dit
UPM

Gestión del tiempo en Ada

Juan Antonio de la Puente
DIT/UPM

El tiempo en Ada

- La gestión del tiempo en Ada está integrada en el lenguaje y en el modelo de tareas
  - relojes
  - relojes de tiempo de ejecución
  - retardos
  - límites de tiempo

Relojes

En Ada hay dos paquetes predefinidos que proporcionan funciones de reloj:

- Ada.Calendar
  - Define un tipo abstracto Time
  - La función Clock da un valor de tiempo para uso externo
  - Los intervalos de tiempo se representan con el tipo predefinido Duration
- Ada.Real_Time
  - Define un tipo abstracto Time
  - La función Clock da un valor monótono, sin saltos
  - Los intervalos de tiempo se representan con el tipo abstracto Time_Span

También hay relojes de tiempo de ejecución

Intervalos de tiempo

- El tipo Duration representa intervalos de tiempo en segundos

- Es un tipo de coma fija:

  type Duration **is delta** ... **range** ...;

  - Su resolución, Duration'Small, no debe ser mayor que 20ms (se recomienda que no sobrepase los 100μs)
  - El intervalo de valores debe comprender ±1 día (-86_400.0 .. +86_400.0)

Ada.Calendar (1)

```
package Ada.Calendar is

   type Time is private;

   subtype Year_Number     is Integer  range 1901..2099;
   subtype Month_Number    is Integer  range 1..12;
   subtype Day_Number                  is Integer  range 1..31;
   subtype Day_Duration    is Duration range 0.0..86_400.0;

   function Clock return Time;

   function Year   (Date : Time) return Year_Number;
   function Month  (Date : Time) return Month_Number;
   function Day    (Date : Time) return Day_Number;
   function Seconds(Date : Time) return Day_Duration;

   procedure Split(Date    : in  Time;
                   Year    : out Year_Number;
                   Month   : out Month_Number;
                   Day     : out Day_Number;
                   Seconds : out Day_Duration);
```

Ada.Calendar (2)

```
   function Time_Of
  (Year    : Year_Number;
   Month   : Month_Number;
   Day     : Day_Number;
   Seconds : Day_Duration := 0.0)
  return    Time;

   function "+" (Left : Time;     Right : Duration) return Time;
  function "+" (Left : Duration; Right : Time)     return Time;
  function "-" (Left : Time;     Right : Duration) return Time;
  function "-" (Left : Time;     Right : Time)     return Duration;

  function "<"  (Left, Right : Time) return Boolean;
  function "<=" (Left, Right : Time) return Boolean;
  function ">"  (Left, Right : Time) return Boolean;
  function ">=" (Left, Right : Time) return Boolean;

  Time_Error : exception;
end Ada.Calendar;
```

Comentarios a Ada.Calendar

- Los valores del tipo Time combinan la fecha y la hora
- La hora se da en segundos desde medianoche
  - cuando hay un segundo intercalar se llega a 86_400.0
- El reloj se supone sincronizado con una referencia externa (UTC o TO)
  - los detalles se dejan para el entorno (SO)
- Los paquetes Ada.Calendar.Time_Zones Ada.Calendar.Arithmetic y Ada.Calendar.Formatting proporcionan funciones adicionales

Ada.Real_Time (1)

```
package Ada.Real_Time is

   type Time is private;
   Time_First : constant Time;
   Time_Last  : constant Time;
   Time_Unit  : constant := -- real number;

   type Time_Span is private;
   Time_Span_First :  constant Time_Span;
   Time_Span_Last  :  constant Time_Span;
   Time_Span_Zero  :  constant Time_Span;
   Time_Span_Unit  :  constant Time_Span;

   Tick : constant Time_Span;

   function Clock return Time;

   function "+"  (Left : Time;      Right : Time_Span) return Time;
   function "+"  (Left : Time_Span; Right : Time)      return Time;
   function "-"  (Left : Time;      Right : Time_Span) return Time;
     function "-"  (Left : Time;      Right : Time) return Time_Span;

   function "<"  (Left, Right : Time) return Boolean;
   function "<=" (Left, Right : Time) return Boolean;
   function ">"  (Left, Right : Time) return Boolean;
   function ">=" (Left, Right : Time) return Boolean;
```

Ada.Real_Time (2)

```
function "+" (Left, Right : Time_Span)          return Time_Span;
function "-" (Left, Right : Time_Span)          return Time_Span;
function "-" (Right : Time_Span)                return Time_Span;
function "*" (Left : Time_Span; Right : Integer)  return Time_Span;
function "*" (Left : Integer;   Right : Time_Span)return Time_Span;
function "/" (Left, Right : Time_Span)          return Integer;
function "/" (Left : Time_Span; Right : Integer)  return Time_Span;
function "abs" (Right : Time_Span)              return Time_Span;

function "<"  (Left, Right : Time_Span) return Boolean;
function "<=" (Left, Right : Time_Span) return Boolean;
function ">"  (Left, Right : Time_Span) return Boolean;
function ">=" (Left, Right : Time_Span) return Boolean;

function To_Duration  (TS : Time_Span)  return Duration;
function To_Time_Span (D : Duration)    return Time_Span;

function Nanoseconds  (NS : integer)    return Time_Span;
function Microseconds (US : integer)    return Time_Span;
function Milliseconds (MS : integer)    return Time_Span;
```

Ada.Real_Time (3)

```
   type Seconds_Count is new Integer range ...;

  procedure Split (T : Time;
                   SC : out Seconds_Count;
                   TS : out Time_Span);
  function Time_Of (SC : Seconds_Count; TS : Time_Span)
      return Time;

end Ada.Real_Time;
```

Comentarios sobre Ada.Real_Time (1)

- El tipo Time representa valores de tiempo absolutos.
  - Un valor *T* de tipo Time representa un intervalo de duración
    $[E + T \cdot \text{Time\_Unit}, E + (T+1) \cdot \text{Time\_Unit}]$
    - » Time_Unit no debe ser mayor de 20ms..
    - » El valor de *E* no está definido
  - El intervalo de valores del tipo Time debe alcanzar al menos 50 años desde el arranque del sistema.
- El tipo Time_Span representa intervalos de tiempo.
  - Un valor *S* de tipo Time_Span representa un intervalo de duración igual a $S \cdot \text{Time\_Span\_Unit}$.
    - » Time_Span_Unit =Time_Unit
    - » Duration'Small debe ser un múltiplo entero de Time_Span_Unit.
  - El intervalo de valores del tipo Time_Span debe abarcar por lo menos -3600..+3600 s .

Comentarios sobre Ada.Real_Time (2)

- La función Clock proporciona el tiempo absoluto transcurrido desde la época.

- Tick es el valor medio del intervalo durante el cual el valor de Clock permanece constante. No debe ser mayor de 1ms
  - Se recomienda que el valor de Tick sea igual al de Time_Span_Unit, o un múltiplo exacto de éste.

- El valor de Clock no debe disminuir en ningún momento (es decir, el reloj es monótono no decreciente).

Ejemplo

```
declare
        use Ada.Real_Time;
        Start, Finish : Time;
        Frame         : Time_Span := Milliseconds(10);
begin
   Start := Clock;
   -- instrucciones
   Finish := Clock;
   if Finish - Start > Frame then
      raise Time_Error;
      -- excepción definida por el programador
   end if;
end;
```

Retardo relativo

- La instrucción

  delay *expresión;*

  suspende la ejecución de la tarea que la invoca durante el intervalo de tiempo que indica el valor de la *expresión*
  - es de tipo Duration (y por tanto se mide en segundos)

- Una instrucción delay con argumento cero o negativo no produce ningún retardo

Retardo absoluto

- La instrucción

  delay until *expresión*;

  suspende la ejecución de la tarea que la invoca hasta que el valor del reloj sea igual al especificado por la *expresión*

- La expresión es de uno de estos tipos:
  - Ada.Calendar.Time
  - Ada.Real_Time.Time
- Se usa el reloj correspondiente al tipo Time utilizado
- Si se especifica un tiempo anterior al valor actual del reloj, no se produce ningún retardo

Ejemplo: tarea periódica

```
use Ada.Real_Time;
task body Periodic is
  Period    : constant Time_Span := ...;
  Next_Time :          Time      := ...;
begin
  -- iniciación
  loop
    delay until Next_Time;
    -- acción periódica
    Next_Time := Next_Time + Period;
   end loop;
end Periodic;
```

Limitación del tiempo de espera

- A menudo conviene limitar el tiempo durante el cual se espera que ocurra un suceso

- Ejemplos:
  - Acceso a una sección crítica:
    La espera está limitada por la duración de la sección crítica
  - Sincronización condicional
    - » llamada a una entrada protegida con barreras
  - Cita entre dos tareas
  - Ejecución de una acción

Ejemplo

```
task Controller is
   entry Call (T : Temperature);
end Controller;


task body Controller is
   -- declaraciones
begin
   loop
      accept Call (T : Temperature) do
         New_Temp := T;
      end Call;
      -- otras acciones
   end loop;
end Controller;
```

Aceptación temporizada

- Se puede especificar una acción alternativa en caso de que la llamada no se reciba dentro de un cierto intervalo mediante una **aceptación temporizada:**

```
select
  accept Call (T : Temperature) do
    New_Temp := T;
  end Call;
or
  delay 10.0;
  -- acción alternativa
end select;
```

El retardo puede ser también absoluto

Llamada temporizada

- Se puede limitar el tiempo que tarda en aceptarse la llamada mediante una **llamada temporizada**

```
loop
    -- leer el nuevo valor de T
    select
      Controller.Call(T);
    or
      delay 0.5;
      -- acción alternativa
    end select;
end loop;
```

Aquí también puede usarse un retardo absoluto

También se puede usar con llamadas a entradas protegidas

Acciones temporizadas

- Se puede usar una **transferencia asíncrona de control** (ATC) para limitar el tiempo de ejecución de una acción:

```
select
    delay 0.1;
then abort
    -- acción
end select;
```

Es útil para detectar y recuperar fallos

Aplicación al cómputo impreciso

- Se trata de ejecutar rápidamente un parte obligatoria de un cálculo, y de iterar sobre una parte opcional que mejora el resultado mientras haya tiempo

```
begin
  -- parte obligatoria
  select
    delay until Completion_Time;
  then abort
    loop
      -- mejorar el resultado
    end loop;
 end select;
end;
```

Relojes de tiempo de ejecución (1)

```
with Ada.Task_Identification;
with Ada.Real_Time; use Ada.Real_Time;
package Ada.Execution_Time is

   type CPU_Time is private;
   CPU_Time_First : constant CPU_Time;
   CPU_Time_Last  : constant CPU_Time;
   CPU_Time_Unit  : constant := implementation-defined-real-number;

   function Clock (T : Ada.Task_Identification.Task_Id
                     := Ada.Task_Identification.Current_Task)
     return CPU_Time;

   function "+" (Left : CPU_Time; Right : Time_Span) return CPU_Time;
   function "+" (Left : Time_Span; Right : CPU_Time) return CPU_Time;
   function "-" (Left : CPU_Time; Right : Time_Span) return CPU_Time;
   function "-" (Left : CPU_Time; Right : CPU_Time)
       return Time_Span;
```

Relojes de tiempo de ejecución (2)

```
   function "<"  (Left, Right : CPU_Time) return Boolean;
   function "<=" (Left, Right : CPU_Time) return Boolean;
   function ">"  (Left, Right : CPU_Time) return Boolean;
   function ">=" (Left, Right : CPU_Time) return Boolean;

   procedure Split (T  : in CPU_Time;
                    SC : out Seconds_Count;
                    TS : out Time_Span);

   function Time_Of (SC : Seconds_Count;
                     TS : Time_Span := Time_Span_Zero)
      return CPU_Time;

private
   ... -- not specified by the language
end Ada.Execution_Time;
```

Temporizadores de tiempo de ejecución (1)

```
with System;
package Ada.Execution_Time.Timers is

   type Timer (T : access Ada.Task_Identification.Task_Id) is
         tagged limited private;

   type Timer_Handler is
         access protected procedure (TM : in out Timer);

   Min_Handler_Ceiling : constant System.Any_Priority
    := implementation-defined;

   procedure Set_Handler (TM      : in out Timer;
                          In_Time : in Time_Span;
                          Handler : in Timer_Handler);

   procedure Set_Handler (TM      : in out Timer;
                          At_Time : in CPU_Time;
                          Handler : in Timer_Handler);
```

Temporizadores de tiempo de ejecución (2)

```
   function Current_Handler (TM : Timer) return Timer_Handler;

   procedure Cancel_Handler (TM        : in out Timer;
                             Cancelled : in out Boolean);

   function Time_Remaining (TM : Timer) return Time_Span;

   Timer_Resource_Error : exception;

private
      ... --  not specified by the language
end Ada.Execution_Time.Timers;
```

Ejemplo (1)

```
…
Periodic_Id : aliased Ada.Task_Identification.Task_Id := Periodic'Identity;
WCET_Timer : Ada.Execution_Time.Timers.Timer (Periodic_Id'Access);
…
task body Periodic is
   Next_Start : Ada.Real_Time.Time := Ada.Real_Time.Clock;
   WCET   : constant Time_Span := Milliseconds(1);
   Period : constant Time_Span := Milliseconds(100);;
   OK     : Boolean;
begin
   loop
      Set_Handler (WCET_Timer, WCET, Supervisor.Overrun'Access);
      -- actividad periódica
      Cancel_Handler (WCET_Timer, OK);
      Next_Start := Next_Start + Period;
      delay until Next_Start;
   end loop;
end Periodic;
```

Ejemplo (2)

```
…
protected Supervisor is
  procedure Overrun;
private
  …
end Supervisor;

protected body Supervisor is

  procedure Overrun is
  begin
    …
  end Overrun;

end Supervisor;
```

Temporizadores para grupos de tareas

- El paquete Ada.Execution_Time.Group_Budgets permite definir grupos de tareas con un presupuesto de tiempo de ejecución global

- Se pueden fijar límites de tiempo para todo el grupo, y detectar si se sobrepasan con temporizadores

Gestión del tiempo en Ravenscar

- El perfil de Ravenscar prohíbe el uso de varios mecanismos de tiempo
  - Ada.Calendar
  - delay relativo
  - temporizadores de tiempo de ejecución
  - grupos de tareas
- Tampoco se puede usar select
  - ni, por tanto, llamadas temporizadas ni ATC
- Se pueden construir sistemas de tiempo real con lo que queda
  - Ada.Real_Time y delay until
  - Ada.Execution_Time

Resumen

- La gestión del tiempo está integrada con el lenguaje Ada
- El perfil de Ravenscar restringe los mecanismos que se pueden usar
  - el objetivo sigue siendo asegurar que los sistemas construidos con este perfil tienen un comportamiento temporal previsible
- Algunos mecanismos de tiempo son nuevos en Ada 2005
  - Ada.Execution_Time y sus hijos