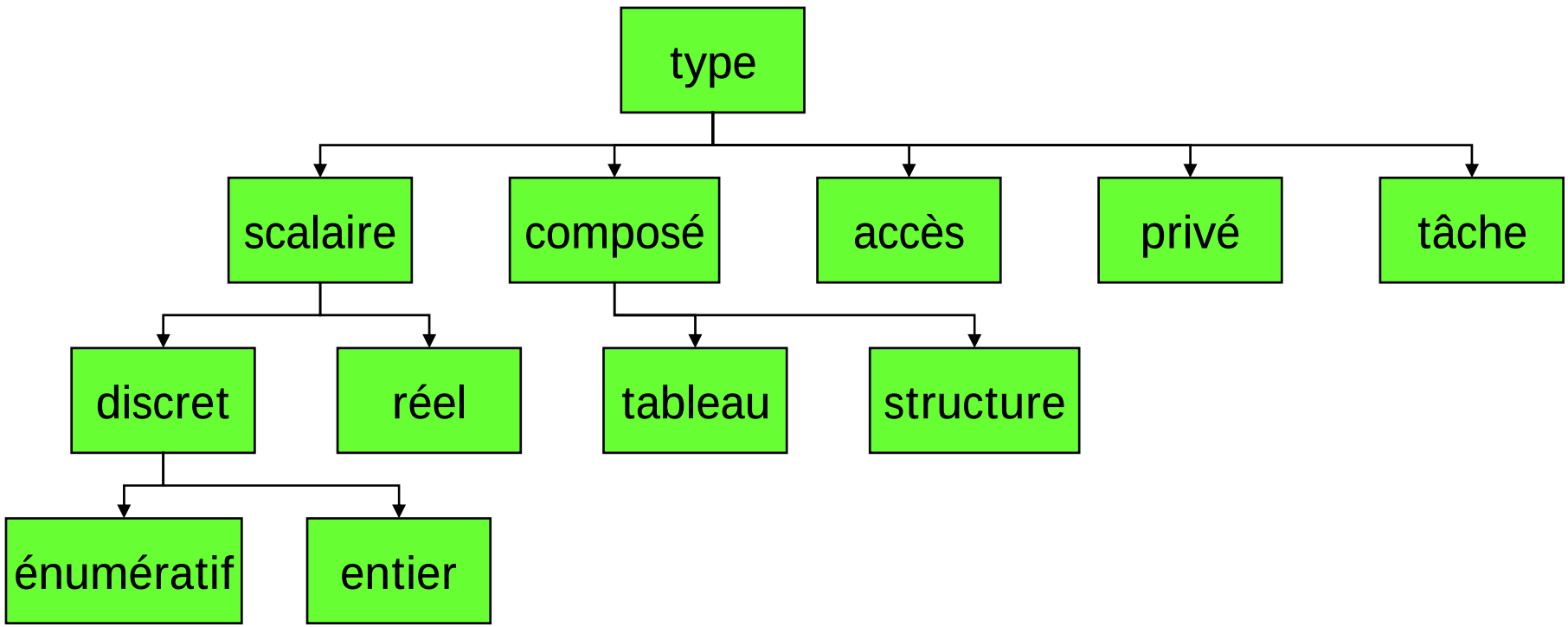
Package générique

- Instanciation d'une pile

```
                                                    -- pile booléenne

package PREMIERE_PILE is new PILE(50, Boolean) ;

                                                    -- pile entière

Package SECONDE_PILE is new PILE(100, Integer);
```

Type

- ADA typage fort
  - Vérification forte lors de la compilation
  - Possibilité de mettre des contraintes sur l'étendue des valeurs



Type

- Constante

```
nom_const : constant := expression ;
```

- Variable

```
nom_var : nom_type [:= expression] ;
```

- Type

```
type nom_type is type_definition ;
```

- Sous type

```
subtype nom_type is nom_type range min..max ;

                                        -- exemple
subtype numero_jour is Integer range 1..31 ;
```

Type énuméré

- Définition

```
type couleur is (rouge,orange,vert,bleu) ;
type fruit is (pomme,orange,poire) ;
type jour is (lundi,mardi,mercredi,jeudi,vendredi,
    samedi,dimanche) ;
```

- Orange ou orange ?

```
couleur'(orange)
fruit'(orange)
```

- Sous type

```
subtype jour_ouvrable is jour range lundi..vendredi ;
```

- Attribut

```
couleur'FIRST = rouge     couleur'LAST = bleu
couleur'SUCC(orange) = vert
couleur'PRED(orange) = rouge
```

Type énuméré point fixe

- Définition

```
type couleur is delta D range m..M ;
```

- Exemple

```
type euro is delta 0.01 range -1_000_000.00..1_000_000.00 ;
```

Type tableau

- Variable tableau

```
nom_tableau : array ( nom_type range m..M ) of nom_type ;
nom_tableau : array (m..M ) of nom_type ;
```

- Type

```
type nom_type_tableau is array ( nom_type range m..M ) of
                                                    nom_type ;
```

- Exemples

```
type vect3D is array ( 1..3 ) of Float ;
type mat3D is array ( 1..3, 1..3 ) of Float ;

type vente is array ( jour ) of euro ;
```

Type tableau

- Sans contrainte

```
type nom_type_tab is array ( nom_type range < > ) of nom_type ;
```

- Exemples

```
type vect is array ( Integer range < > ) of Float ;
 A :  vect( 1..3);
 B :  vect( -10..10);
subtype vect4D is vect(0..3) ;
```

- Attribut (A objet tableau ou type tableau contraint)

```
A'FIRST(n)  A'LAST(n)      -- limite min max du nième index de A
A'LENGTH(n)
A'RANGE(n)                 -- soustype A'FIRST(n) .. A'LAST(n)
```

Type structuré

```
type type_mois is (JAN,FEV,MAR,AVR,MAI,JUN,JUL,
                                AUT,SEP,OCT,NOV,DEC);
type date is
        record
                jour: Integer range 1..31 ;
                mois: type_mois ;
                annee : Integer;
        end record ;
D: date;
D.jour := 14 ; D.mois := JUL; D.annee := 1789 ;
D:date :=(14,JUL,1789) ;
E:date; D := E ;
E:=(annee => 1789, mois => JUL, jour => 14) ;
```

Type structuré avec discriminant

```
type poly (degre:Integer) is
      record
            coef : array (0..degre) of Float ;
      end record ;

R : poly(5) ; P : poly :=(3,(5.0,0.0,4.0,2.0)) ;

function normaliser(p:poly) return poly is
   taille : Integer := p.degre ;
   begin
     while taille>0 and p.coef(taille) = 0 loop
            taille := taille -1 ;
     end loop;
     return (taille,p.coef(0..taille));
   end normaliser ;
```

Type structuré avec partie variable

```
type genre is (masculin,feminin);

type personne(sexe:genre) is
        record
                naissance : date ;
                case sexe is
                        when masculin => barbu : Boolean ;
                        when feminin => enfants : Integer ;
                end case;
        end record ;

pierre : personne(masculin) ; pierre.barbu := true ;
barbara:personne :=(feminin,(21,MAI,1971),2) ;
```

Type accès

```
type cellule ;
type lien is access cellule ;
type cellule is
    record
        valeur : Integer ;
        suivant : lien ;
    end record ;

L , K : lien;                                   -- L:lien := null ;

L := new cellule ; L.valeur := 37; L.suivant := null ;
L := new cellule'(37, null);

K := new cellule; K.all := L.all;
```

Structure de contrôle

- Conditionnelle

```
if expression_boleenne then
        instructions
{elsif expression_boleenne then
        instructions}
[else
        instructions]
end if ;
```

- Cas

```
case expression  is
        when choix{| choix} => instructions
        { when choix{| choix} => instructions }
        [when others => instructions]
end case ;
```

Structure de contrôle

- Tant que

```
while expression_boleenne loop
        instructions
end loop ;
```

- Pour

```
for expression  in [reverse] intervalle loop
        instructions
end loop ;
```

Fonction et procédure

- Déclaration

```
procedure nom [(paramètres)] ;
function nom [(paramètres)] return type ;
```

- Corps

```
procedure nom[(paramètres)] is
        déclaration
begin
        instructions
end name ;

function nom [(paramètres)] return type is
        déclaration
begin
        instructions
end name ;
```

Fonction et procédure

- Paramètres

```
nom : (in | out| in out) type ;
```

- Paramètre valeur par défaut

```
procedure test ( a : in Integer ; option : in Integer := 0);

test(10,2) ; test(15) ;
```

- Appel avec paramètres nommés

```
test(option =>3, a => 5) ;
```

Autres

- Exception
  - Gestion des erreurs
  - Lever une erreur `raise nom_erreur ;`
  - Traiter une erreur `when nom_erreur{|nom_erreur} => instructions ;`
- Pragma
  - Ordre de compilation
- Objet et héritage

```
type Object is tagged
        record
                X: Float;Y: Float;
        end record;
type Circle is new Object with
        record
                Radius: Float;
        end record;
type Point is new Object with null record;
```

Tâche

- Tâche : entité d'exécution concurrente
- Définition
  - Construction explicite, pas d'appel au système
- Création
  - Implicite lors de la déclaration
  - Explicite lors d'une allocation
- Synchronisation et communication
  - Rendez vous (ADA 83)
  - Objet protégé (ADA 95)

Types et objets tâches

- Deux grandes possibilités
  - Définition d'un objet tâche
  - Définition d'un type tâche, puis d'une instance du type tâche
- Objet tâche
  - Définition implicite d'un type tâche anonyme
  - Définition explicite d'**une** instance
- Type tâche
  - Définition explicite d'un type tâche
  - Définition d'instances du type tâche
- Spécification d'un type tâche
  - Nom
  - Discriminant (éventuellement)
  - Partie visible, interface
  - Partie privée, corps

Types et objets tâches

- Objet tâche

```
task S is
        entry Service(r:Request)
end S;
task body S is
        c: Context;                     -- déclarations locales
begin
        Init(c,p);                      -- initialisation
        accept Service(r:Request)       -- tâche en attente
        do
                modify(c,r) ;           -- traitement appel
        end Service
        End(c);                         -- terminaison tâche
end S;
```

Types et objets tâches

- type tâche

```
task type Server (p:Parameter) is
        entry Service(r:Request)
end Server;
task body Server is
        c: Context;
begin
        Init(c,p);
        accept Service(r:Request) do modify(c,r) ;
        end Service ;
        End(c);
end Server;
S : server(my_parameters) ; -- création tâche
```

Types et objets tâches

- Discriminant sert de paramètre d'initialisation lors de la création de la tâche
- Les entrées servent à d'autres tâches à se synchroniser avec la tâche qui les définit
  - Rendez vous ADA
- Les information de l'interface sont visibles des autres tâches
  - Discriminant
  - Entrées
- Les informations du corps sont cachées

Types et objets tâches

- Création statique et dynamique

```
declare
        task type S (p:Parameter) is
                entry Service(r:Request)
        end S;
        task body S is
                ...
        end S;
        myS : S(my_parameters);   -- création et activation tâche
        mySs : array(1..3) of S ;   -- création et activation 3 tâches
        type pS is acces S ;
        mypS : pS;                -- définition tâche
begin
        mypS := new S(p);         -- activation tâche
```

Gestion du temps

- Deux types pour le temps
  - Time : temps absolue (01/01/05 16:28:3244)
    - Opérateur arithmétiques standards disponibles
  - Duration : durée en seconde
- Fonction Clock retourne l'horloge de type Time
- Ordre **delay**
  - Absolu
    - Attendre jusqu'à une certaine « heure »
  - Relatif
    - Attendre 10 secondes

Gestion du temps

- Tâche périodique

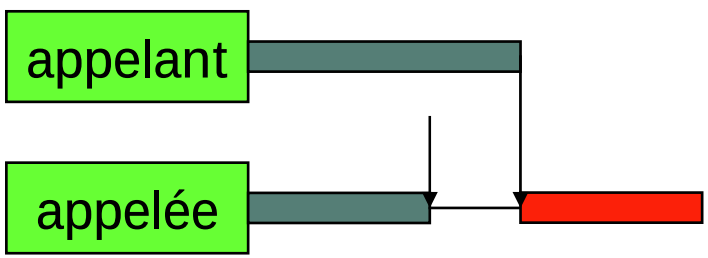
```
task body S is
                T1, T2 : Time ;
                P : constant Duration := 0.05 ; -- période
begin
        loop
                T1 := Clock;
                Work ;
                T2 := Clock ;
                if T2-T1 > P then raise time_error end if ;
                delay T1+P-T2 ;
                -- ou delay until T1+P ;
        end loop ;
end S;
```

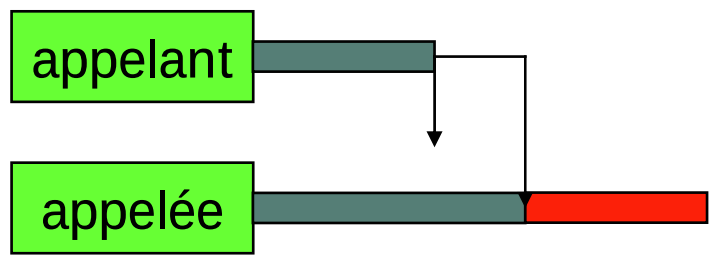
Synchronisation directe par rendez vous

- Tâches communiquent et se synchronisent grâce aux entrée
- Entrées sont publics
- Synchronisation
  - Appelant invoque l'entrée et se bloque en attendant que l'appelée accepte
  - L'appelé se bloque en attente d'un appel
  - Lorsque les deux tâches sont prêtes, l'appel se déroule chez l'appelé
- Appel chez l'appelant

  nom_tâche_appelée.entrée(paramètres)
- Entrée chez l'appelée

  accept entrée(paramètres) **do** instructions **end** entrée ;





Synchronisation sélective

```
Task body tampon is
    v : item ;
begin
    loop
        accept deposer(x: in item) do
                v:= x ;
        end deposer;
        accept prendre(x : out item) do
                x := v ;
        end prendre;
    end loop;
end tampon;
```

Synchronisation sélective

```
Task body tampon is
    v : item ;
begin
    accept deposer(x: in item) do
            v:=x;
    end deposer ;
    loop
            select
                    accept deposer(x: in item) do v:= x ; end deposer;
            or
                    accept prendre(x : out item) do x := v ; end prendre;
            end select;
    end loop;
end tampon;
```

Synchronisation sélective (clause de garde)

```
Task body tampon is
   m : constant := 8; v : array (1..m) of item ; i : integer range 0..m := 0;
begin
   loop
        select
                when i<m =>
                accept deposer(x: in item) do i := i+1; v(i):= x ;
                end deposer;
        or
                when i>0 =>
                accept prendre(x : out item) do x := v(i) ; i:=i-1;
                end prendre;
        end select;
   end loop;
end tampon;
```

Synchronisation sélective

```
loop
        select
                accept A do
                        instructions_A ;
                end A ;
        or
                accept B do
                        instructions_B;
                 end B;
        or
                terminate ; -- permet à la tâche de s'achever si aucune
                            --tâche ne peut l'appeler
        end select;
end loop;
```

Synchronisation sélective

```
loop
        select
                accept A do
                        instructions_A ;
                end A ;
        or
                accept B do
                        instructions_B;
                 end B;
        or
                delay D ;
                instructions_D ; -- si aucun appel est reçu avant D
        end select;
end loop;
```

Synchronisation sélective

```
loop
        select
                accept A do
                        instructions_A ;
                end A ;
        or
                accept B do
                        instructions_B;
                 end B;
        else      -- aucune tâche n'est bloquée sur un appel de A ou B
                instructions_D ;
        end select;
end loop;
```

Synchronisation sélective

```
loop
        select
                accept A do
                        instructions_A ;
                end A ;
        else -- Si aucune tâche  n'est bloquée sur A
                select
                        accept B do instructions_B; end B;
                or
                        accept C do instructions_C; end C;
                end select;
        end select;
end loop;
```

Synchronisation sélective

```
loop
        select
                tâche1.A ;          -- tentative d'appel de A de tâche1
        or
                delay D ;           -- Si pas de réponse avant D
                tâche2.B ;          -- alors appel B de tâche2
        end select;
end loop;
loop
        select
                tâche1.A ;          -- tentative d'appel de A de tâche1
        else                        -- Si pas disponible
                tâche2.B ;          -- alors appel B de tâche2
        end select;
end loop;
```

Objet protégé

- « Tâche » pour les données
  - Type anonyme ou type défini
- Mécanismes
  - Accès concurrents en lecture
  - D'exclusion mutuelle
  - Files d'attentes
- Procédure
  - Accès en exclusion mutuelle
- Fonction
  - Accès concurrents en lecture
- Entrée
  - Exclusion mutuelle
  - Clause de garde

Objet protégé

```
protected type SI(init : item) is
    function get return item ;
    procedure set(v:item);
private
    i : item := init ;
end SI

protected body SI is
    function get return Item is begin return I ; end get;
    procedure set(v:item) is
    begin i:=v; end set;
end SI;

My_SI : SI(5) ;
```

Objet protégé (clause de garde)

```
protected type tampon is
    entry deposer (x: in item);
    entry prendre (x : out item);
private
    m : constant := 8;
    v : array (1..m) of item ;
    i : integer range 0..m := 0;
end tampon ;
```

Objet protégé (clause de garde)

```
protected body tampon is
    entry deposer (x: in item) when i<m is
        begin
                i:= i+1;
                v(i) := x;
        end deposer;
    entry prendre (x : out item) when i>0 is
        begin
                x := v(i) ;
                i := i-1;
        end prendre;
end tampon ;
```

Objet protégé (appel)

```
My_SI.set(v)

select
    My_SI.get(v)
or
    delay 0.1;
    -- on fait autre chose
end select

select
    My_SI.get(v)
else
    -- on fait autre chose
end select
```

Avortement

- L'instruction **ABORT** permet de tuer une tâche

```
task type T ;
type pT is access T ;
myT1 : T ;
myT2 : pT : = new T ;
abort T1;
abort T2.all
```

- **ABORT** permet de tuer un calcul trop long

```
select
        delay 5.0; -- triggering alternative
                Put_Line("Calculation did not complete");
        then abort
                Invert_Giant_Matrix(M); -- abortable part
end select;
```