



Ada, langage de programmation pour le temps réel

Laurent Pautet
Jérôme Hugues

Laurent.Pautet@enst.fr

Version 1.5

(tutorial Ada95 – <http://www.enst.fr/~pautet/Ada95>)



Des langages concurrents aux langages temps réel

Un langage de programmation pour le temps réel doit faciliter la mise en œuvre de la concurrence

- Au travers de constructeurs du langage
 - Ada, Occam, ...
- Au travers de bibliothèques prédéfinies
 - Java, Modula, ...
- Au travers de bibliothèques du système
 - C ou C++ et Threads POSIX, ...

mais ce n'est pas suffisant



Programmation bas-niveau

- Maîtrise sur le flot de contrôle
 - Pas de *goto*
 - Boucles non-bornées
 - Restriction sur les appels récurifs
 - Pas de fonctions à effet de bord
- Maîtrise sur les données
 - Typage fort nécessaire pour la fiabilité du code
 - Préférence pour les accès directs aux données
 - Structures très dynamiques
- Restrictions dès que les contraintes temporelles ou le déterminisme ne sont pas toujours garanties

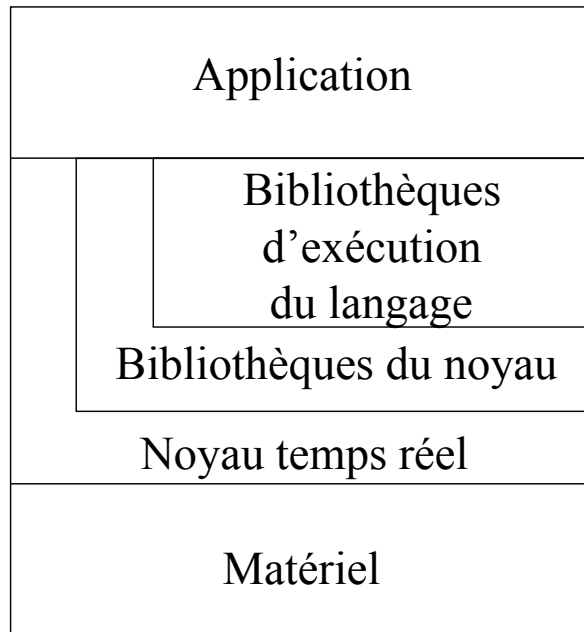


Programmation haut-niveau

- Principes de Génie Logiciel extrêmement stricts
 - Forte sémantique => Forte vérification => Forte fiabilité
- Analyse et preuve de l'architecture et des composants
 - Restrictions sur le pré-processeur (quel code vraiment compilé?)
- Réutilisation maximum de modules existants prouvés
- Décomposition et tests de modules simples
- Visibilité des données (privées, publiques, ...)
- Restriction sur l'orienté objet
 - Peu de polymorphisme (calcul du pire cas)
 - Peu de structure dynamique (gestion mémoire)
- Gestion de configuration et du déploiement
 - Dépendances vis à vis du matériel et du système



Mise en œuvre d'un langage de programmation temps réel



- Chaque couche, si elle existe, fournit un niveau d'abstraction
- Les constructions du langage sont étendues (*expansion*) par le compilateur pour utiliser celles de la biblio du langage ou celles de l'exécutif TR
- Exemples :
 - Tâche \Leftrightarrow Thread
 - Obj protégé \Leftrightarrow Mutex+CondVar
 - Delay \Leftrightarrow Timer



Avantages d'un langage de programmation temps réel

- Les bibliothèques fournissent une sémantique réduite et s'avèrent sources fréquentes d'erreurs
- Les constructions de langages fournissent plus de sémantiques et de richesses
- Le compilateur ne se trompent jamais dans l'expansion de ces constructions vers des bibliothèques
- Une forte sémantique permet de mener une analyse des sources ainsi que des vérifications formelles
- Un langage de programmation temps réel vient également avec des notions de typage fort, de visibilité, de précision sur les données (IEEE 754) ...



Validation d'un compilateur Ada Cœur, Annexes et Profiles

- Pour être qualifié de compilateur Ada, celui-ci doit passer un ensemble de tests qui vérifient exhaustivement le respect de la norme (et pas uniquement l'usage fait par les utilisateurs)
- Le compilateur peut valider
 - Le cœur du langage
 - Puis les annexes spécialisées dont celle de temps réel
 - Puis les profiles spécialisées comme Ravenscar
- Chaque spécialisation définit un sous-ensemble du langage dont les restrictions améliorent le déterminisme ou facilitent l'obtention de preuves



Spécifications des annexes et profiles

- Les pragmas sont des directives de compilation
 - ne changent pas la sémantique
 - mais en précisent ou restreignent l'usage général
- Les annexes et les profiles spécialisés pour le temps réel fournissent de nouveaux paquetages et définissent de nouveaux paquetages



Ada, annexes du langage

- Annexe des systèmes temps réel
 - Précision de l'horloge
 - Politiques d'ordonnancement, de files, de verrous
- Annexe de programmation système
 - Gestion des interruptions
 - Représentation des données
- Annexe de fiabilité et sécurité
 - Validation du code
- Annexe d'interfaçage avec les autres langages
 - Un système temps réel est rarement uni-langage



Définition des priorités

- Un environnement Ada doit fournir au moins 30 priorités dont une propre aux interruptions

```
subtype Any_Priority is Integer
    range Implementation-Defined;
subtype Priority is Any_Priority
    range Any_Priority'First .. Impl_Defined;
subtype Interrupt_Priority is Any_Priority
    range Priority'Last + 1 .. Any_Priority'Last;
Default_Priority : constant Priority :=
    (Priority'First + Priority'Last)/2;
```



Priorité statique des tâches

- Un pragma Priority ajouté à la spécification d'une tâche fixe statiquement la priorité initiale de la tâche

```
task Server is  
    entry Service;  
    pragma Priority (10);  
end Server;
```

```
task Critical_Server is  
    pragma Priority (Priority'Last);  
end Critical_Server;
```

```
task type Servers (My_Priority : System.Priority) is  
    pragma Priority (My_Priority);  
end Servers;
```



Priorité dynamique des tâches

- Le paquetage Ada.Dynamic_Priorities permet de modifier dynamiquement les priorités des tâches
- Attention : changer une priorité est une opération coûteuse pour tout exécutif

```
package Ada.Dynamic_Priorities is
  procedure Set_Priority
    (Priority : Any_Priority;
     T        : Task_Id := Current_Task);
  function Get_Priority
    (T : Task_Id := Current_Task)
    return Any_Priority;
private
  -- not specified by the language
end Ada.Dynamic_Priorities;
```



Héritage de priorité

Rendez-vous

- Un rendez-vous s'exécute à la plus haute des priorités courantes de l'appelant et de l'appelé. Chacun retrouve sa priorité précédente
- Une tâche fille effectue son activation à la priorité courante de la tâche mère puis poursuit son exécution à sa priorité initiale pour éviter une inversion de priorité

```
task Server is  
  entry Service;  
  pragma Priority (10);  
end Server;
```

```
task Critical_Server is  
  pragma Priority  
    (Priority'Last);  
end Critical_Server;
```

```
task body Critical_Server is  
begin  
  Server.Service;  
end Critical_Server;
```

```
task body Server is  
begin  
  accept Service do  
    -- s'exécute à Priority'Last  
  end Service;  
end Server;
```

le serveur récupère la priorité de l'appelant

13 le max des

Héritage de priorité

Objet protégé

- Un objet protégé garantit une exclusion mutuelle sur l'objet
- Le pragma Priority ajouté à la spécification d'un objet protégé permet d'élever immédiatement la priorité de la tâche appelante à celle de l'objet protégé
- La priorité joue un rôle de plafond de priorité comme dans la politique Highest Priority Protocol, variante de PCP
- Si la priorité de la tâche appelante est supérieure à celle du plafond de priorité, l'exception Program_Error est levée puisque le plafond a été mal calculé
- Par défaut, le plafond de priorité est la plus haute des priorités de tâche, Priority'Last



Plafond de priorité

- Classiquement, une tâche T exécute au plafond de priorité une opération P sur l'objet protégé mais également celles des autres tâches qui se trouvent débloquées comme conséquence de P
- Si T_2 (priorité 16) est bloquée sur `Altitude.Read` et que T_1 (priorité 18) exécute `Altitude.Write`, T_1 se charge d'exécuter `Altitude.Read` à la priorité 20 pour T_2 (moins de changement de contexte)

```
protected Altitude is
  entry Read
    (V : out Meter);
  procedure Write
    (V : Meter);
  pragma Priority (20);
private
  Value : Meter;
  Ready : Boolean
    := False;
end Altitude;
```

indique la
priorité plafond

```
protected body Altitude is
  entry Read (V : out Meter)
    when Ready is
  begin
    V := Value; Ready := False;
  end Read;
  procedure Write (V : Meter) is
  begin
    Value := V; Ready := False;
  end Write;
end Altitude;
```



Politique de gestion de l'ordonnancement

pragma Task_Dispatching_Policy (FIFO_Within_Priorities);

- La politique par défaut de gestion de l'ordonnancement est non-spécifiée (par exemple Time_Sharing)
- Le pragma Task_Dispatching_Policy force une politique de gestion de l'ordonnancement particulière
- Un environnement pour le temps réel doit proposer FIFO_Within_Priorities et éventuellement d'autres politiques
- FIFO_Within_Priorities spécifie les règles suivantes
 - On exécute la première tâche de la file plus haute priorité
 - Si une tâche devient prête, elle se place en fin de file
 - Si une tâche est interrompue, elle reste en début de file



Politique de gestion des files d'attente

pragma Queuing_Policy (FIFO_Queueing);

- La politique par défaut de gestion des files d'attente est FIFO_Queueing
- Le pragma Queuing_Policy force une politique de gestion des files d'attente particulière
- Un environnement pour le temps réel doit proposer FIFO_Queueing et surtout Priority_Queueing et éventuellement d'autres politiques
- Priority_Queueing spécifie les règles suivantes
 - Lorsqu'une entrée devient passante, la tâche de plus forte priorité est sélectionnée
 - En cas d'égalité de priorité, la première tâche dans l'ordre alphabétique est sélectionnée



Politique de gestion des verrous

pragma Locking_Policy (Ceiling_Priority);

- La politique par défaut de gestion des verrous est non-spécifiée
- Le pragma Locking_Policy force une politique de gestion des verrous particulière
- Un environnement pour le temps réel doit proposer Ceiling_Priority et éventuellement d'autres politiques
- Ceiling_Priority spécifie les règles suivantes
 - Une tâche hérite du plafond de priorité (Priority_Last par défaut)
 - Si la tâche a une priorité supérieure, Program_Error est levée



Horloge temps réel

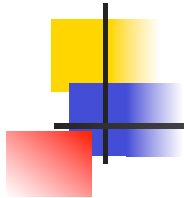
- Il faut disposer d'une horloge croissante monotone
 - Ada.Calendar (horloge normal)
 - Ada.Real_Time (horloge temps réel)
- Il faut disposer d'une horloge précise
 - Soit deux tâches l'une d'une période de 10ms et l'autre de 40ms
 - Elles peuvent ne pas être harmoniques (compensation)

```
Time_Unit = 2-31  
Millisecond (40) = 101000111101011100001010010  
                  = 0.04 + (21/25) * 2-31  
Millisecond (10) = 1010001111010111000010100  
                  = 0.01 - (03/25) * 2-31  
Millisecond (10) * 4 - Millisecond (40) = 2-30
```



Attente et échéance

- Il faut pouvoir suspendre une tâche pendant un temps d'attente ou jusqu'à une échéance
 - delay Temps_Relatif (attente normale)
 - delay until Temps_Absolu (attente temps réel)
- Le temps d'attente est un temps minimum
 - delay T : la tâche est réveillée après que ce soit écoulé au moins la durée T (résolution du timer)
 - delay until D : la tâche est réveillée à une date ultérieure (à la date D)
- delay until D # delay (D – Clock)
 - Car la soustraction peut être préemptée



Tâche périodique

```
task Normal_Sensor is
    pragma Priority (10);
end Normal_Sensor ;
task body Normal_Sensor
is
    use Ada.Real_Time;
    P : constant Time
        := Millisecond (40);
    D : Time := Clock + P;
begin
    loop
        -- Lit les capteurs
        delay D - Clock;
        D := D + P;
    end loop;
end Normal_Sensor ;
```

```
task Real_Time_Sensor is
    pragma Priority (10);
end Real_Time_Sensor;
task body Real_Time_Sensor
is
    use Ada.Real_Time;
    P : constant Time
        := Millisecond (40);
    D : Time := Clock + P;
begin
    loop
        -- Lit les capteurs
        delay until D;
        D := D + P;
    end loop;
end Real_Time_Sensor;
```



Chiens de garde

- Une tâche ne souhaite pas bloquer sur une entrée plus d'un certain temps
- Une tâche ne souhaite pas bloquer sur une entrée si elle ne l'obtient pas immédiatement

```
select  
    A_Task.An_Entry;  
or  
    delay Timeout;  
end select;
```

```
select  
    A_Task.An_Entry;  
else  
    Do_Something;  
end select;
```



Gestion des interruptions

Annexe de programmation système

```
protected Message_Driver is  
  entry Get  
    (M : out Message);  
private  
  Current : Message := None;  
  procedure Handle;  
  pragma Attach_Handler  
    (Handle, Device_IT_Id);  
  pragma Interrupt_Priority;  
end Message_Driver ;
```

```
protected body Message_Driver is  
  entry Get (M : out Message)  
    when Current /= None is  
    begin  
      M := Current;  
      Current := None;  
    end Get;  
  procedure Handle is  
    begin  
      if Current /= None then  
        Report_Overflow;  
      end if;  
      Current := Read_Device;  
    end Handle;  
end Message_Driver ;
```

Contrôle de l'ordonnancement

Annexe de programmation système

- Par des interfaces comme POSIX, on peut accéder directement à l'ordonnancement et l'effectuer soi même
- En Ada, les paquetages Asynchronous_Task_Control et Task_Identification permettent d'avoir un contrôle fin.

```
package Task_Identification is  
  type Task_ID is private;  
  function Current_Task  
    return Task_ID;  
  procedure Abort_Task  
    (T : in out Task_ID);  
  ...  
end Task_Identification;
```

```
package Asynchronous_Task_Control is  
  procedure Hold (T : Task_ID);  
  procedure Continue (T : Task_ID);  
  function Is_Held (T : Task_ID)  
    return Boolean;  
end Asynchronous_Task_Control;
```


Représentation des données

Annexe de programmation système

- Il faut s'interfacer avec le matériel
- Même le langage C a ses limites (*bit fields*)

```
type Status is
  (Off, On, Busy);
for Status use
  (Off => 0,
   On  => 1,
   Busy => 3);
for Status'Size use 2;
type Data is range 0 .. 127;
for Data'Size use 7;
type Register is record
  Send : Data;
  Recv : Data;
  Stat : Status;
end record;
```

```
for Register use record
  Send at 0 range 0 .. 6;
  Recv at 0 range 7 .. 13;
  Stat at 0 range 14 .. 15;
end record;

Device_Register : Register;
for Device_Register
  use at 8#100#;
```



Accès physique aux données

Annexe de programmation système

- **pragma Volatile**
 - L'objet est directement accédé en mémoire
 - L'usage de cache et de registre est prohibé
 - La dernière valeur écrite correspond à la valeur lue
- **pragma Atomic**
 - L'objet est toujours accédé de manière atomique
 - Les opérations sur de petits objets ne sont pas toujours atomiques

```
package Shared_Buffer is
  Buffer_Array : array (0 .. 1023) of Character;
  pragma Volatile_Components (Buffer_Array);
  Next_In, Next_Out : Integer := 0;
  pragma Atomic (Next_In, Next_Out);
end Shared_Buffer;
```



Validation du logiciel

Annexe de fiabilité et sécurité

- `pragma Reviewable;`
 - Ce pragma demande au compilateur d'annoter son code et de fournir des informations telles que l'on puisse déterminer le temps d'exécution, l'utilisation de la mémoire, etc.
- `pragma Inspection_Point (V);`
 - Ce pragma demande au compilateur de rendre la variable V accessible à l'endroit où se trouve le pragma. A priori, le compilateur ne pourra pas garder la variable dans des registres et devra également éviter d'optimiser ces modifications sur la variable à cet endroit précis.

Efficacité du logiciel

Annexe de fiabilité et sécurité

- pragma Suppress (...);
 - Ce pragma demande au compilateur de supprimer certaines voire toutes les vérifications qu'il peut effectuer implicitement sur les contraintes de type, sur les contraintes de bornes de tableau, etc.
- De nombreux pragmas permettent d'inhiber les vérifications que le compilateur génère normalement pour l'utilisateur. Toutefois, si le logiciel a été sévèrement testé, certaines vérifications deviennent inutiles et peuvent ainsi être élégamment supprimées.

Vers les autres langages

Annexe d'interfaçage

- Un système est multi-langages
 - S'interfacer facilement avec d'autres langages
 - S'assurer d'une certaine interopérabilité
 - Contrôler
 - Le passage des paramètres,
 - La représentation des données
 - Le caractère réentrant des fonctions
- Ada offre des interfaces vers les types prédéfinis de divers langages C, COBOL, FORTRAN, etc.

```
function Gethostbyname (Name : in C.Address)
    return Host_Entry_Address;
pragma Import (C, Gethosbyname, "gethostbyname");
```



Restrictions sur les constructions du langage

- Dès lors, aucun développeur n'utilise des points interdits du langage dans le système temps réel
 - pragma Restrictions (No Task Allocator);
 - -- No allocators for task types
 - pragma Restrictions (No Task Hierarchy);
 - -- All tasks depend directly from env. task ...
 - pragma Restrictions (No Allocator);
 - -- There are no occurrences of an allocator
 - pragma Restrictions (No Recursion);
 - -- As part of the execution of a subprogram,
 - -- the same subprogram is not invoked
 - pragma Restrictions (No Dispatch);
 - ...



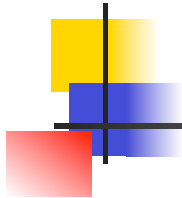
Profil Ravenscar

- Sous-ensemble des constructions concurrentes:
 - Ada (Ada 0Y) et Java (RTJS)
- Conçu pour permettre une analyse complète d'une application temps réel
 - Déterminisme de l'exécution
 - Ordonnancement
 - Empreinte mémoire bornée
- Possibilité de définir des exécutifs légers
- Possibilité de vérification statique et certification
 - Flux de données, preuves formelles



Objectifs généraux du profil Ravenscar

- Une règle: supprimer tout comportement dynamique non déterministe
- Ordonnanceur RMS: FIFO_Within_Priorities
 - Techniques d'analyse
- Politique de verrous: Ceiling_Locking
 - Calcul du WCET lors d'interblocage
- Pas d'allocation implicite sur la pile
- Pas de délais relatifs



Modèle de tâches réduit

- Pré-allocation statique de tâches
 - Pas d'allocation dynamique de tâches
- Pas de hiérarchie de tâches
- Pas d'avortement
- Pas de finalisation
- Pas de finalisation
 - Analyse facilitée du cycle de vie d'une tâche



Boîte aux lettres avec priorité 1/2

Solution avec queue

```
type Prioritized_Messages is
    array (Any_Priority) of Messages;
protected Prioritized_Mailbox is
    procedure Put (M : Message; P : Any_Priority);
    entry Get (M : out Message; P : Any_Priority);
private
    Mailbox : Prioritized_Messages;
    Updating : Boolean := False;
    entry Wait (M : out Message; P : Any_Priority);
end Prioritized_Mailbox;

protected Prioritized_Mailbox is
    procedure Put (M : Message; P : Any_Priority) is
    begin
        Append (Mailbox (P), M);
        if Wait'Count /= 0 then
            Updating := True;
        end if;
    end Put;
```



Boîte aux lettres avec priorité 2/2

Problème de déterminisme

```
entry Get (M : out Message; P : Any_Priority)
  when not Updating is
  begin
    if Length (Mailbox (P)) = 0 then
      requeue Wait;
    end if;
    Extract (Mailbox (P), M);
  end Get;

entry Wait (M : out Message; P : Any_Priority)
  when Updating is
  begin
    if Wait'Count = 0 then
      Updating := False;
    end if;
    requeue Get;
  end Get;
end Prioritized_Mailbox;
```



Boîte aux lettres avec priorité 1/2

Solution avec familles d'entrée

```
protected Prioritized_Mailbox is
  procedure Put (M : Message; P : Any_Priority);
  entry Get (M : out Message; P : Any_Priority);
private
  Mailbox : Prioritized_Messages;
  entry Wait (Any_Priority)
    (M : out Message; P : Any_Priority);
end Prioritized_Mailbox;

protected Prioritized_Mailbox is
  procedure Put (M : Message; P : Any_Priority) is
  begin
    Append (Mailbox (P), M);
  end Put;
```



Boîte aux lettres avec priorité 2/2

```
entry Get (M : out Message; P : Any_Priority)
  when True is
begin
  if Length (Mailbox (P)) = 0 then
    requeue Wait (P);
  end if;
  Extract (Mailbox (P), M);
end Get;

entry Wait (for A in Any_Priority)
  (M : out Message; P : Any_Priority)
  when Length (Mailbox (A)) > 0 is
begin
  Extract (Mailbox (P), M);
end Get;
end Prioritized_Mailbox;
```