Tema 3

Gestión del Tiempo en STR

Objetivos

1. Comprender los requerimientos temporales en el diseño e implementación de un STR
2. Manejar el tiempo en un STR
3. Estudiar algunas funcionalidades de Ada para gestionar el tiempo

Índice

Requerimientos temporales en STR

1. Interacción con el tiempo
   - medir el paso del tiempo en relojes
   - retrasar la ejecución de las tareas
   - programación de esperas limitadas (*timeouts*)
   - ejecutar acciones en determinados instantes
2. Representación de requerimientos temporales
   - periodos de activación
   - plazos de ejecución (***deadlines***)
3. Satisfacción de requerimientos temporales
   - ***Scheduling***

Facilidades de control de tiempo real

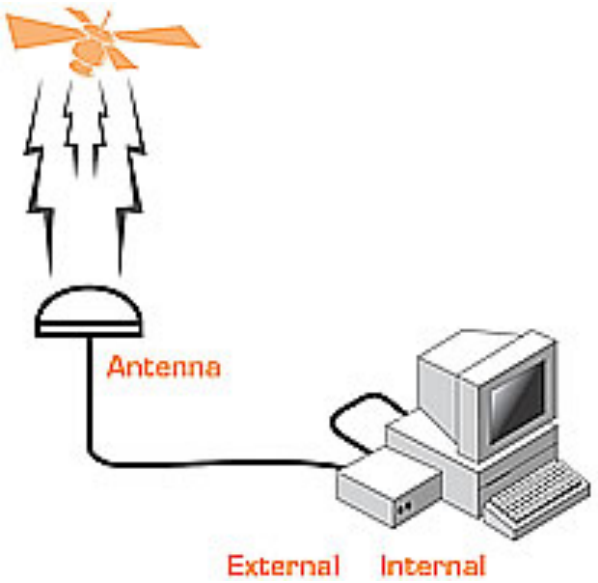
- Especificar los tiempos en los que las acciones tienen que ejecutarse
- Especificar los tiempos en los que las acciones tienen que completarse
- Responder a situaciones en las que no se pueden atender todos los plazos
- Responder a situaciones en las que los requerimientos temporales cambian dinámicamente

Índice

1. Requerimientos de tiempo en un STR
2. **Acceso a relojes**
3. Retardos
4. Temporizadores
5. Análisis de requerimientos temporales

Medir el paso del tiempo

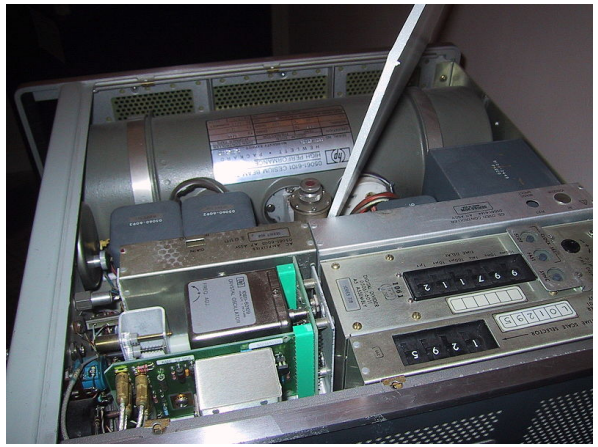
- Mediante acceso directo al marco temporal del entorno



- Mediante el uso de un reloj hardware interno que proporcione una aproximación adecuada del paso del tiempo en el entorno

Propiedades de un reloj hardware

- **Exacto**
  - Diferencia con TAI o UTC acotada por un valor ε
- **Preciso**
  - Drift limitado
- **Monotónico**
  - Valores crecientes
- **Estable**
  - Sin variaciones grandes en intervalos de tiempo



http://es.wikipedia.org/wiki/Reloj_atómico

Acceso a relojes en Ada

Paquetes	Ada.**Calendar**	Ada.**Real_Time**
Tipo de datos para **instantes de tiempo absoluto**	tipo **Time**	tipo **Time**
Tipo de datos para **intervalos de tiempo**	tipo **Duration**	tipo **Time_Span**
Función **Clock**	Devuelve un valor que combina la **fecha y la hora** actual	Devuelve un valor **monótono creciente** que representa el tiempo transcurrido desde un instante inicial prefijado

Paquete Ada.Calendar

```
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..86400.0;

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

Paquete Ada.Calendar

```
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;
   -- Time_Error may be raised by Time_Of,
   -- Split, Year, "+" and "-"

private
   --implementation-dependent
end Ada.Calendar;
```

Tipos definidos en Ada.Calendar

- El tipo **Time** proporciona instantes de tiempo absolutos
- El tipo **Duration** proporciona intervalos de tiempo relativos
  - Es un real de punto fijo
  - Rango mínimo: -86400.0 .. +86400.0
  - Precisión: su granularidad no debe ser mayor de 20 milisegundos
  - Los segundos se describen en términos del subtipo **Day_Duration**, que a su vez se define a través de Duration
    - La hora se da en segundos desde la medianoche

Ejemplo de uso de Ada.Calendar

```
declare
   with Ada.Calendar; use Ada.Calendar;
   Inicio, Fin : Time;
   Duracion : Duration;
begin
   Inicio := Clock;
   ...   -- secuencia de instrucciones
   Fin := Clock;
   Duracion := Fin - Inicio;
end;
```

Paquete Ada.Real_Time

```
package Ada.Real_Time is
   type Time is private;
   Time_First: constant Time;
   Time_Last: constant Time;
   Time_Unit: constant := implementation_defined_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;
   -- similarly for "-", "<",etc.
```

Paquete Ada.Real_Time

```
   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;

   type Seconds_Count is range implementation-defined;

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

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

Tipos definidos en Ada.Real_Time

- El tipo **Time** representa el tiempo transcurrido desde el comienzo de la ejecución del programa o de un determinado instante de la época
  - El rango debe ser al menos de 50 años

- El tipo **Time_Span** representa intervalos de tiempo
  - La granularidad no debe ser mayor de 1 ms
  - To_Duration(…) convierte Time_Span a Duration

Ejemplo de uso de Ada.Real_Time

```
declare
   with Ada.Real_Time; use Ada.Real_Time;
   Comienzo, Fin : Time;
   Duracion : Time_Span := To_Time_Span(1.7);
            -- o bien Time_Span := Milliseconds(1700);
begin
   Comienzo := Clock;
   ... -- secuencia de instrucciones
   Fin := Clock;
   if Fin - Comienzo > Duracion then
      raise Tiempo_Excedido; -- excepción definida por usuario
   end if;
end;
```

Ada.Real_Time Vs Ada.Calendar

- Ofrece **mayor precisión** (menor granularidad)
- Proporciona un **valor monótono** creciente

Índice

1. Requerimientos de tiempo en un STR
2. Acceso a relojes
3. **Retardos**
4. Temporizadores
5. Análisis de requerimientos temporales

Concepto y tipos de Retardo

- Suspender la ejecución de una tarea durante un cierto tiempo
- Tipos de retardos:
  - **Relativo**
    - La ejecución se suspende durante un intervalo de tiempo relativo al instante actual
  - **Absoluto**
    - La ejecución se suspende hasta que se llegue a un instante determinado de tiempo absoluto

Retrasando una tarea

Espera ocupada

```
Comienzo := Clock;  -- de Ada.Calendar
loop
   exit when (Clock - Comienzo) > 10.0;
end loop;
```

Debemos evitar esperas ocupadas

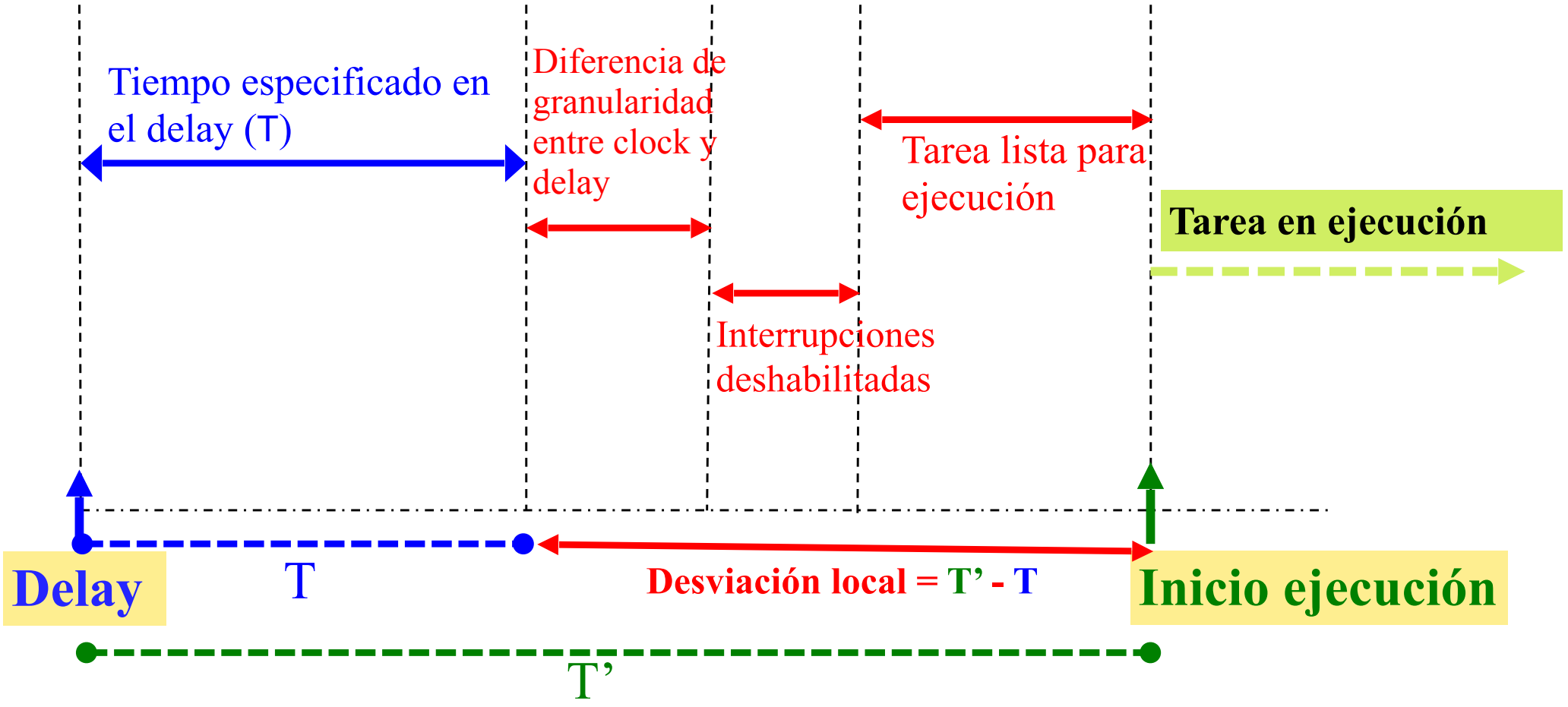
mediante primitivas del lenguaje y/o del S.O.

Retardos relativos en Ada

delay (*T*);

- Suspende la ejecución de la tarea que la invoca durante el intervalo de tiempo que indica el valor de *T*
- *T :* **Duration;** -- por tanto se mide en segundos
- Si *T* ≤ *0* no se produce ningún retardo

Ejecución de un retardo



T = retardo especificado

T' = retardo real

Retardos absolutos en Ada

delay until (*T*);

- Suspende la ejecución de la tarea que la invoca hasta que el valor del reloj (*clock*) sea igual al especificado por *T*
- *T* : **Time;** -- de Ada.Calendar o Ada.Real_Time
- Si *T* ≤ *clock* no se produce ningún retardo

Tarea periódica en Ada

```
task T;

task body T is
begin
   loop
      Accion;
      delay 5.0;
   end loop;
end T;
```

```
task body T is
   periodo: constant Duration := 5.0;
   Siguiente : Time;
begin
   Siguiente := Clock + periodo;
   loop
      Accion;
      delay until Siguiente;
      Siguiente := Siguiente + periodo;
   end loop;
end T;
```

Desviación acumulativa

El delay espera 5 segundos como mínimo

Desviación local

Se ejecutará con una media de 5 segundos

Ejemplo de tarea periódica

```
with Ada.Real_Time; use Ada.Real_Time;
with Data_Types; use Data_Types;
with IO; use IO;

task body Control_Temperatura is
   LT: Lectura_Temp;
   AC: Ajuste_Calor;
   Siguiente: Time;
   periodo: Time_Span := Milliseconds(30);
begin
   Siguiente := Clock + periodo;
   loop
      Read(LT);
      Convertir_Temp(LT, AC);
      Write(AC);
      delay until Siguiente;
      Siguiente := Siguiente + periodo;
   end loop;
end Control_Temperatura;
```

Índice

1. Requerimientos de tiempo en un STR
2. Acceso a relojes
3. Retardos
4. **Temporizadores**
5. Análisis de requerimientos temporales

Programación de *timeouts*

- Limitar el tiempo durante el cual una tarea está lista para **esperar una comunicación**
  - Acceso a memoria compartida
  - Paso de mensajes entre tareas
- Limitar el **tiempo de ejecución de una acción**
  - Detección de pérdidas de plazo
  - Aplicación al cómputo impreciso

Sentencia *Select then abort* en Ada

- Se utiliza para notificaciones asíncronas: **transferencia asíncrona de control** (**ATC**)

```
select
   delay T; -- o también delay until
   -- sentencias opcionales
then abort
   -- sentencias abortables
end select;
```

1. Cuando se ejecuta la select, se comprueba si el tiempo de espera del delay ha expirado
   1.1. Si ha expirado, se ejecutan las sentencias opcionales y después la select termina.
   1.2. Si no ha expirado, se inicia la ejecución de las sentencias abortables
2. Si expira el tiempo de espera y todavía se están ejecutando las sentencias abortables, entonces se interrumpe su ejecución y se ejecutan las sentencias opcionales, después de lo cual termina la select
3. Si termina la ejecución de las sentencias abortables antes de que expire el tiempo de espera, entonces se cancela esta espera y termina la select (sin ejecutarse las sentencias opcionales)

Detección de pérdidas de plazo

```
select
   delay 0.1;
   -- recuperación de la pérdida del plazo
then abort
   -- acción
end select;
```

- Permite también añadir código para recuperación de fallos

Aplicación al cómputo impreciso

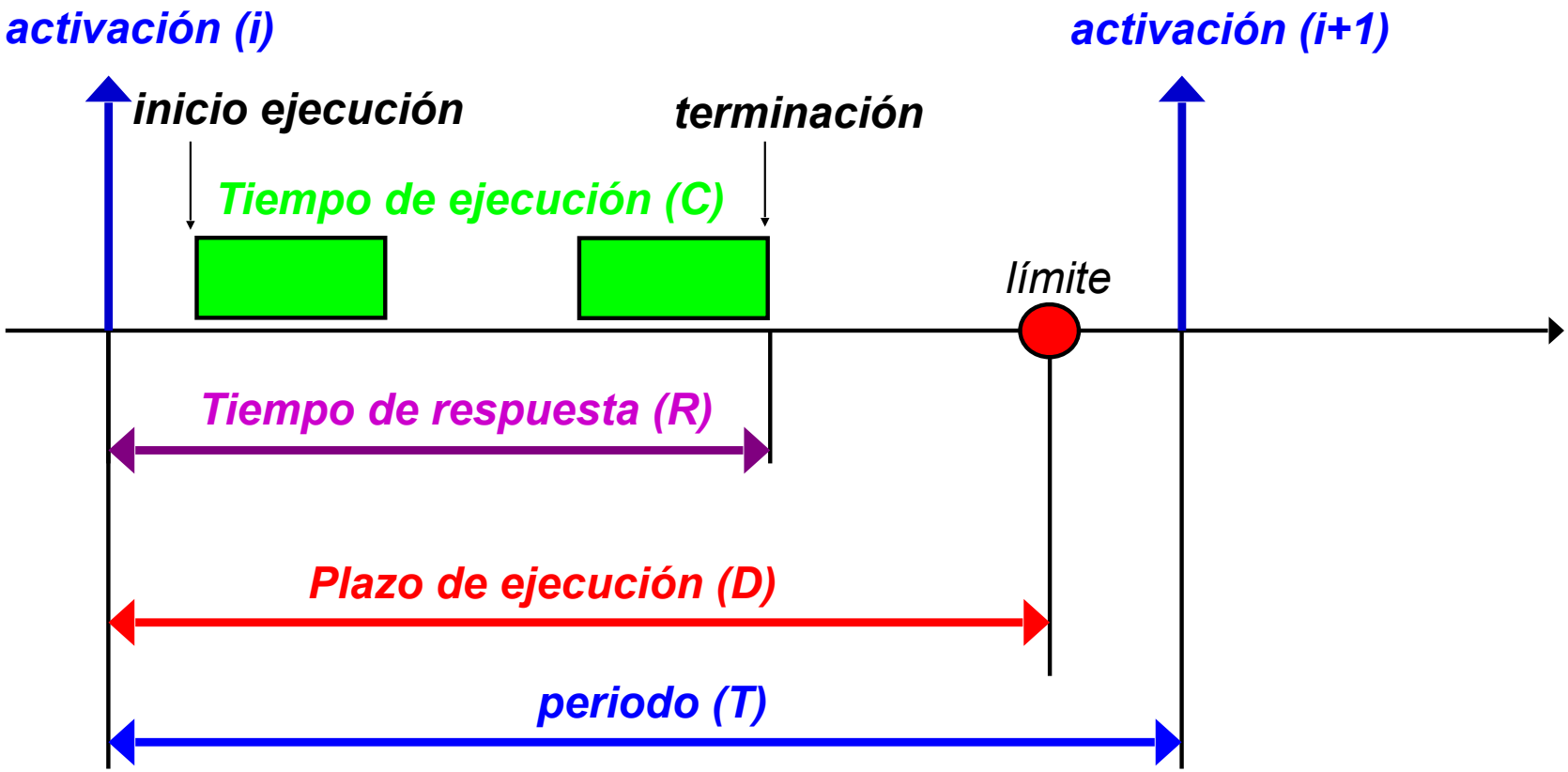
- Se trata de ejecutar rápidamente una parte obligatoria de un cálculo, y de iterar sobre una parte opcional que mejora el resultado mientras quede tiempo

```
begin
   -- parte obligatoria
   Escribe(…); -- actualizamos el resultado inicial
   select
      delay until Fin_Plazo;
   then abort
      loop
         -- mejorar el resultado
         Escribe(…);
      end loop;
   end select;
end;
```

Índice

1. Requerimientos de tiempo en un STR
2. Acceso a relojes
3. Retardos
4. Temporizadores
5. **Análisis de requerimientos temporales**

Ejecución de una tarea de tiempo real



Requerimientos temporales de tareas

- ejecutar **tareas periódicas**
- ejecutar **tareas aperiódicas** o **esporádicas**, cuando ocurre un evento
- completar la ejecución de todas las tareas dentro de su **plazo de respuesta**

Análisis temporal

- Si el sistema cumple determinadas propiedades estructurales (modelo de tareas) se puede analizar su comportamiento temporal

- Los métodos de análisis temporal están estrechamente relacionados con la planificación de tareas
  - el método de planificación debe asegurar un comportamiento temporal **previsible** y **analizable**

Ada y STR críticos

- **Perfil de Ravenscar**
  - Impone ciertas restricciones a la parte concurrente de Ada para poder realizar análisis temporales y permitir una implementación eficiente del núcleo de ejecución
  - **pragma** Profile (Ravenscar);
  - Objetivos:
    - Conseguir un modelo de ejecución concurrente determinista
    - Permitir una implementación pequeña y eficiente

https://es.wikipedia.org/wiki/Perfil_de_Ravenscar

- **Lenguaje Spark**
  - Subconjunto de Ada con anotaciones formales para realizar automáticamente análisis de flujo de datos
  - Diseñado para sistemas de alta integridad

https://es.wikipedia.org/wiki/SPARK

https://es.hrvwiki.net/wiki/SPARK_(programming_language)

Conclusiones

- El lenguaje de programación o el S.O. debe proporcionar **utilidades de control de tiempo**
- Se deben usar primitivas del lenguaje o del S.O. para evitar **bucles de espera ocupada**
- En Ada, las tareas periódicas deben ser implementadas usando la sentencia **delay until**
- En STR es frecuente la programación de **esperas limitadas**
- Las tareas se deben **planificar** para que cumplan con sus requerimientos temporales
- Se puede restringir el modelo de concurrencia de Ada para **implementar STR críticos**

Bibliografía Recomendada

Sistemas de tiempo real y lenguajes de programación (**3ª edición**)
Alan Burns and Andy Wellings
Addison Wesley (2002)

? Capítulo 12 (excepto apartado 12.8 y lo referente a otros lenguajes)

Concurrency in Ada (**2nd edition**)
Alan Burns and Andy Wellings
Cambridge University Press (1998)

? Capítulos 2 (Apartado 2.5)

Otras fuentes de información

Manual de Referencia de Ada2005

- Secciones: 9.6 ; 9.7.4
- Anexos: D.8 ; D.13.1