Tema 10

La metodología HRT-HOOD

Universidad de Extremadura

Tema 10: La metodología HRT-HOOD

1. Introducción
2. Especificación de requisitos
3. Tipos de objetos
4. El diseño de la arquitectura lógica
5. Traducción a Ada
6. Traducción a POSIX/C

Introducción

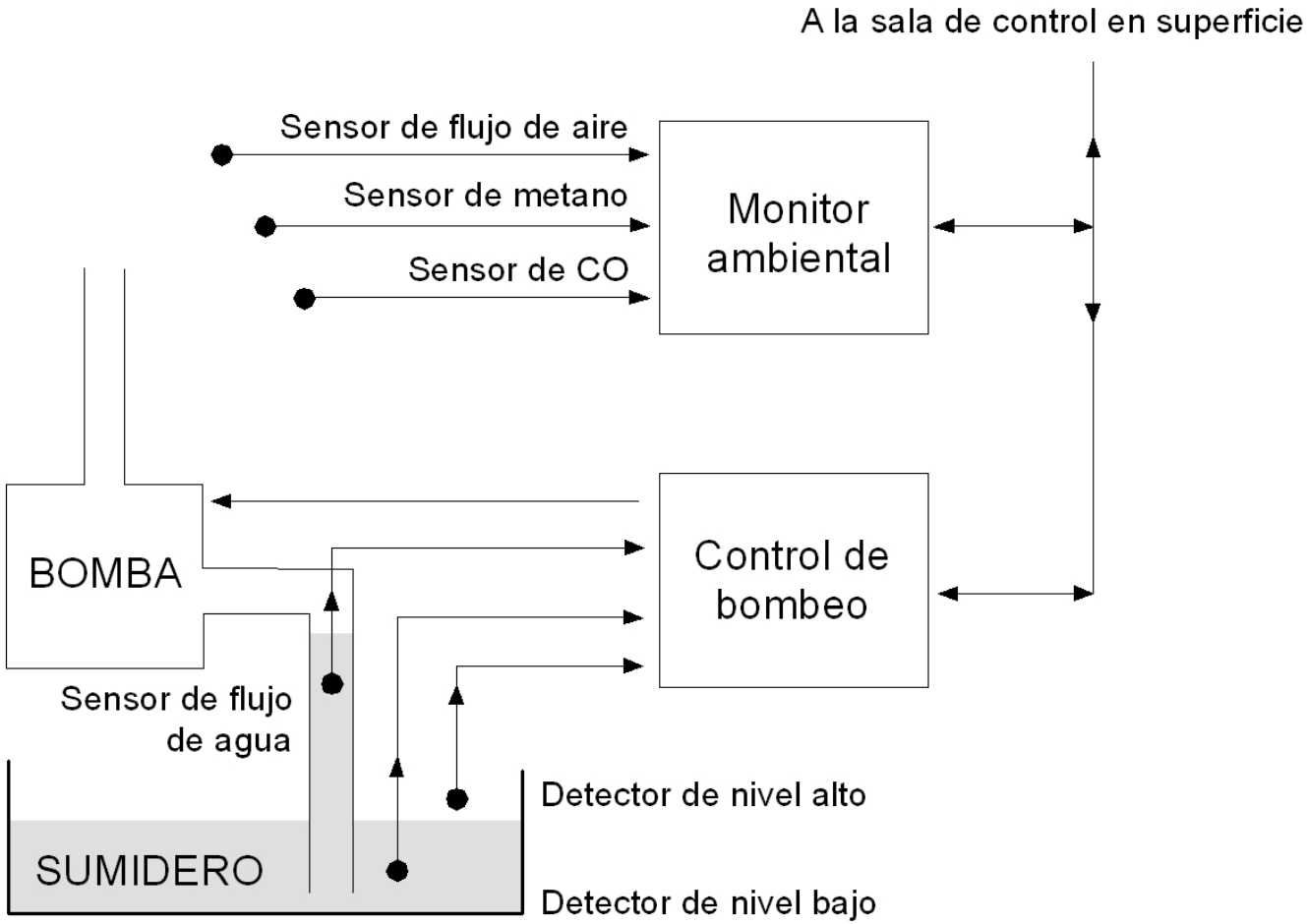
- HRT-HOOD significa Diseño Jerárquico Orientado a Objetos de Sistemas Críticos de Tiempo Real :

  (***H**ard **R**eal **T**ime **H**ierarchical **O**bject **O**riented **D**esign*)

- Se centra en el diseño de la arquitectura lógica y física del sistema y usa una notación basada en objetos
- En este capítulo utilizamos una versión simplificada de HRT-HOOD y la aplicamos al caso de estudio de un sistema de drenaje de una mina
- El caso de la mina aparece en numerosas ocasiones en la literatura sobre sistemas de tiempo real porque muestra de forma simple las características más típicas de estos sistemas
- Se asume que se implementa en un sistema monoprocesador con registros de entrada/salida mapeados en memoria

Especificación de requisitos

La función del sistema es bombear a la superficie el agua que se acumula en un sumidero del pozo de una mina



Especificación de requisitos

Requisitos funcionales

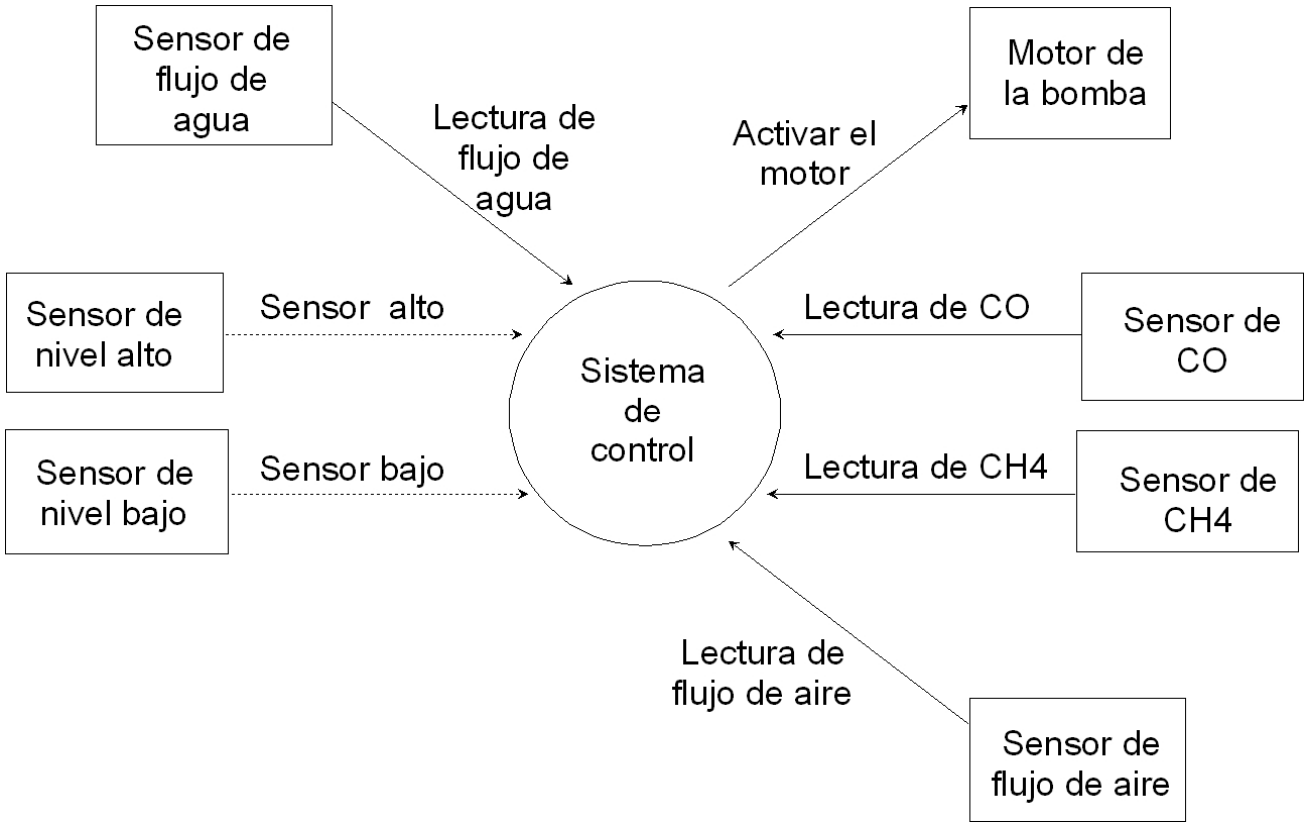
La especificación funcional del sistema abarca cuatro componentes:

- **Control de bombeo**. Monitoriza los niveles del agua en el sumidero. Cuando se alcanza el límite superior o cuando lo requiera el operador, activa la bomba, que comienza a drenar agua hasta que esta alcanza el límite inferior o hasta que así lo requiera el operador. Si se especifica, se puede detectar el flujo de agua en la tubería

  Sólo se permite la operación de la bomba si el nivel de metano en el pozo está por debajo de un nivel crítico (Requisito Principal)
- **Monitorización del ambiente**. Mide el nivel de metano y monóxido de carbono en el ambiente y si hay un flujo de aire adecuado. Cuando se alcanza algún nivel crítico, se activa una alarma
- **Interacción con el operador**. Todo el sistema es controlado por un operador en superficie, al que se informa de los eventos críticos
- **Monitorización del sistema**. Todos los eventos se almacenan en una base de datos y son recuperados cuando lo requiera el operador

Especificación de requisitos

Gráfico de dispositivos externos



- Flechas <u>discontinuas</u>: el dispositivo interrumpe al sistema
- Flechas <u>continuas</u>: el dispositivo es controlado mediante escrutinio

Especificación de requisitos

Requisitos no funcionales

Los requisitos no funcionales de un sistema empotrado son dos:

- La temporización
- La seguridad

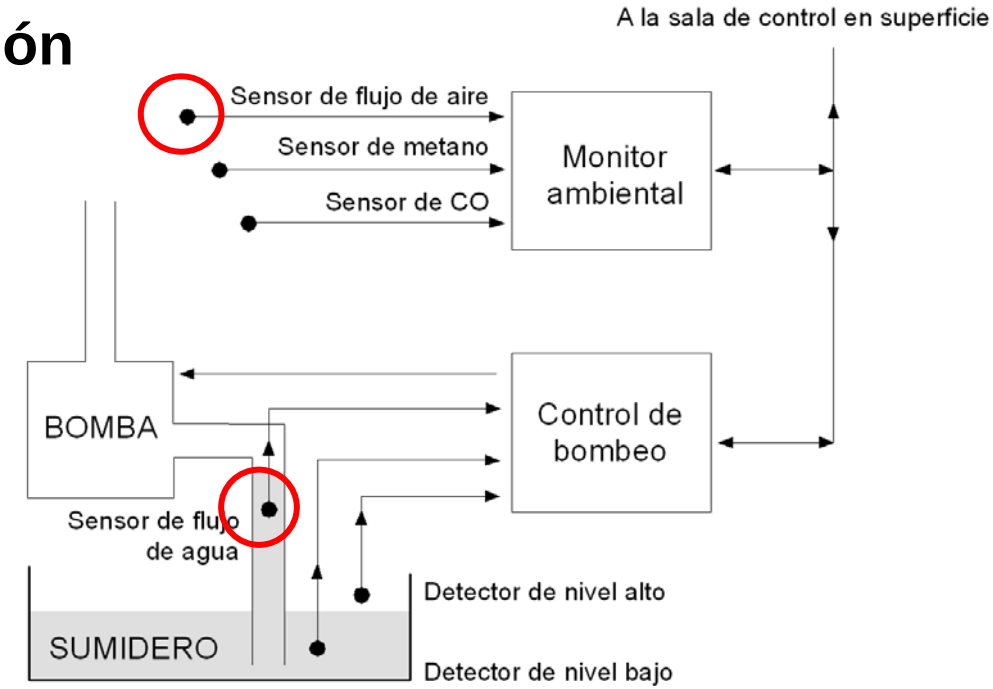
Los **requisitos temporales** de son tres:

1. Periodos de monitorización
2. Plazo de parada
3. Plazo de información al operador

Especificación de requisitos

Periodos de monitorización

En entornos industriales como el que nos ocupa, la frecuencia de lectura de sensores puede estar dictada por la legislación



- El **sensor de flujo de aire** tiene un periodo de *T* = 100 ms. y un plazo *D* del mismo valor
- El **sensor de flujo de agua** determina el estado real de la bomba (funcionando o parada). La inercia natural de variación del flujo impone un periodo amplio *T* = 1000 ms

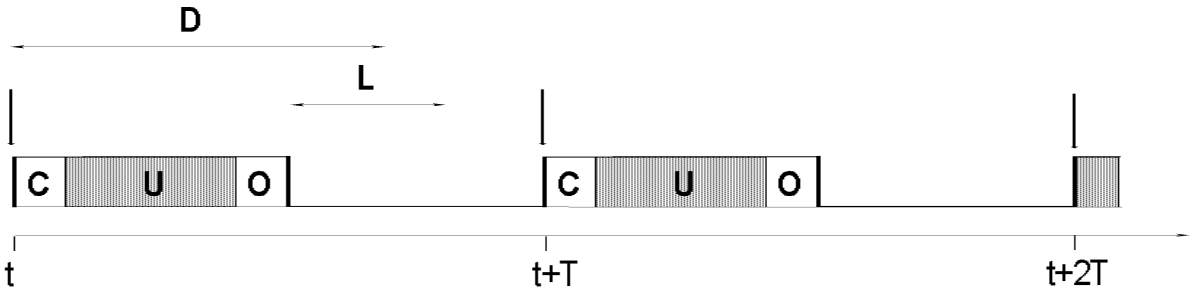
Especificación de requisitos

Periodos de monitorización

Flujo de agua:

Dos lecturas consecutivas a 1: La bomba está funcionando

Dos lecturas consecutivas a 0: La bomba está parada



C Comprobacion de bit 'hecho'
D Plazo de la tarea
L Latencia del dispositivo
O Orden de lectura
U Uso de la lectura

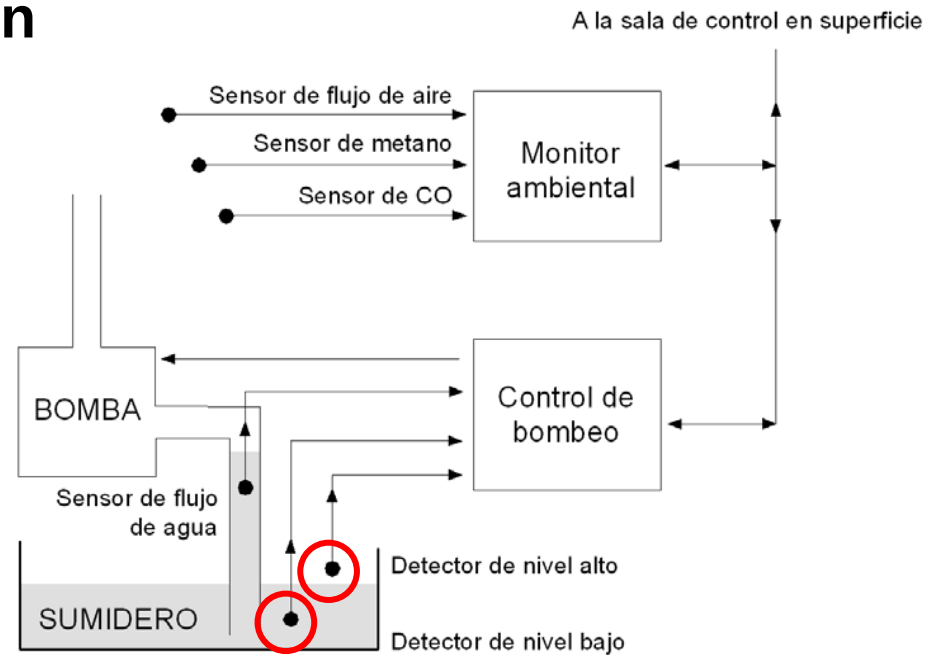
- Utilizando la técnica de ***desplazamiento de periodo*** la edad de cada lectura viene dada por $T - D < e < T + D$
- Para que este margen sea estrecho se impone un plazo breve ($D = 40$ ms.), lo que garantiza edades de lecturas en el margen de 960 y 1040 ms.

Especificación de requisitos

Periodos de monitorización

Los sensores de **nivel** de agua, a diferencia del resto, son controlados por interrupción atendida por programa.

El sistema debe atender al evento de nivel (alto o bajo) con D = 200 ms.

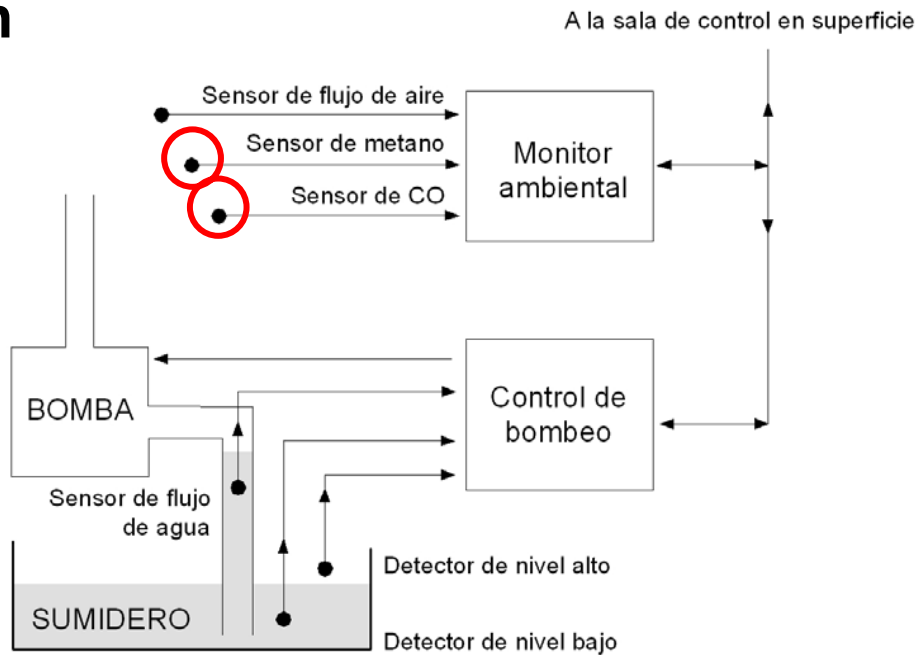


- La experiencia en el pozo muestra que es suficiente un intervalo mínimo entre eventos de 6 segundos (T = 6000 ms.)
- Los sensores de **flujo** y **nivel** proporcionan valores binarios (flujo/no flujo, alcanzado/no alcanzado) por lo que no necesitan ADC's. Como consecuencia, la latencia es 0

Especificación de requisitos

Periodos de monitorización

Los sensores de CH4 y de **CO**, en contraste, proporcionan valores cuantitativos, por lo que llevan ADC's incorporados



Estos ADC's tienen una latencia L de 40 ms., lo que acarrea una lectura más lenta

El plazo de estas tareas de lectura ha de cumplir que $D \leq T - L$

Para el CO, podemos imponer un periodo T = 100 ms. y un plazo D = 60 ms.

Especificación de requisitos

Plazo de parada

- En cuanto al **CH_4**, el sistema de seguridad de la mina se basa en:

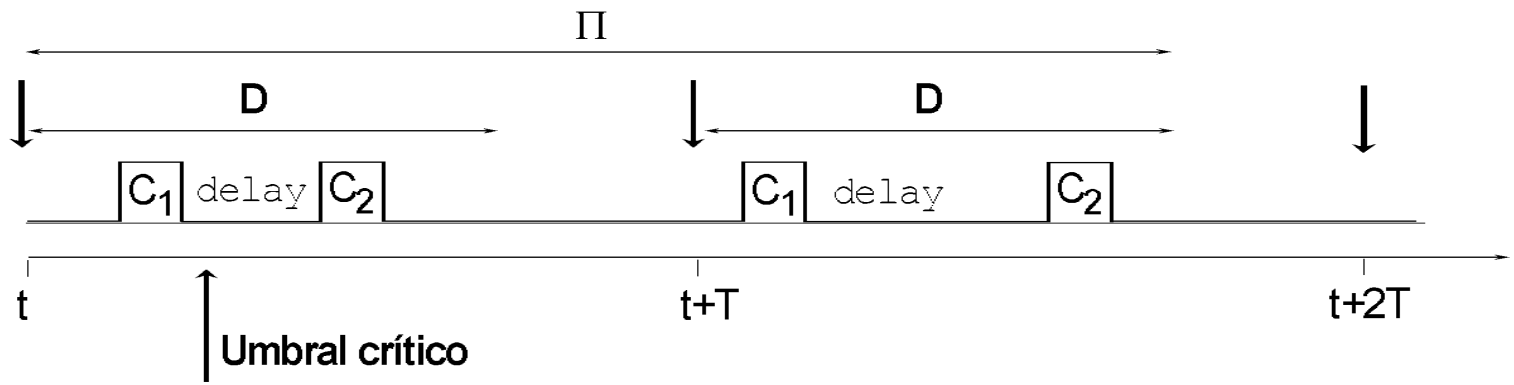
> *"Para evitar que la operación de la bomba pueda provocar la explosión del metano, debe haber un plazo Π dentro del cual la bomba debe ser apagada una vez que el CH_4 excede un umbral de concentración crítico"*

- Si el umbral de concentración crítico se alcanza inmediatamente después de realizar una lectura, no será detectado hasta la próxima

Especificación de requisitos

Plazo de parada

Utilizando la técnica de **replanificación**, en el caso más desfavorable puede pasar un tiempo de $T+D$ desde que el umbral crítico se alcanza hasta que se da la orden de parar el motor de la bomba:

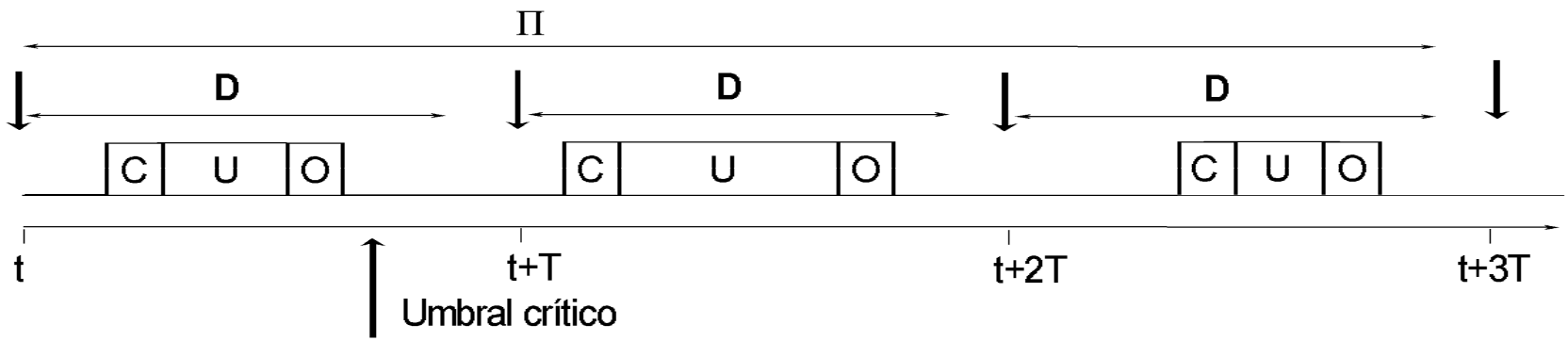


El objetivo del sistema es garantizar que $T+D < \Pi$, lo que se consigue disminuyendo T y/o disminuyendo D

Especificación de requisitos

Plazo de parada

- Utilizando la técnica de planificación con **desplazamiento de periodo** este tiempo se dilata: $2T + D$



El objetivo del sistema es garantizar que $2T+D < \Pi$, lo que se consigue disminuyendo T y/o disminuyendo D

Especificación de requisitos

Plazo de parada

- El CH_4 se concentra rápidamente $\Rightarrow$ respuesta rápida de Π = 200 ms.
- Esta respuesta se garantiza con una tarea periódica con desplazamiento con periodo T = 80 ms y con plazo D = 30 ms., pues 2*80 + 30 = 190 < 200
- Dado que la latencia L del ADC del sensor de metano es de 40 ms., se respeta la restricción de que $D + L \leq T$, ya que 30 + 40 = 70 $\leq$ 80

Especificación de requisitos

Plazo de información al operador

El operador debe ser informado:

- En el plazo de un segundo tras la detección de valores críticos de CH_4 y CO
- En el plazo de dos segundos tras la detección de valores críticos de flujo de aire
- En el plazo de tres segundos de un fallo en la operación de la bomba

Especificación de requisitos

Tabla resumen de periodos y plazos de las tareas asociadas a los sensores:

Sensor	Tipo	'Periodo' (ms)	Plazo (ms)
CH_4	Periódico	80	30
CO	Periódico	100	60
Flujo de aire	Periódico	100	100
Flujo de agua	Periódico	1000	40
Nivel de agua	Esporádico	6000	600

Tipos de objetos

HRT-HOOD facilita el diseño de la **arquitectura lógica** del sistema proporcionando los siguientes tipos de objetos:

1. **Objeto pasivo**.

- No controla cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos (no tiene restricciones de sincronización, como por ejemplo la exclusión mutua o las condiciones de sincronización de una guarda).
- No invoca operaciones en otros objetos de forma espontánea (es decir, no tiene tareas).

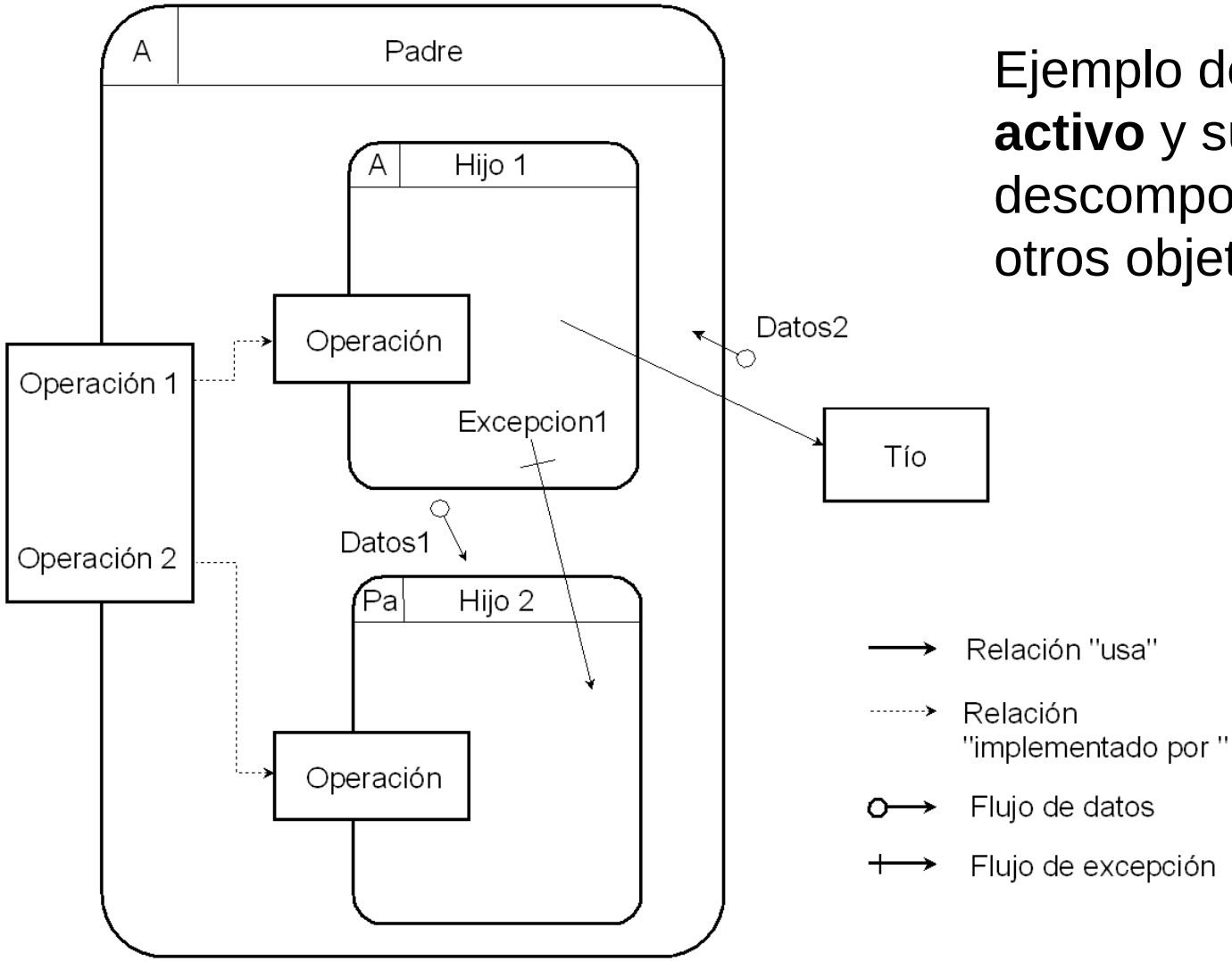
2. **Objeto activo**. Es la clase más general de objetos y no tiene restricciones.

- Puede controlar cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos.
- Puede invocar operaciones en otros objetos espontáneamente

Tipos de objetos

3. **Objeto protegido**. Deben ser analizables, ya que imponen bloqueo a sus invocantes
   - Puede controlar cuándo se ejecutan sus operaciones al ser éstas invocadas por otros objetos. Se permite al diseñador restringir las condiciones de sincronización del objeto
   - No puede invocar operaciones en otros objetos de forma espontánea
4. **Objeto cíclico**. Representa una actividad periódica
   - Puede invocar operaciones en otros objetos de forma espontánea
   - No tiene interfaz
5. **Objeto esporádico**. Representa una actividad esporádica
   - Puede invocar operaciones en otros objetos de forma espontánea
   - Tiene una *única* operación, que es la que lo invoca

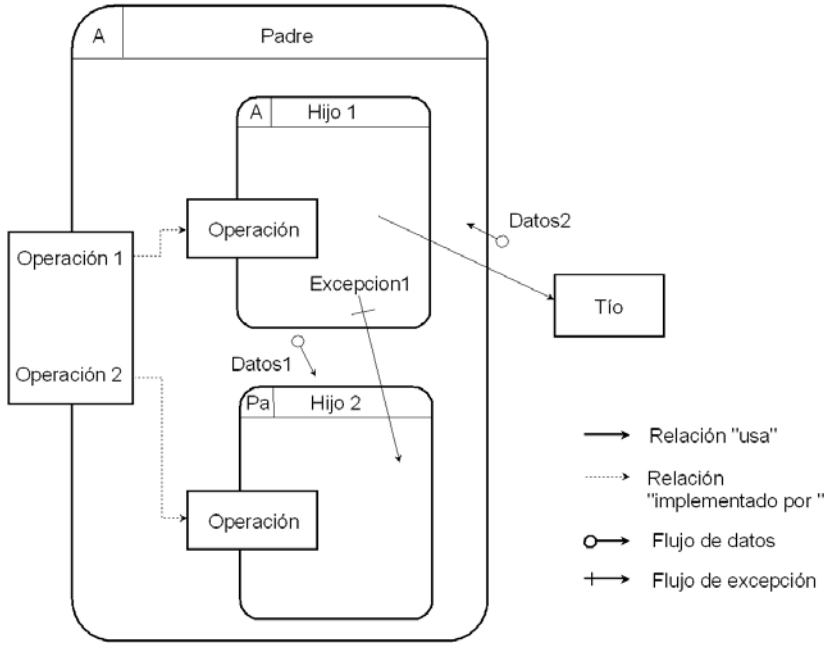
Tipos de objetos



Ejemplo de **objeto activo** y su descomposición en otros objetos

Tipos de objetos

- Una aplicación crítica de tiempo real HRT-HOOD contendrá en su nivel terminal de descomposición sólo objetos
  - Cíclicos,
  - esporádicos,
  - protegidos y
  - pasivos



- Los activos no se permiten, ya que no son analizables
- Pueden utilizarse en la descomposición del sistema principal, pero debe transformarse en uno de los anteriores antes de alcanzar el nivel terminal

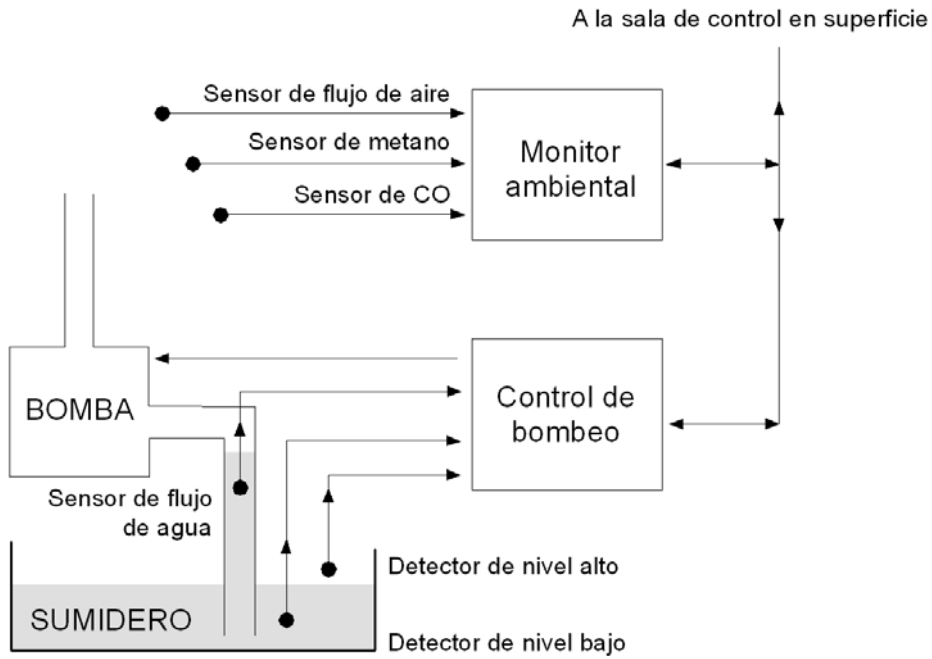
El diseño de la arquitectura lógica

La **arquitectura lógica** trata los requisitos funcionales. Estos son independientes de las restricciones físicas impuestas por el entorno de ejecución (velocidad del procesador, etc.)

Descomposición de primer nivel

- El primer paso es identificar las clases de objetos adecuados a partir de los cuales se va a construir el sistema
- Los requisitos funcionales del sistema sugieren cuatro subsistemas:
  1. **Control de bombeo**. Responsable de la operación de la bomba
  2. **Monitorización del ambiente**. Responsable de la monitorización del ambiente
  3. **Consola de operador**. Responsable de la interfaz con el operador
  4. **Registro de datos**. Responsable del registro de datos de operaciones y medio ambiente

El diseño de la arquitectura lógica

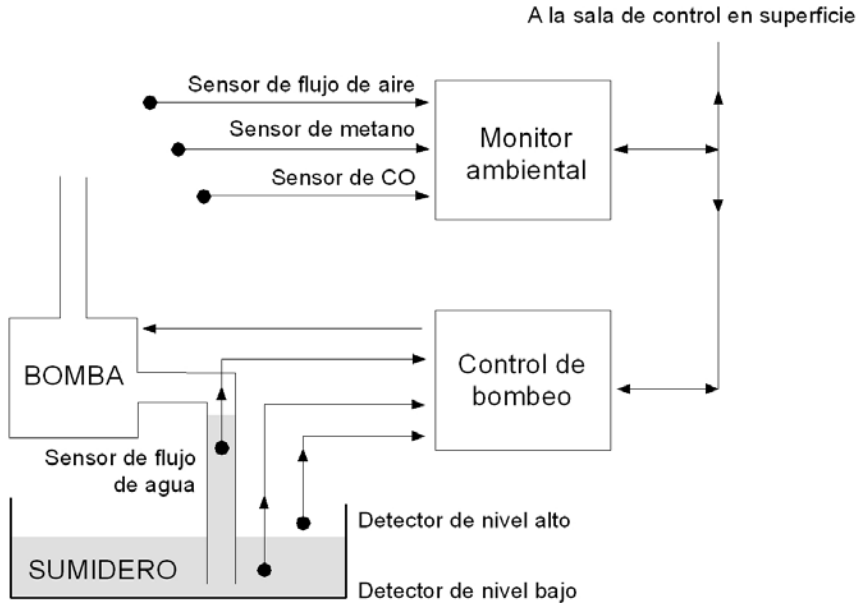


Control de bombeo tiene cuatro operaciones:

- **No seguro** y **Seguro** son invocadas por el *Monitor de entorno* para indicarle si es posible activar la bomba (debido a la concentración en el ambiente de CH_4)
- **Petición de estado** y **Actuar** son invocadas por el *Operador de consola*

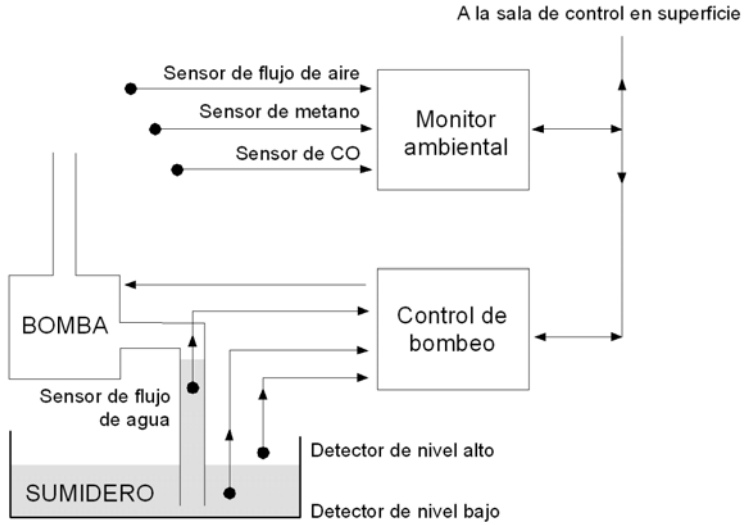
El diseño de la arquitectura lógica

Monitor de entorno tiene una operación:



- Como una característica adicional de seguridad, el *Control de Bombeo* pedirá el nivel de CH_4 al Monitor de entorno antes de activar la bomba invocando la operación **Comprueba seguro**.
- Si *Control de Bombeo* encuentra que la bomba no puede ser activada o si en teoría la bomba está activada pero no detecta flujo de agua, envía una alarma al operador

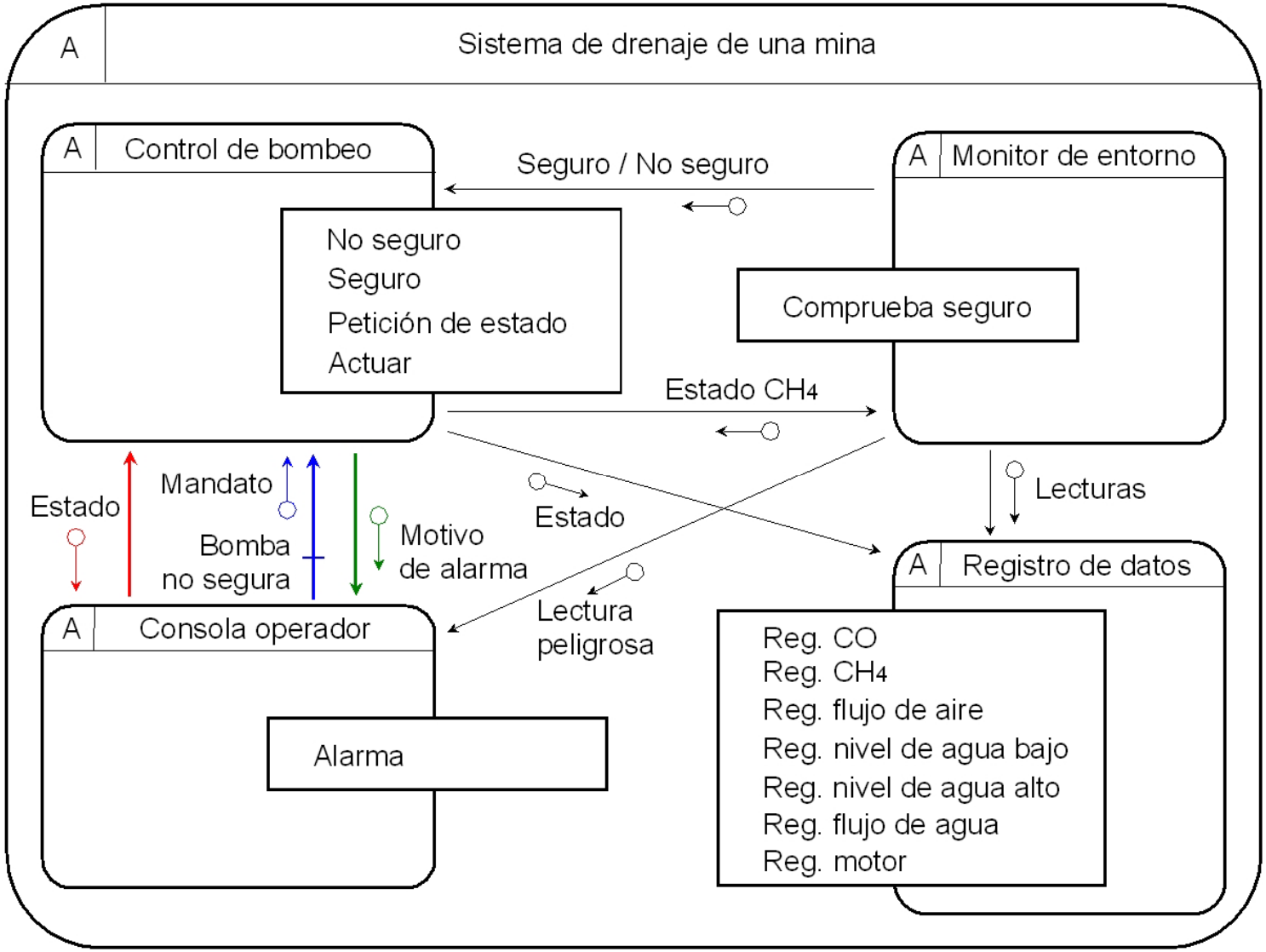
El diseño de la arquitectura lógica



Consola del operador
tiene una operación

- La operación **Alarma** es invocada por *Control de Bombeo*, y por *Monitor de Entorno* cuando una lectura es demasiado alta.
- *Consola del operador* puede solicitar el estado de la bomba y operar sobre ella independientemente de los sensores de nivel
- En tal caso, la comprobación del nivel de CH_4 también se hace, de modo que el arrancar la bomba manualmente eleva una excepción si la bomba no puede ser activada

El diseño de la arquitectura lógica



El diseño de la arquitectura lógica

Control de bombeo Se descompone en tres objetos:

Motor Controla el motor de la bomba

- Únicamente responde a mandatos, requiere exclusión mutua para estos y no invoca de forma espontánea otros objetos
- Lo lógico es darle un carácter de objeto *protegido*
- Implementa todas las operaciones de *Control de bombeo*

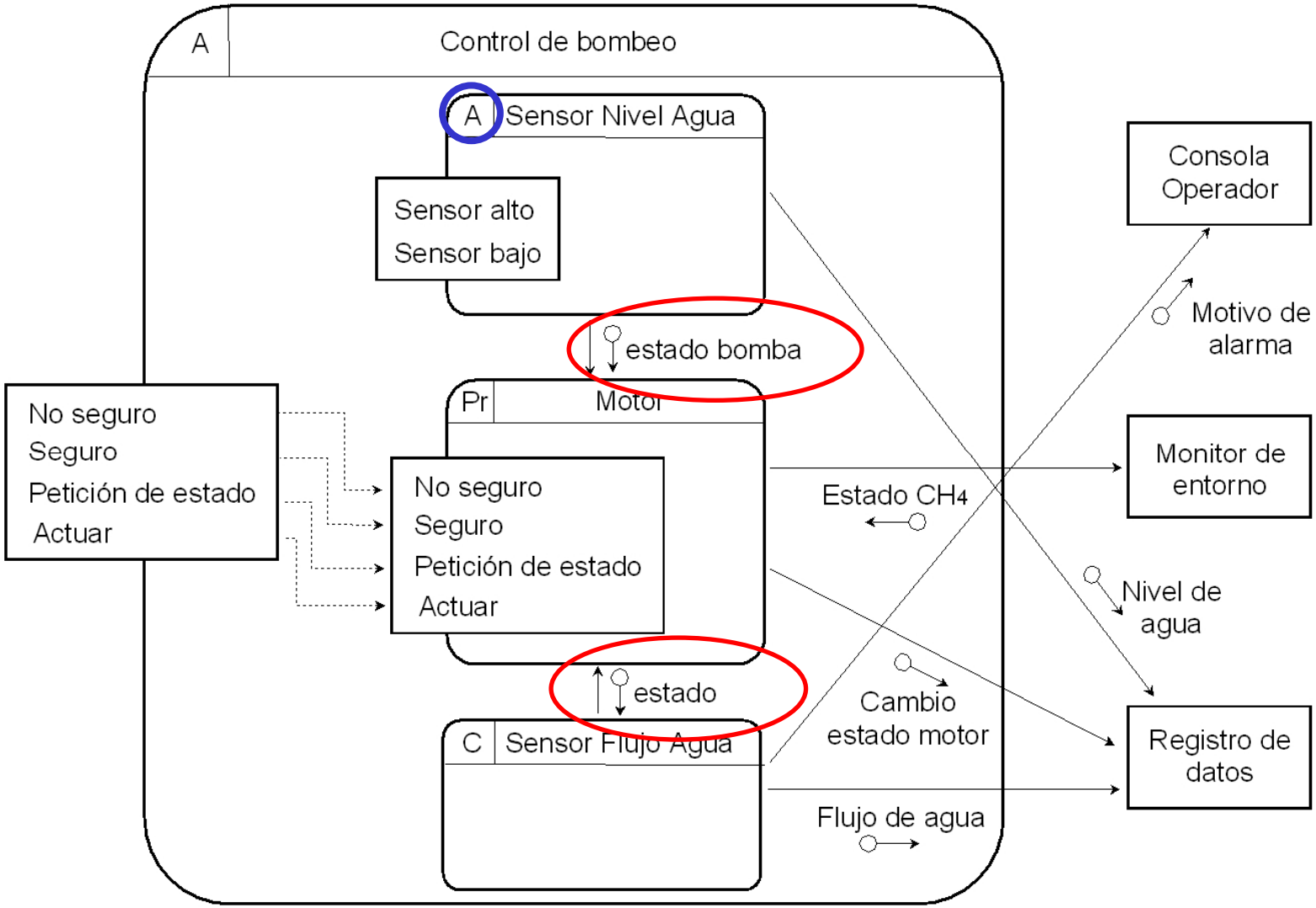
Sensor flujo agua

Objeto *cíclico* que continuamente monitoriza el flujo de agua del sumidero e invoca al objeto **motor** para conocer su estado

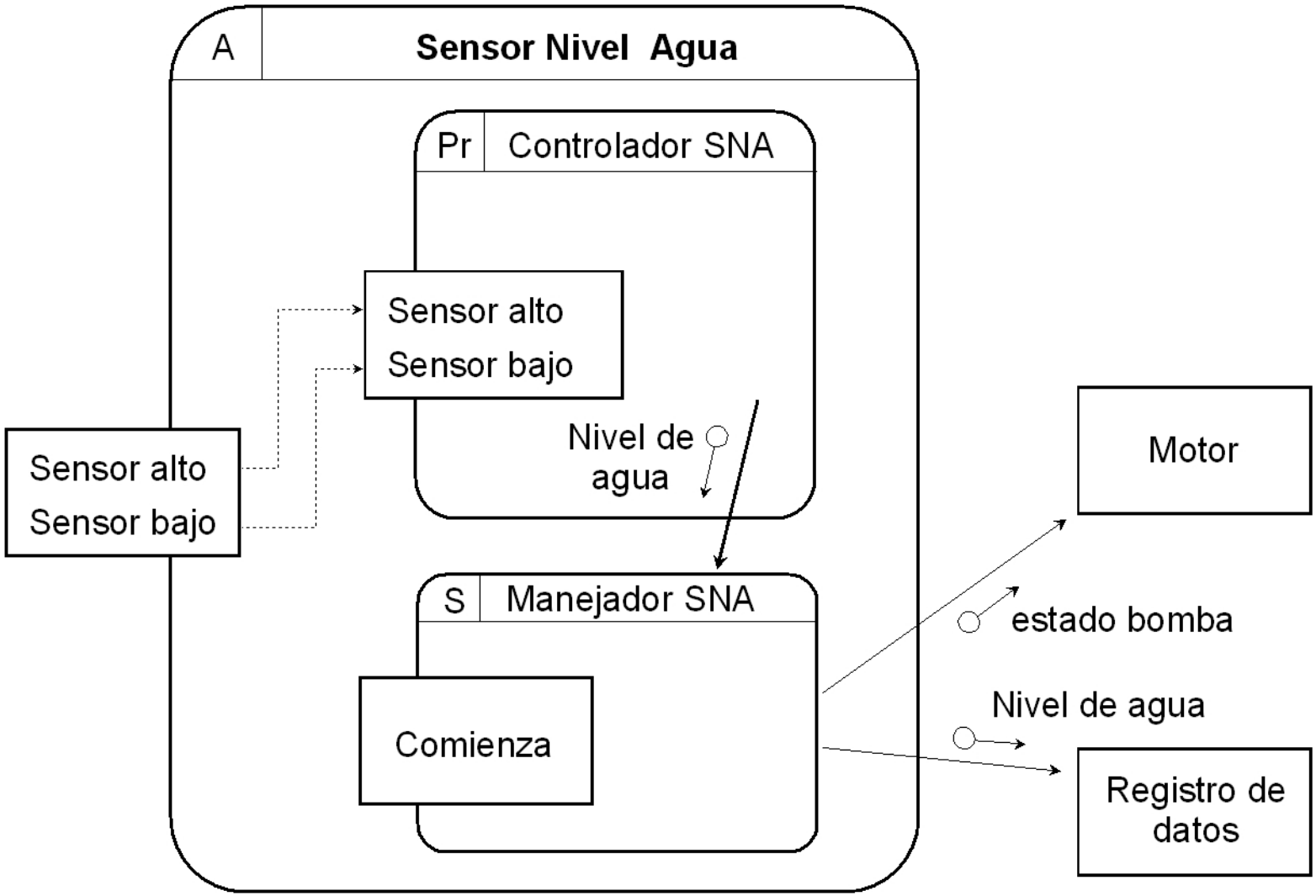
Sensor nivel agua

- Objeto *activo* que soporta las interrupciones de los sensores de nivel de agua
- Se descompone en un objeto *protegido* y otro *esporádico*

El diseño de la arquitectura lógica

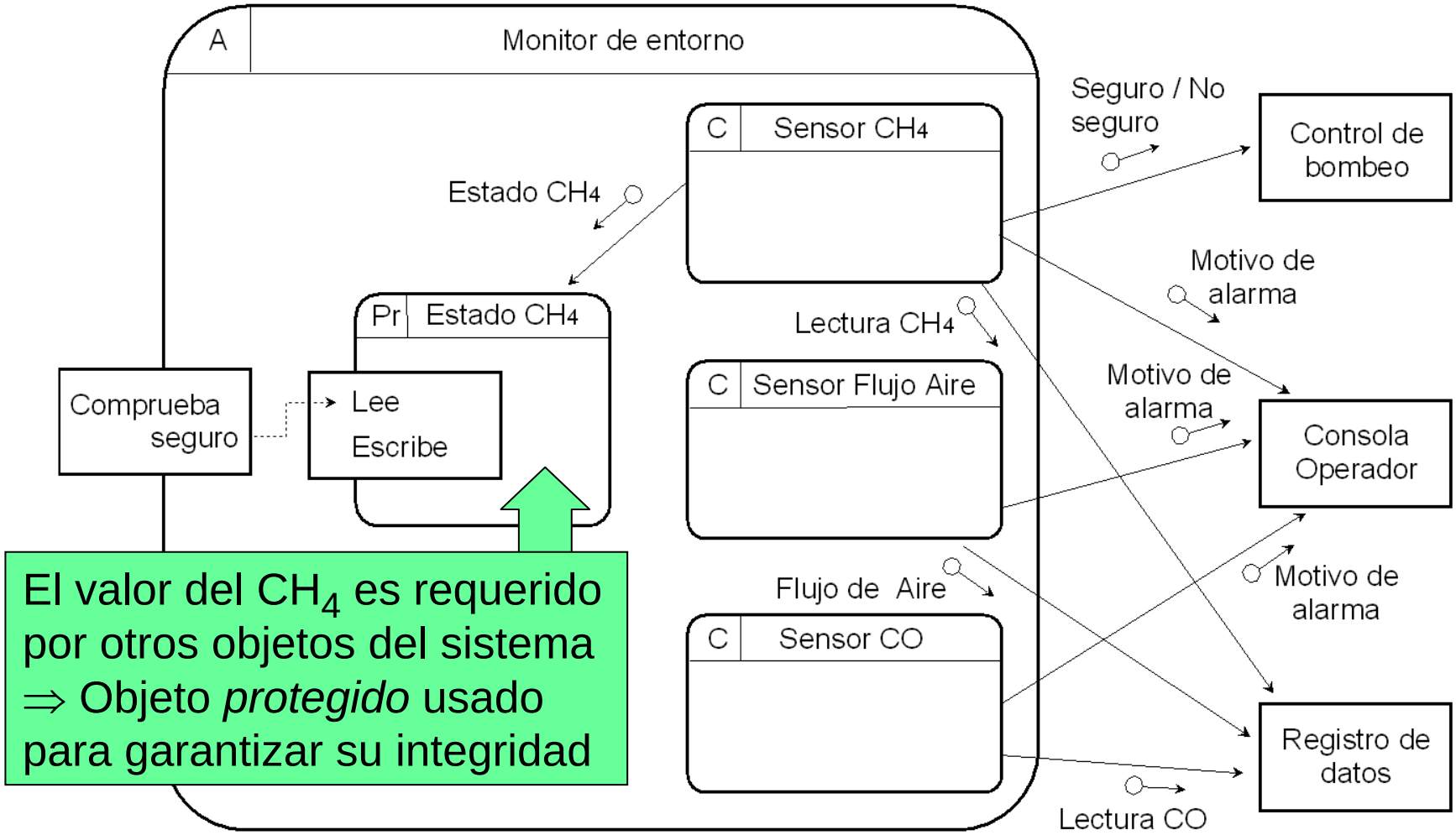


El diseño de la arquitectura lógica



El diseño de la arquitectura lógica

El monitor de entorno



Traducción a Ada

- HRT-HOOD soporta una traducción sistemática a ADA
- Para cada objeto terminal se generan dos paquetes:
  1. Tipos de datos y variables que definen los atributos de tiempo real del objeto
  2. El código del objeto.

Traducción a Ada

El objeto "Motor"

- Los registros del dispositivo que controla el motor se declaran en un paquete denominado `Device_Register_Types`:

```
package Device_Register_Types is
   Word : constant := 2; -- Una palabra tiene dos bytes
   One_Word : constant := 16 -- Una palabra tiene 16 bits
   type Device_Error     is (Clear, Set);
   type Device_Operation is (Clear, Set);
   type Interrupt_Status is (I_Disabled, I_Enable);
   type Device_Status    is (I_Disabled, I_Enable);
   type Csr is
      record
         Error_Bit : Device_Error;
         Operation : Device_Operation;
         Done      : Boolean;
         Interrupt : Interrupt_Status;
         Device    : Device_Status;
      end record;
```

Traducción a Ada

El objeto “Motor”

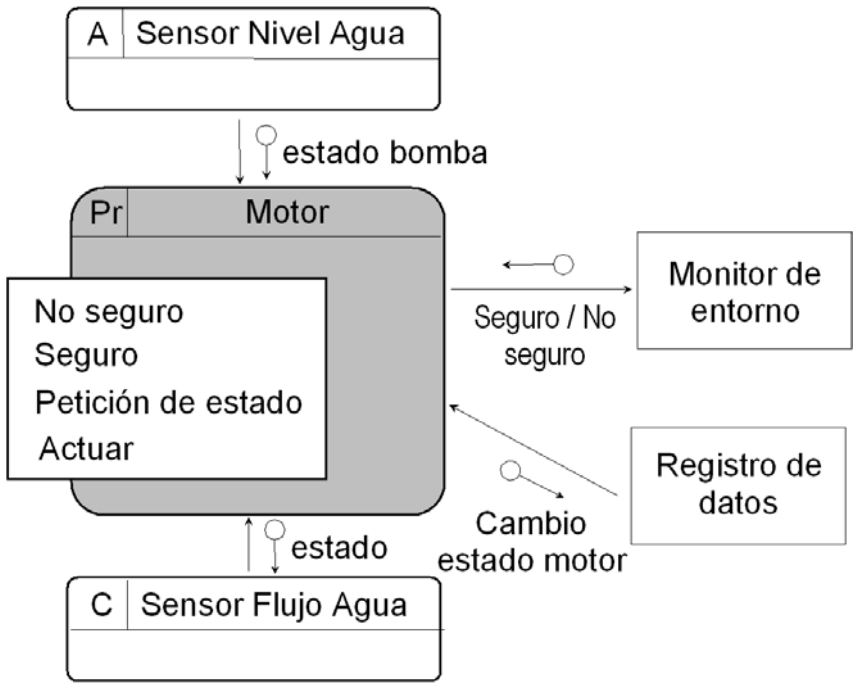
```
  for Device_Error     use (Clear => 0, Set => 1);
  for Device_Operation use (Clear => 0, Set => 1);
  for Interrupt_Status use (I_Disabled => 0, I_Enable => 1);
  for Device_Status    use (I_Disabled => 0, I_Enable => 1);
  for Csr use
    record
      Error_Bit at 0 range 15 .. 15;
      Operation at 0 range 10 .. 10;
      Done      at 0 range  7 ..  7;
      Interrupt at 0 range  6 ..  6;
      Device    at 0 range  0 ..  0;
    end record;
  for Csr’Size      use One_Word;
  for Csr’Alignment use Word;
  for Csr’Bit_Order use Low_Order_First;
end Device_Register_Types;
```

Traducción a Ada

El objeto “Motor”

1. **Atributos de tiempo real**
   (Objeto protegido que requiere un techo de prioridad)

```
package Motor_Rtatt is
  Ceiling_Priority : constant := 10;
end Motor_Rtatt;
```

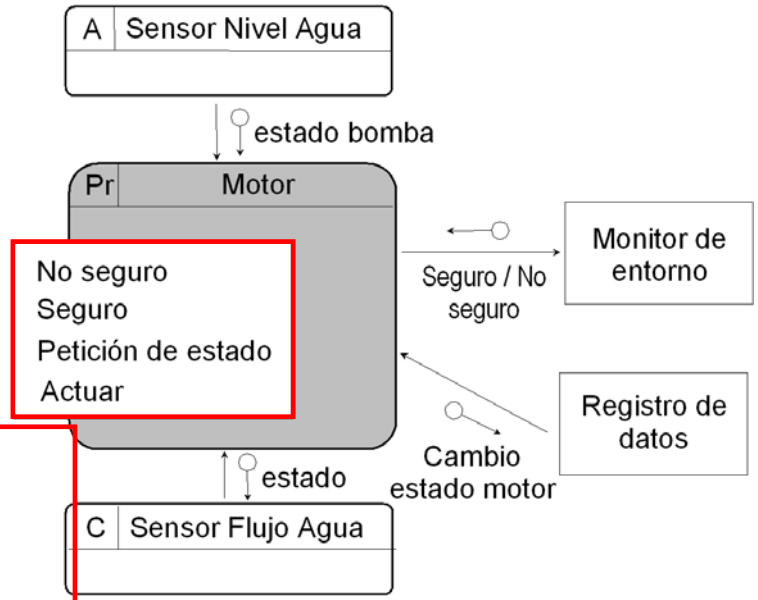


Traducción a Ada

El objeto “Motor”

2. Interfaz

```
package Motor is  -- PROTEGIDO
  type Pump_Status    is (On, Off);
  type Pump_Condition is (Enable, Disable);
  type Motor_State_Changes is
   (Motor_Started, Motor_Stopped,
    Motor_Safe,    Motor_Unsafe);
  type Operational_Status is
    record
      Ps : Pump_Status;
      Pc : Pump_Condition;
    end record;
  Pump_Not_Safe : exception;
  procedure Not_Safe;
  procedure Is_Safe;
  function Request_Status
    return Operational_Status;
  procedure Set_Pump(To : Pump_Staus);
end Motor;
```

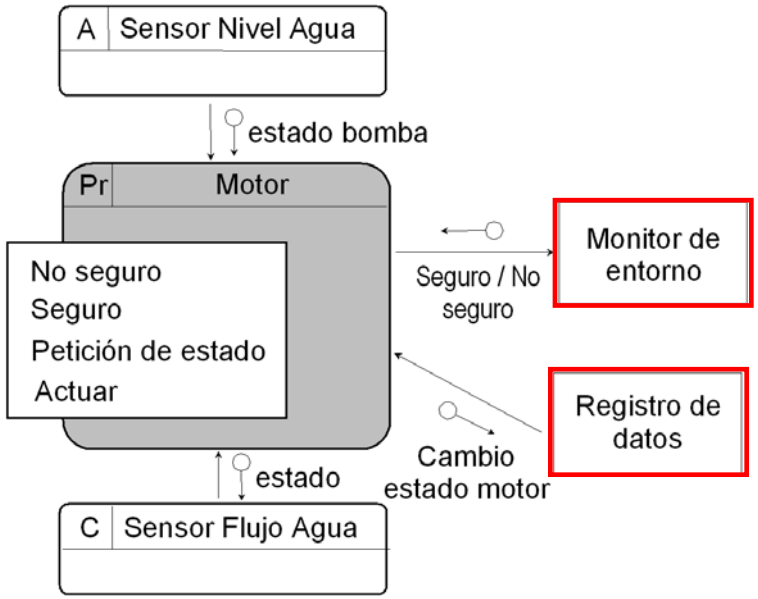


Traducción a Ada

El objeto “Motor”

3. Implementación

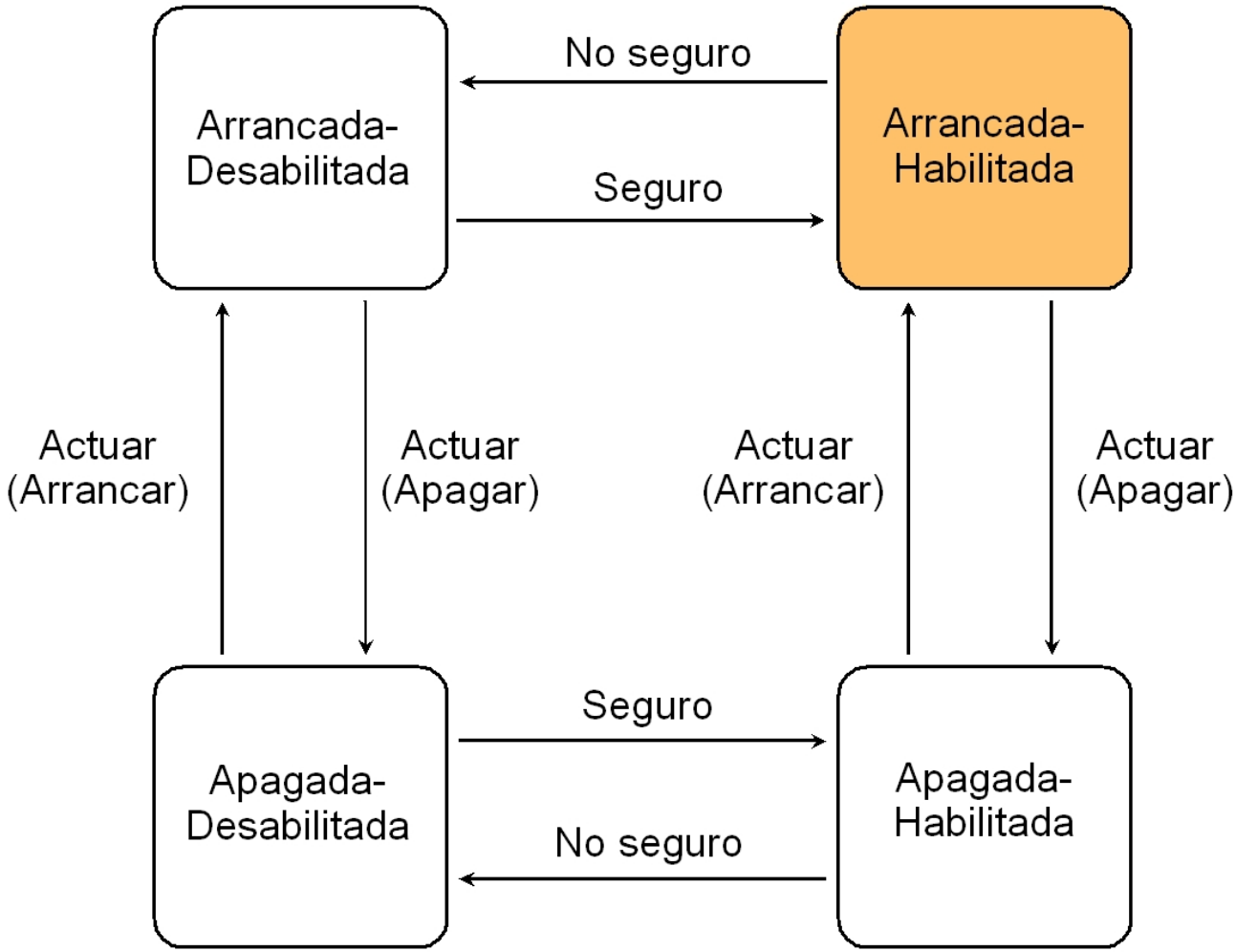
```
with Data_Logger;
with Ch4_Status;
use  Ch4_Status;
with Device_Register_Types;
use  Device_Register_Types;
with System; use System;
with Motor_Rtatt; use Motor_Rtatt;
with System.Storage_Elements;
use  System.Storage_Elements;
package body Motor is
  Control_Reg_Addr : constant
    Address := To_Address(16#AA14#);
  Pcsr : Device_Register_Types.Csr :=
           (Error_Bit => Clear,
            Operation => Clear,
            Done => False,
            Interrupt => I_Enabled,
            Device => D_Enabled);
  for Pcsr'Address use Control_Reg_Addr;
```



Traducción a Ada

El objeto "Motor"

3. Implementación



Traducción a Ada

El objeto “Motor”

3. Implementación

```
package body Motor is
  ...
  protected Agent is
    pragma Priority(Motor_Rtatt.Ceiling_Priority);
    procedure Not_Safe;
    procedure Is_Safe;
    function  Request_Status return Operational_Status;
    procedure Set_Pump(To : Pump_Staus);
  private
    Motor_Status    : Pump_Status    := Off;
    Motor_Condition : Pump_Condition := Disable;
  end Agent;
  procedure Not_Safe is begin Agent.Not_Safe; end;
  procedure Is_Safe  is begin Agent.Is_Safe;  end;
  function  Request_Status return Operational_Status is
    begin return Agent.Request_Status; end;
  procedure Set_Pump(To : Pump_Staus) is
    begin Agent.Set_Pump(To : Pump_Staus); end;
```

Traducción a Ada

El objeto “Motor”

3. **Implementación** del objeto

```
protected body Agent is
  procedure Not_Safe is
  begin
    if Motor_Status = On then
      Pcsr.Operation = Clear; --Apaga motor
      Data_Logger.Motor_Log(Motor_Stopped);
    end if;
    Motor_Condition := Disabled;
    Data_Logger.Motor_Log(Motor_Unsafe);
  end Not_Safe;
  procedure Is_Safe is
  begin
    if Motor_Status = On then
      Pcsr.Operation = Set; --Enciende motor
      Data_Logger.Motor_Log(Motor_Started);
    end if;
    Motor_Condition := Enabled;
    Data_Logger.Motor_Log(Motor_Safe);
  end Is_Safe;
  function Request_Status return Operational_Status is
  begin
    return (Ps => Motor_Status, Pc => Motor_Condition);
  end Request_Status;
```

Traducción a Ada

```
    procedure Set_Pump(To : Pump_Status) is
    begin
      if To = On then
        if Motor_Status = Off then
          if Motor_Condition = Disabled then
            raise Pump_Not_safe;
          end if;
          if Ch4_Status.Read = Motor_Safe then
            Motor_Status := On;
            Pcsr.Operation = Set; --Arranca el motor
            Data_Logger.Motor_Log(Motor_Started);
          else
            raise Pump_Not_Safe;
          end if;
        end if;
      else
        if Motor_Status = On then
          if Motor_Condition = Enabled then
            Motor_Status := Off;
            Pcsr.Operation = Clear; --Apaga el motor
            Data_Logger.Motor_Log(Motor_Stopped);
          end if;
        end if;
      end if;
    end Set_Pump;
  end Agent;
end Motor;
```

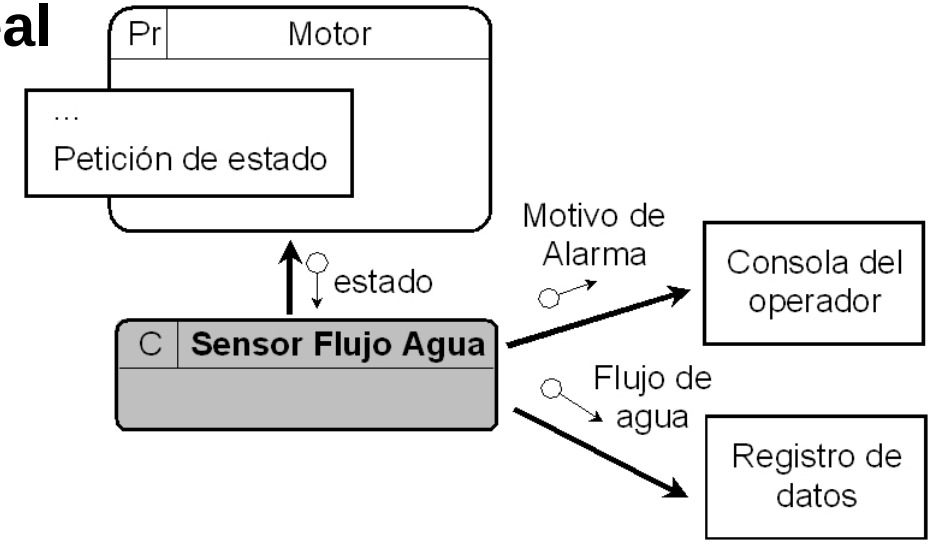
El objeto “Motor”

3. Implementación

Traducción a Ada

El objeto “Sensor Flujo Agua”

1. **Atributos de tiempo real**
   (Objeto cíclico requiere periodo y prioridad)



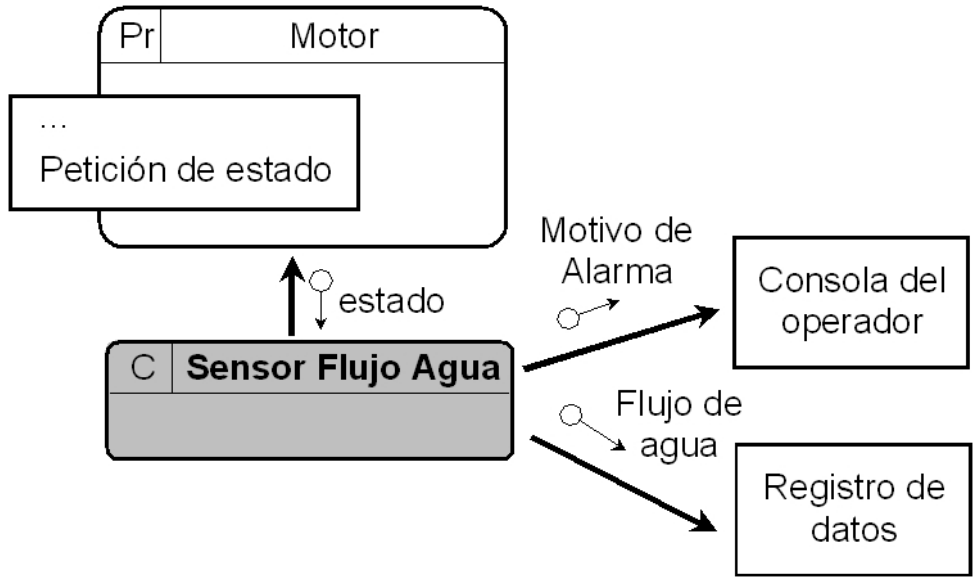

```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
package Water_Flow_Sensor_Rtatt is
  Period : Time_Span := Milliseconds(1000);
  Thread_Priority : constant Priority := 9;
end Water_Flow_Sensor_Rtatt;
```

Traducción a Ada

El objeto “Sensor Flujo Agua”

2. Interfaz

No tiene interfaz al exterior. No obstante: necesitamos el **pragma** para asegurar que se elabora el cuerpo del paquete y necesitamos definir un **tipo** accesible al registro de datos



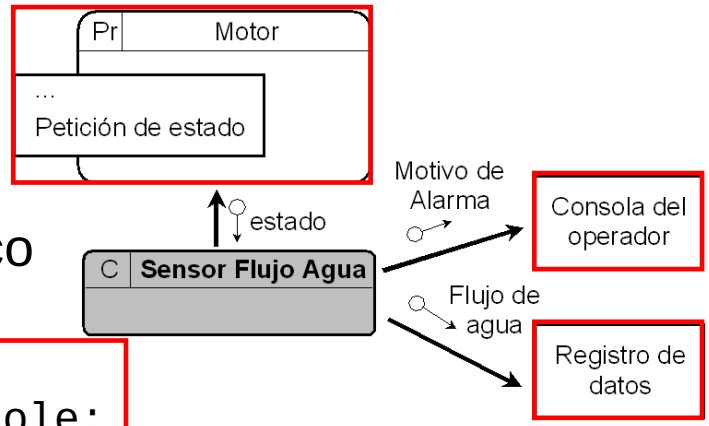

```
package Water_Flow_Sensor is --Ciclico
   pragma Elaborate_Body;
   type Water_Flow is (Yes, No);
end Water_Flow_Sensor;
```

Traducción a Ada

El objeto “Sensor Flujo Agua”

3. Implementación

1. `Initialize`, inicializa el sensor
2. `Periodic_Code`, el código periódico



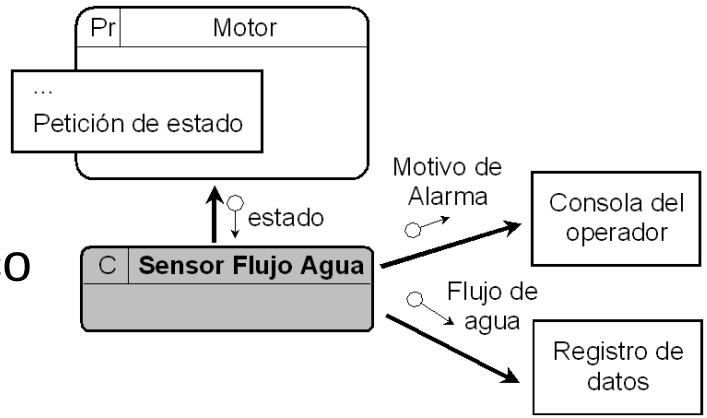

```
with Motor; use Motor;
with Operator_Console; use Operator_Console;
with Data_Logger; use Data_Logger;
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
with Device_Register_Types; use Device_Register_Types;
with Water_Flow_Sensor_Rtatt; use Water_Flow_Sensor_Rtatt;
with System.Storage_Elements; use System.Storage_Elements;

package body Water_Flow_Sensor is
  Flow : Water_Flow := No;
  Current_Pump_Status, Last_Pump_Status : Pump_Status := Off;
  Control_Reg_Addr : constant Address := To_Address(16#AA1b#);
  Wfcsr : Device_Register_Types.Csr;
  for Wfcsr’Address use Control_Reg_Addr;
```

Traducción a Ada

El objeto “Sensor Flujo Agua”

3. Implementación

1. `Initialize`, inicializa el sensor
2. `Periodic_Code`, el código periódico

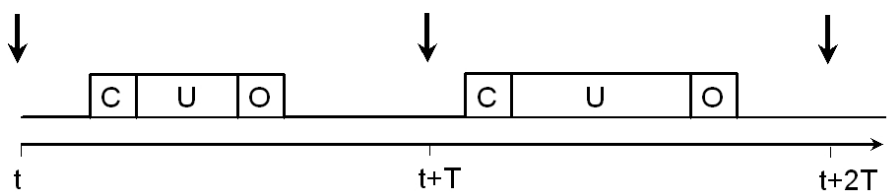



```
procedure Initialize is
begin
   --Enable device
   Wfcsr := D_Enabled;
end Initialize;
```

Traducción a Ada

El objeto “Sensor Flujo Agua”

3. Implementación



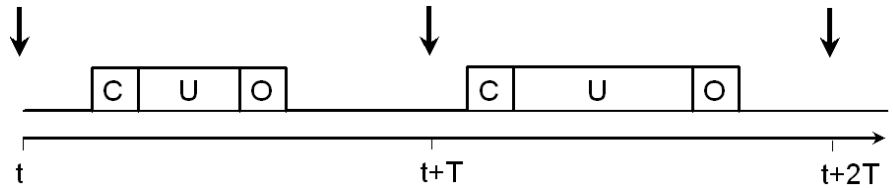

```
procedure Periodic_Code is
begin
  Current_Pump_Status := Motor.Request_Status.Ps;
  if (Wfcsr.Operation = Set) then
    Flow := Yes;
  else
    Flow := No;
  end if;
  if Current_Pump_Status = On and Last_Pump_Status = On
                              and Flow = No then
    Operator_Console.Alarm(Pump_Fault);
  elsif Current_Pump_Status = Off and Last_Pump_Status = Off
                              and Flow = Yes then
    Operator_Console.Alarm(Pump_Fault);
  end if;
  Last_Pump_Status = Current_Pump_Status;
  Data_Logger.Water_Flow_Log(Flow);
end Periodic_Code;
```

Traducción a Ada

El objeto “Sensor Flujo Agua”

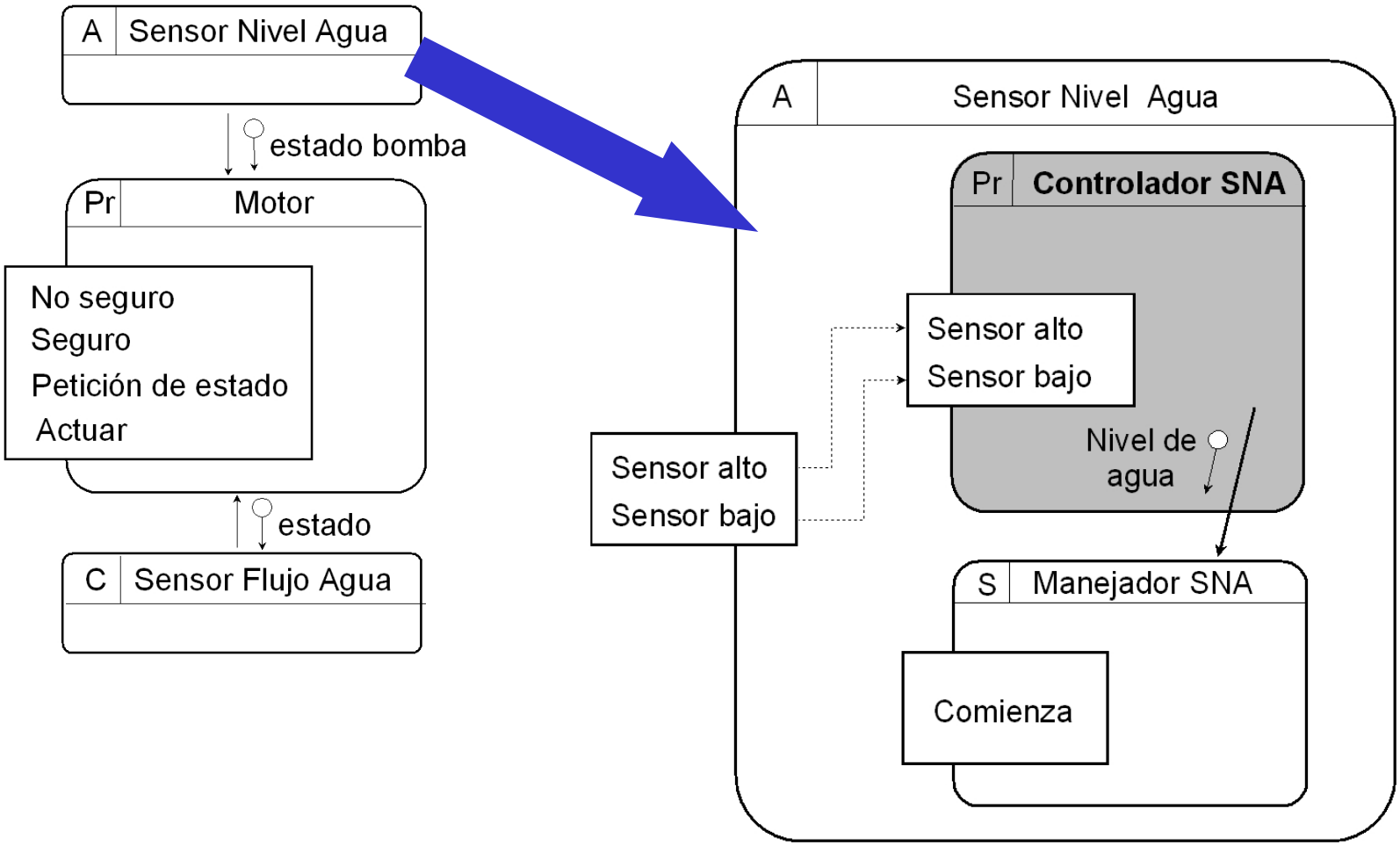
3. Implementación

```
package body Water_Flow_Sensor is
  ...
  procedure Initialize ...
  procedure Periodic_Code ...

  task Thread is
    pragma Priority(Water_Flow_Sensor_Rtatt.Thread_Priority);
  end Thread;

  task body Thread is
    T      : Time;
    Period : Time_Span := Water_Flow_Sensor_Rtatt.Period;
  begin
    T := Clock;
    Initialize;
    loop
      Periodic_Code;
      T := T + Period;
      delay until(T);
    end loop;
  end Thread;
end Water_Flow_Sensor;
```



Traducción a Ada

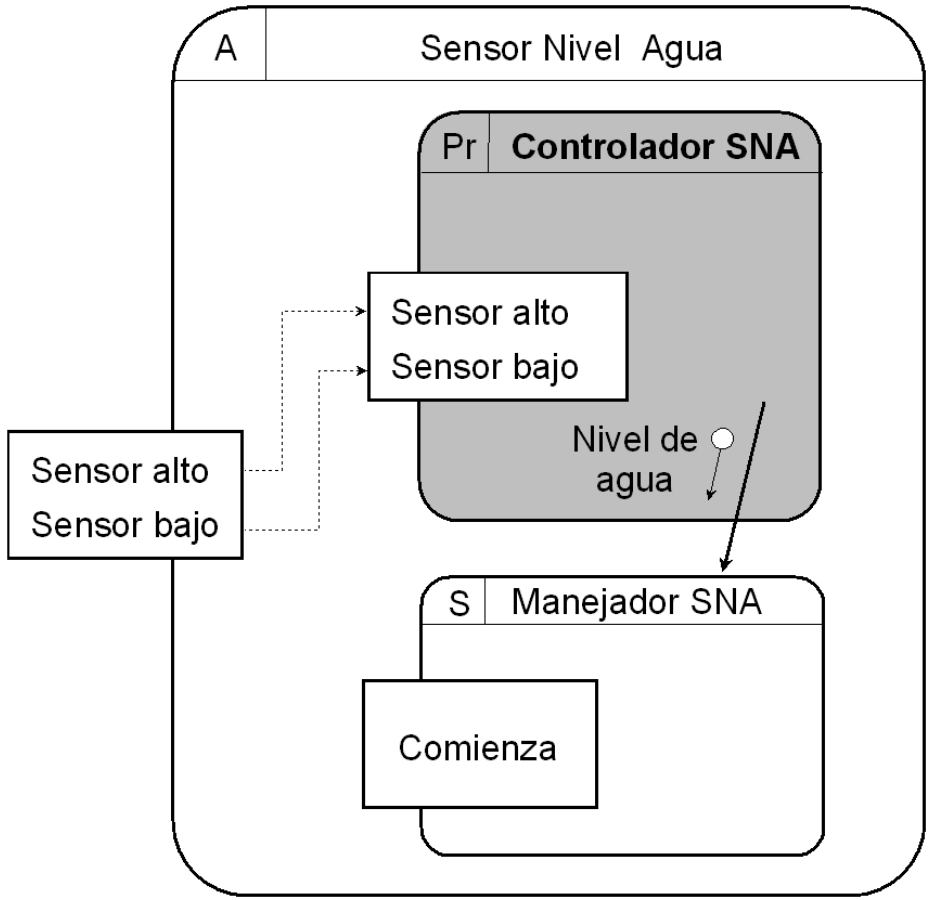
El objeto “Controlador Sensor Nivel Agua (SNA)”



Traducción a Ada

El objeto “Controlador SNA”

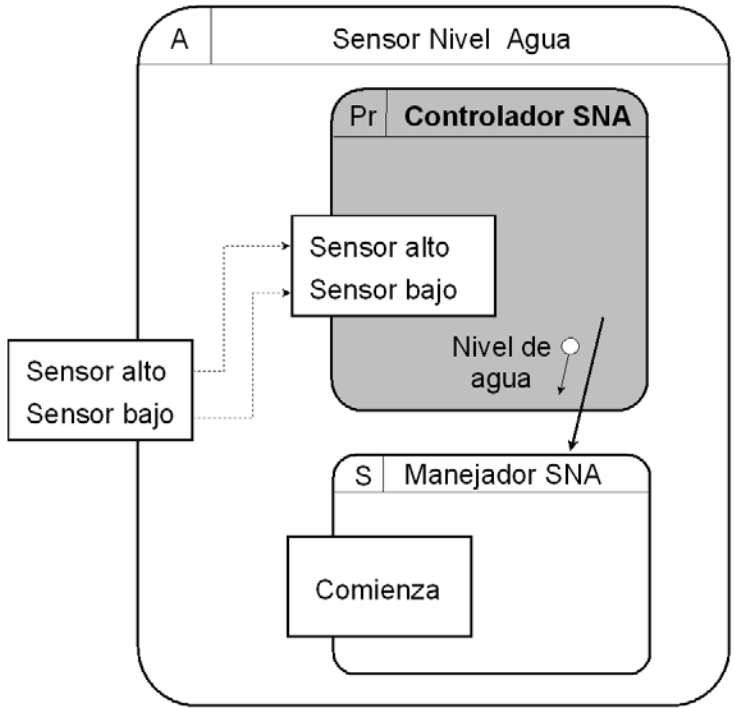
- Es un objeto protegido que encapsula las *rutinas de interrupción* de los sensores de nivel
- HRT_HOOD no permite que un objeto esporádico sea invocado por más de una operación de arranque
- Su objetivo es mapear dos rutinas de interrupción en una única operación "Comienza" del objeto “Manejador SNA” con el parámetro de nivel correspondiente



Traducción a Ada

El objeto “Controlador SNA”

1. Atributos de tiempo real (Objeto protegido que requiere un techo de prioridad)

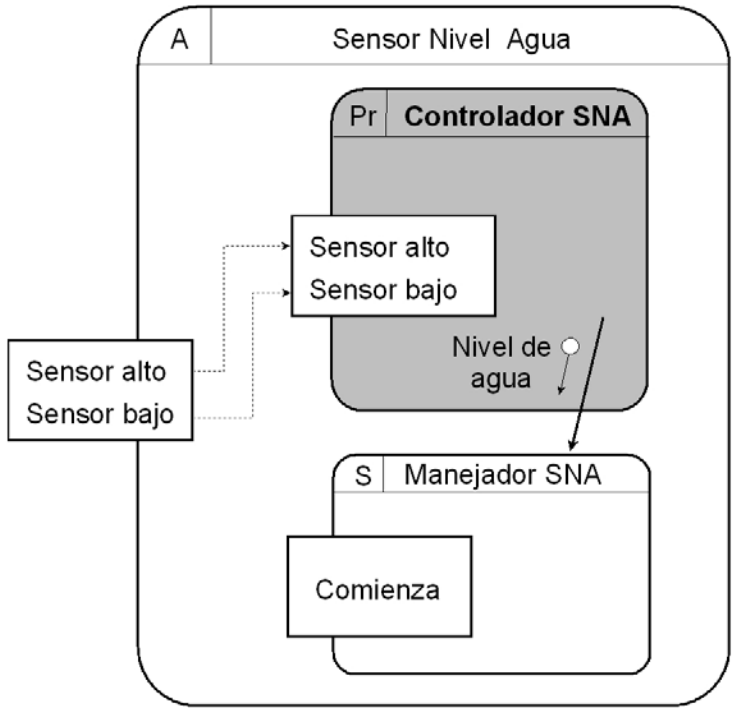



```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time
package Hlw_Controler_Rtatt is
  Ceiling_Priority : constant := 11;
end;
```

Traducción a Ada

El objeto “Controlador SNA”

2. Interfaz

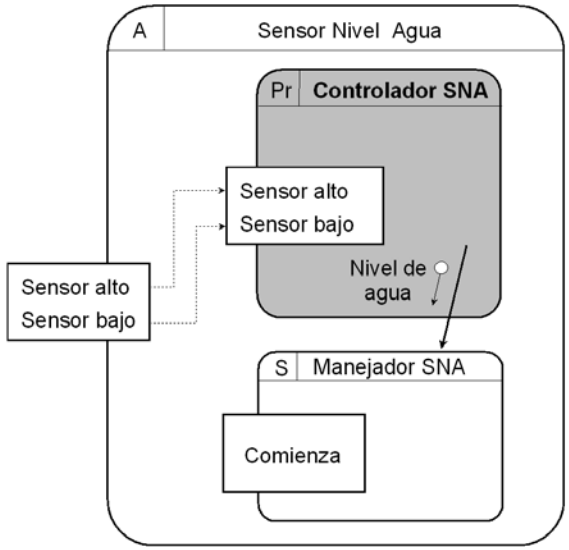



```
with Hlw_Controler_Rtatt;
use  Hlw_Controler_Rtatt;
package Hlw_Controler is  --PROTEGIDO
  procedure Sensor_High_Ih;
  procedure Sensor_Low_Ih;
end Hlw_Controler;
```

Traducción a Ada

El objeto “Controlador SNA”

3. Implementación



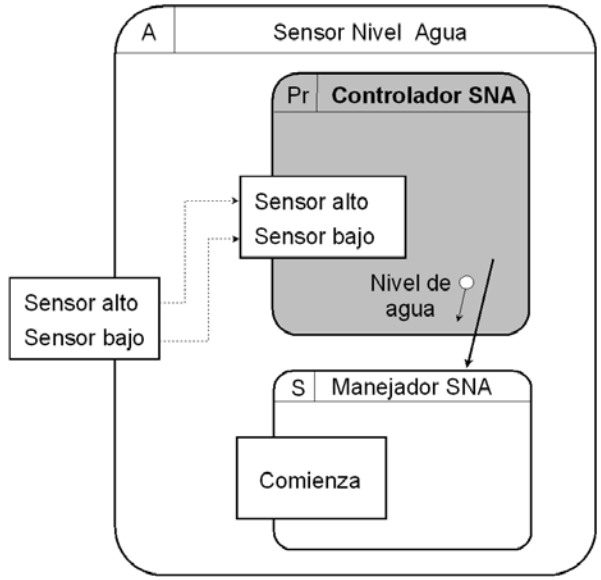

```
with Hlw_Handler; use Hlw_Handler;
with System; use System;
with Ada.Interrupts; use Ada.Interrupts;
with Ada.Interrupts.Names; use Ada.Interrupts.Names;

package body Hlw_Controler is
  protected Agent is
    pragma Priority(Hlw_Controler_Rtatt.Ceiling_Priority);
    procedure Sensor_High_Ih;
    pragma    Attach_Handler(Sensor_High_Ih, Waterh_Interrupt);
    procedure Sensor_Low_Ih;
    pragma    Attach_Handler(Sensor_Low_Ih, Waterl_Interrupt);
  private
  end Agent;
```

Traducción a Ada

El objeto “Controlador SNA”

3. Implementación

```
package body Hlw_Controler is
  protected Agent ... ⇑
  procedure Sensor_High_Ih is begin Agent.Sensor_High_Ih; end;
  procedure Sensor_Low_Ih  is begin Agent.Sensor_Low_Ih;  end;

  protected body Agent is
    procedure Sensor_High_Ih is
    begin
      Hlw_Handler.Start(High);
    end Sensor_High_Ih;
    procedure Sensor_Low_Ih is
    begin
      Hlw_Handler.Start(Low);
    end Sensor_Low_Ih;
  end Agent;
end Hlw_Controler;
```

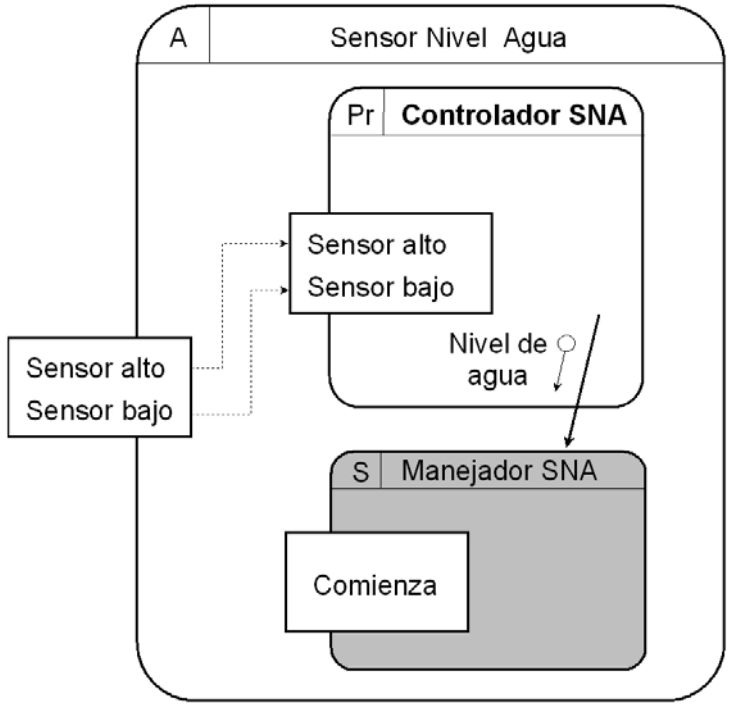


Traducción a Ada

El objeto “Manejador SNA”

Arranca y apaga la bomba en respuesta a la interrupción de nivel

1. Atributos de tiempo real (El objeto protegido *interno* requiere un techo de prioridad y la tarea una prioridad)

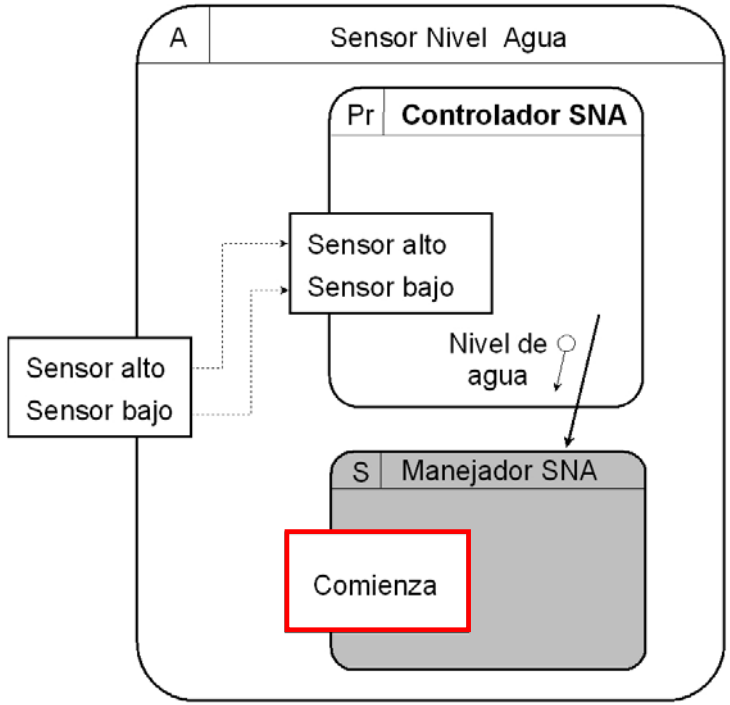



```
with System; use System;
package Hlw_Handler_Rtatt is
  Ceiling_Priority : constant := 11;
  Thread_Priority  : constant Priority := 6;
end Hlw_Handler_Rtatt;
```

Traducción a Ada

El objeto “Manejador SNA”

2. Interfaz

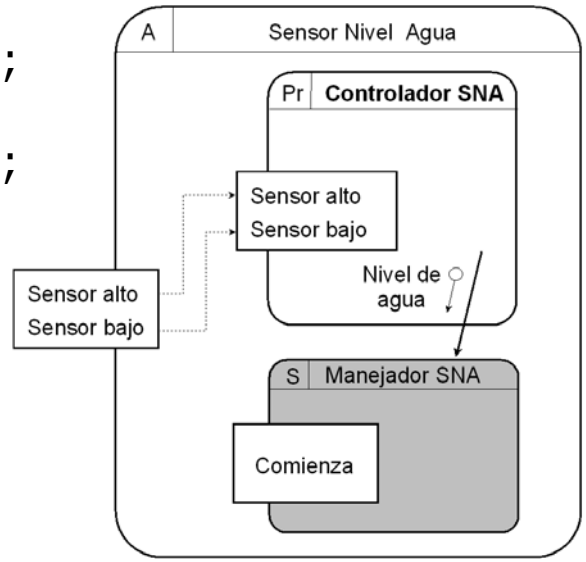



```
with Hlw_Handler_Rtatt;
use  Hlw_Handler_Rtatt;
package Hlw_Handler is  -- ESPORÁDICO
  type Water_Mark is (High, Low);
  procedure Start(Int : Water_Mark);
end Hlw_Handler;
```

Traducción a Ada

El objeto “Manejador SNA” 3. Implementación

```
with Motor;         use Motor;
with Data_Logger;   use Data_Logger;
with Device_Register_Types; use Device_Register_Types;
with System; use System;
with System.Storage_Elements; use System.Storage_Elements;

package body Hlw_Handler is
   Hw_Control_Reg_Addr : constant Address := To_Address(16#AA10#);
   Lw_Control_Reg_Addr : constant Address := To_Address(16#AA12#);
   Hwcsr : Device_Register_Types.Csr;
   for Hwcsr’Address use Hw_Control_Reg_Addr;
   Lwcsr : Device_Register_Types.Csr;
   for Lwcsr’Address use Lw_Control_Reg_Addr;
```

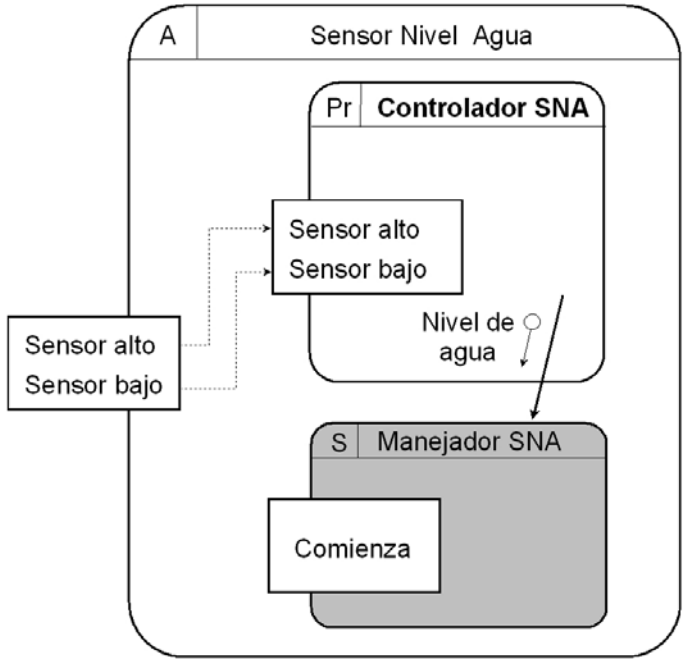


Traducción a Ada

El objeto “Manejador SNA”

3. Implementación

```
package body Hlw_Handler is
  ...
  procedure Sporadic_Code(Int : Water_Mark) is
  begin
    if Int = High then
      Motor.Set_Pump(On);
      Data_Logger.High_Low_Water_Log(High);
      Lwcsr.Interrupt = I_Enable;
      Hwcsr.Interrupt = I_Disable;
    else
      Motor.Set_Pump(Off);
      Data_Logger.High_Low_Water_Log(Low);
      Lwcsr.Interrupt = I_Disable;
      Hwcsr.Interrupt = I_Enable;
    end if;
  end Sporadic_Code;

  procedure Initialize is
  begin
    -- Enable device
    Hwcsr.Device    := D_Enabled;
    Hwcsr.Interrupt := I_Enabled;
    Lwcsr.Device    := D_Enabled;
    Lwcsr.Interrupt := I_Enabled;
  end Initialize;
```

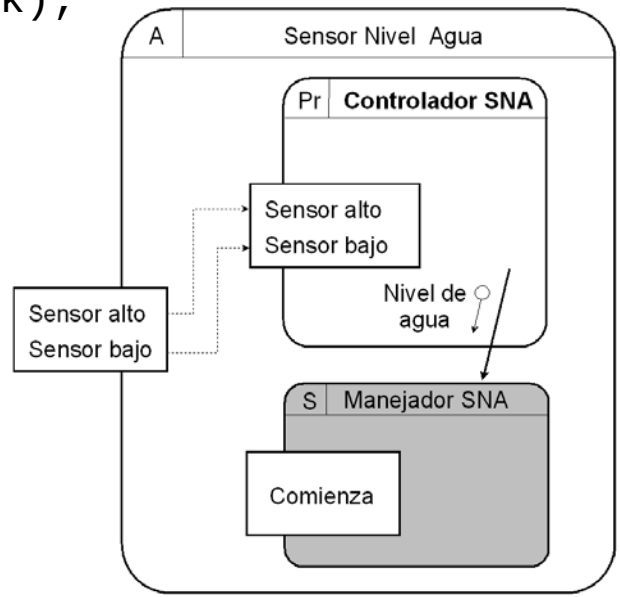


Traducción a Ada

El objeto “Manejador SNA”

3. Implementación

```
package body Hlw_Handler is
  ...
  task Thread is
    pragma Priority(Hlw_Handler_Rtatt.Thread_Priority);
  end Thread;

  protected Agent is
    pragma Priority(Hlw_Handler_Rtatt.Ceiling_Priority);
    procedure Start(Int : Water_Mark);
    entry Wait_Start(Int : out Water_Mark);
  private
    Start_Open : Boolean := False;
    W : Water_Mark;
  end Agent;

  procedure Start(Int : Water_Mark) is
  begin
    Agent.Start(Int);
  end;
```

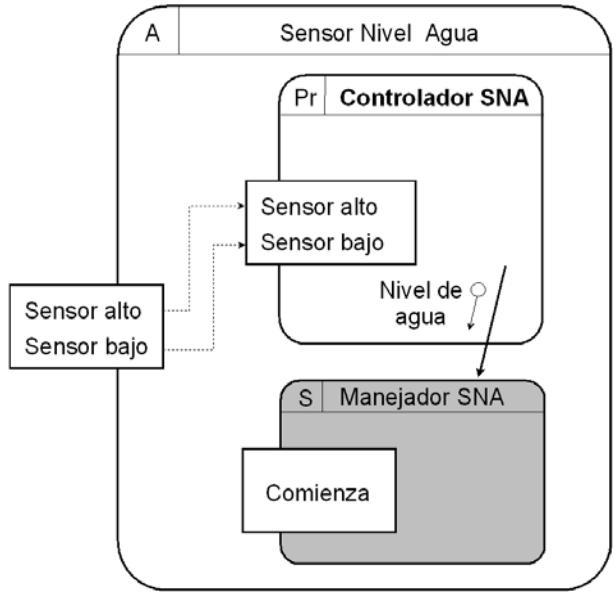


Traducción a Ada

El objeto “Manejador SNA”

3. Implementación

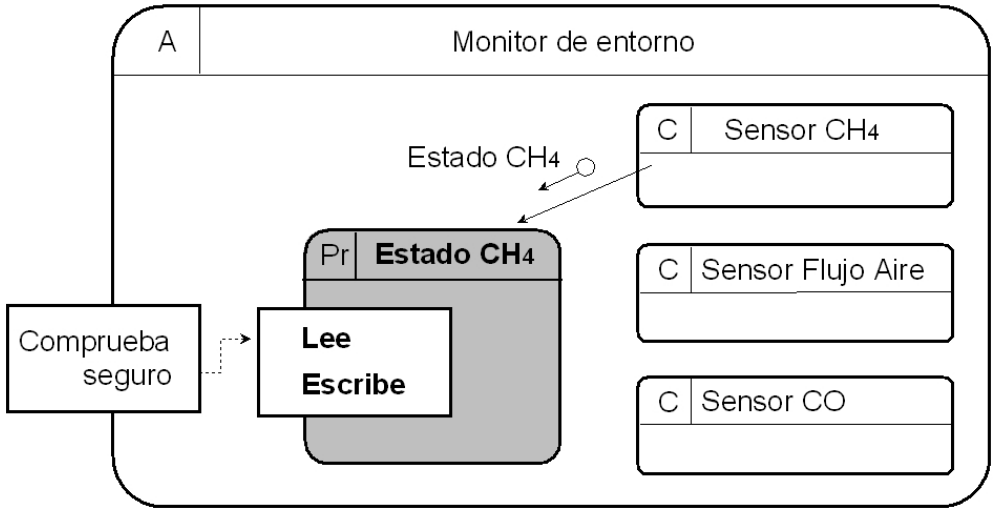
```
  protected body Agent is
    procedure Start(Int : Water_Mark) is
    begin
      W := Int;
      Start_Open := True;
    end Start;
    entry Wait_Start(Int : out Water_Mark) when Start_Open is
    begin
      Int := W;
      Start_Open := False;
    end Wait_Start;
  end Agent;
  task body Thread is
    Int : Water_Mark;
  begin
    Initialize;
    loop
      Agent.Wait_Start(Int);
      Sporadic_Code(Int);
    end loop;
  end Thread;
end Hlw_Handler;
```



Traducción a Ada

El objeto “Estado CH4”

1. Atributos de tiempo real (Objeto protegido que requiere un techo de prioridad)

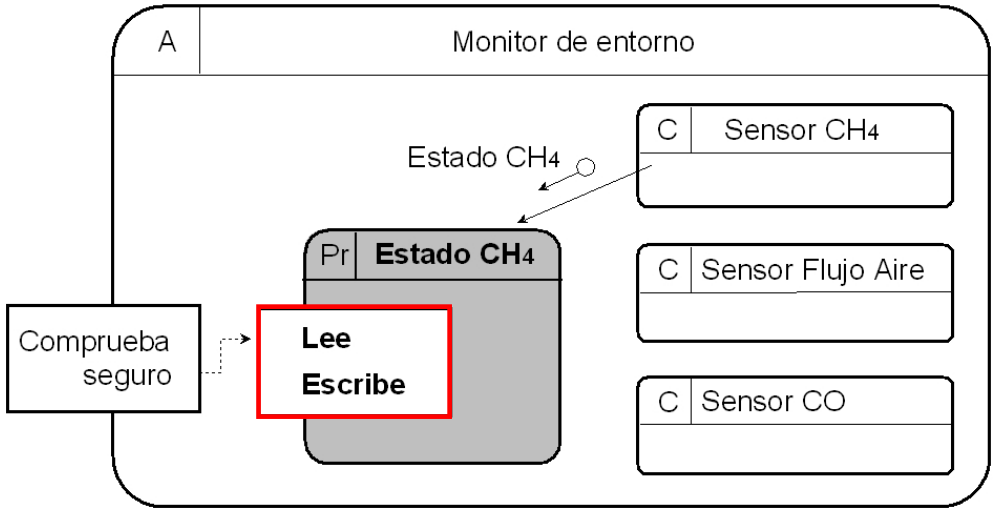



```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
package Ch4_Status_Rtatt is
  Ceiling_Priority : constant := 10;
end;
```

Traducción a Ada

El objeto “Estado CH4”

2. Interfaz




```
package Ch4_Status is
  type Methane_Status is (Motor_Safe, Motor_Unsafe);
  function  Read return Methane_Status;
  procedure Write(Current_Status : Methane_Status);
end Ch4_Status;
```

Traducción a Ada

El objeto "Estado CH4"

3. Implementación

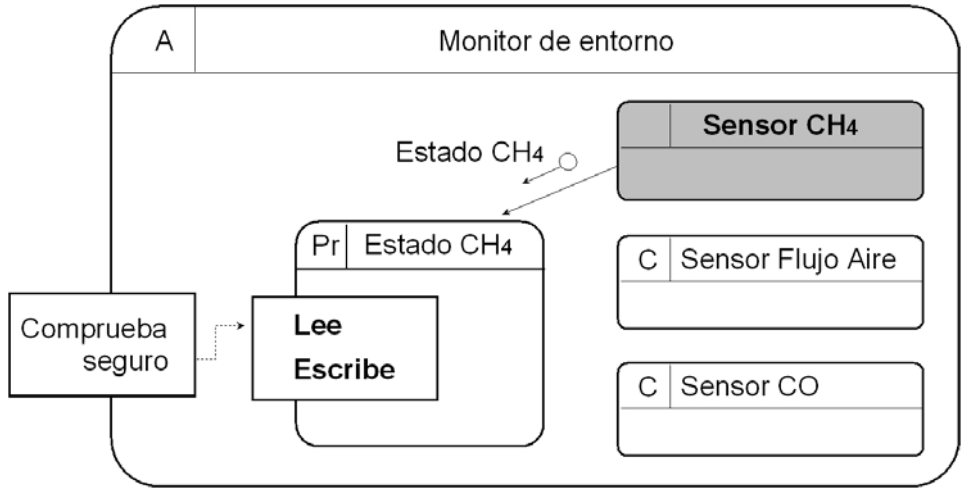
```
with Ch4_Status_Rtatt; use Ch4_Status_Rtatt;
package body Ch4_Status is
  protected Agent is
    pragma    Priority(Ch4_Status_Rtatt.Ceiling_Priority);
    function  Read return Methane_Status;
    procedure Write(Current_Status : Methane_Status);
  private
    Environment_Status : Methane_Status := Motor_Unsafe;
  end Agent;

  function  Read return Methane_S is begin return Agent.Read; end;
  procedure Write is begin Agent.Write; end;

  protected body Agent is
    function Read return Methane_Status is
    begin
      return Environment_Status;
    end;
    procedure Write(Current_Status : Methane_Status) is
    begin
      Environment_Status : Current_Status;
    end;
  end Agent;
end Ch4_Status;
```

Traducción a Ada

El objeto “Sensor CH4”

Se ocupa de medir la concentración del metano en el ambiente

1. Atributos de tiempo real (Objeto cíclico requiere periodo y prioridad)

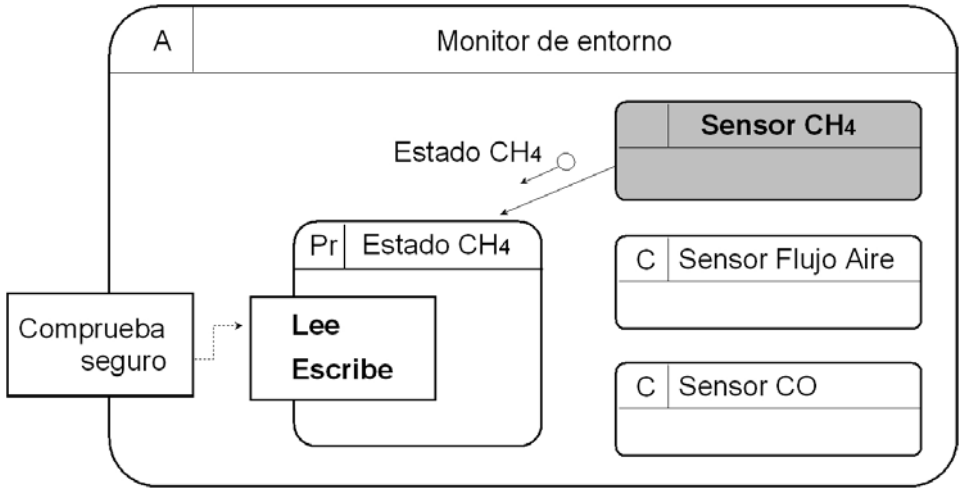



```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
package Ch4_Sensor_Rtatt is
   Period          : Time_Span := Milliseconds(80);
   Thread_Priority : constant Priority := 10;
end Ch4_Sensor_Rtatt;
```

Traducción a Ada

El objeto “Sensor CH4”

2. Interfaz




```
package Ch4_Sensor is  -- Ciclico
  pragma Elaborate_Body;
  type Ch4_Reading is new Integer range 0 .. 1023;
  Ch4_High : constant Ch4_Reading := 400; --Nivel Crítico
                -- Invoca Motor.Is_Safe
                -- Invoca Motor.Not_Safe
                -- Invoca Operator_Console.Alarm
                -- Invoca Data_Logger.Ch4_Log
end Ch4_Sensor;
```

Traducción a Ada

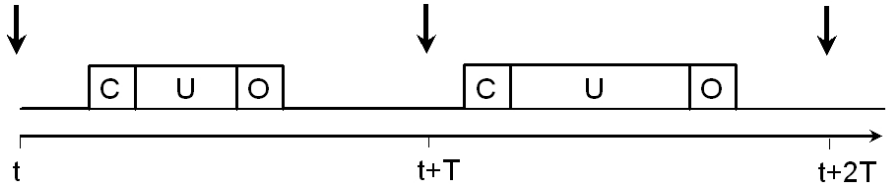
El objeto “Sensor CH4”

3. Implementación

```
with System;                    use System;
with Ada.Real_Time;             use Ada.Real_Time;
with Ch4_Status_Rtatt;          use Ch4_Status_Rtatt;
with Device_Register_Types;     use Device_Register_Types;
with System.Storage_Elements;   use System.Storage_Elements;
with Operator_Console;          use Operator_Console;
with Data_Logger;               use Data_Logger;
with Motor;                     use Motor;
with Ch4_Status;                use Ch4_Status;

package body Ch4_Sensor is
  Ch4_Present : Ch4_Reading;
  Control_Reg_Addr : constant Address := To_Address(16#AA18#);
  Ch4csr : Device_Register_Types.Csr;
  for Ch4csr'Address use Control_Reg_Addr;
  Data_Reg_Addr :    constant Address := To_Address(16#AA1A#);
  Ch4dbr : Ch4_Reading;
  for Ch4dbr'Address use Data_Reg_Addr;
  Jitter_Range : constant Ch4_Reading := 40;
```

Traducción a Ada

```
procedure Initialize is
begin -- Enable device
  Ch4csr.Device    := D_Enabled;
  Ch4csr.Operation := Set;
end Initialize;

procedure Periodic_Code is
begin
  if not Ch4csr.Done then
    Operator_Console.Alarm(Ch4_Device_Error);
  else
    Ch4_Present := Ch4dbr;
    if Ch4_Present > Ch4_High then
      if Ch4_Status.Read = Motor_Safe then
        Motor.Not_Safe;
        Ch4_Status.Write(Motor_Unsafe);
        Operator_Console.Alarm(High_Methane);
      end if;
    elsif (Ch4_Present < (Ch4_High - Jitter_Range))
       and Ch4_Status.Read = Motor_Unsafe; then
      Motor.Is_Safe;
      Ch4_Status.Write(Motor_Safe);
    end if;
    Data_Logger.Ch4_Log(Ch4_Present);
  end if;
  ...
```

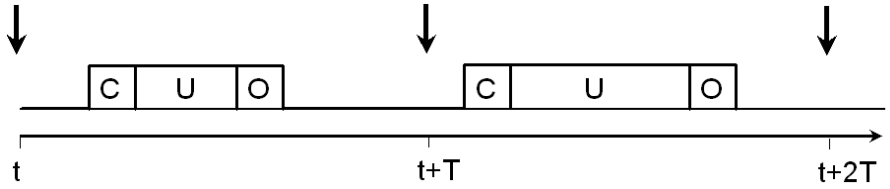


El objeto "Sensor CH4"

3. Implementación

Traducción a Ada

```
package body Ch4_Sensor is
  ... ⇑
  procedure Periodic_Code is
  begin
    ... ⇑
    Ch4csr.Operation := Set;
  end Periodic_Code;
  task Thread is
    pragma Priority(Ch4_Sensor_Rtatt.Thread_Priority);
  end Thread;
  task body Thread is
    T : Time;
    Period : Time_Span := Ch4_Sensor_Rtatt.Period;
  begin
    T := Clock + Period;
    Initialize;
    loop
      delay until(T);
      Periodic_Code;
      T := T + Period;
    end loop;
  end Thread;
end Ch4_Sensor;
```

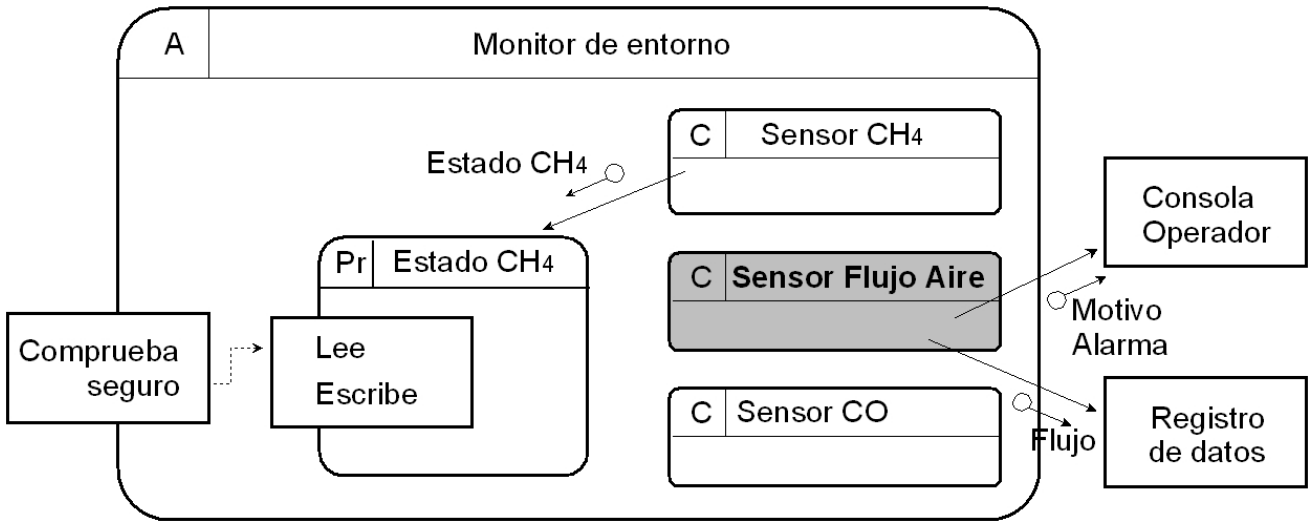


El objeto "Sensor CH4"

3. Implementación

Traducción a Ada

El objeto “Sensor Flujo Aire”



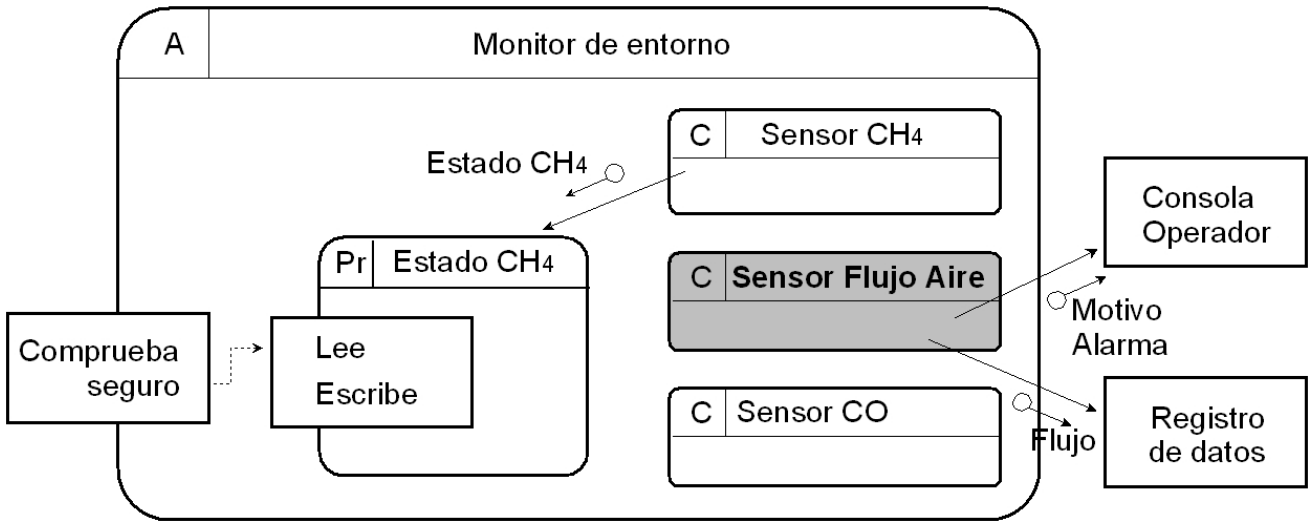
1. Atributos de tiempo real (Objeto cíclico requiere periodo y prioridad)

```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
package Air_Flow_Sensor_Rtatt is
  Period          : Time_Span := Milliseconds(100);
  Thread_Priority : constant Priority := 7;
end Air_Flow_Sensor_Rtatt;
```

Traducción a Ada

El objeto “Sensor Flujo Aire”

2. Interfaz

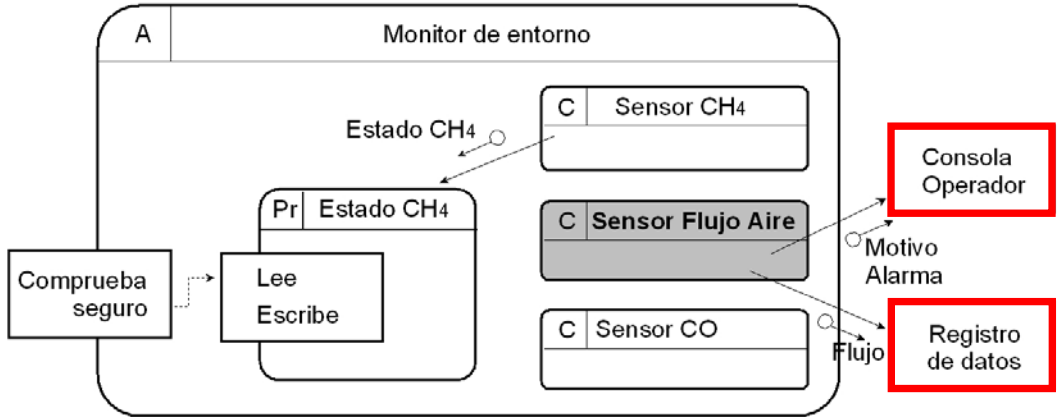



```
package Air_Flow_Sensor is  -- Ciclico
  pragma Elaborate_Body;
  type Air_Flow_Status is (Air_Flow, No_Air_Flow);
                -- Invoca Operator_Console.Alarm
                -- Invoca Data_Logger.Air_Flow_Log
end Air_Flow_Sensor;
```

Traducción a Ada

El objeto “Sensor Flujo Aire”

3. Implementación

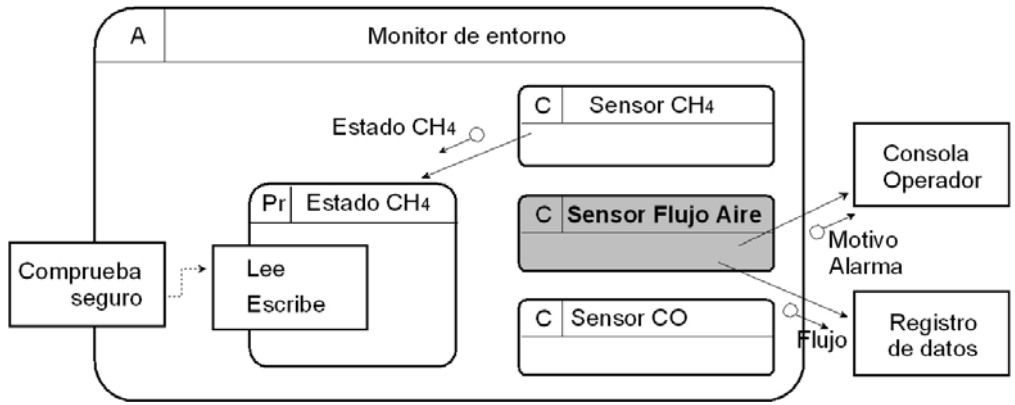



```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
with Air_Flow-_Sensor_Rtatt; use Air_Flow-_Sensor_Rtatt;
with Device_Register_Types; use Device_Register_Types;
with System.Storage_Elements; use System.Storage_Elements;
with Operator_Console; use Operator_Console;
with Data_Logger; use Data_Logger;
package body Air_Flow_Sensor is
  Air_Flow_Reading : Boolean;
  Control_Reg_Addr : constant Address := To_Address(16#AA20#);
  Afcsr : Device_Register_Types.Csr;
  for Afcsr’Address use Control_Reg_Addr;
```

Traducción a Ada

El objeto “Sensor Flujo Aire” 3. Implementación

```
package body Air_Flow_Sensor is
  ...
  task Thread is
    pragma Priority(Air_Flow_Sensor_Rtatt.Initial_Thread_Priority);
  end Thread;

  procedure Initialize is
  begin
    -- Enable device
    Afcsr.Device := D_Enabled;
  end Initialize;
  ...
```

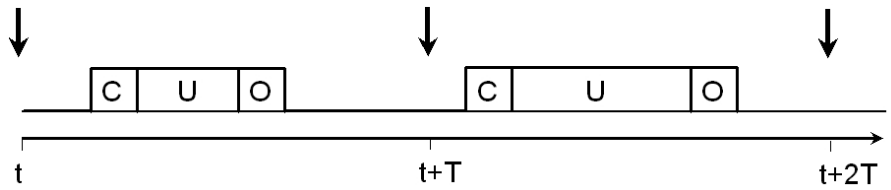


Traducción a Ada

```
...
procedure Periodic_Code is
begin
  --Cuando hay flujo de aire, el bit operation está a 1, si no, a 0
  Air_Flow_Reading := Afcsr.Operation = Set;
  if not Air_Flow_Reading then
    Operator_Console.Alarm(No_Air_Flow);
    Data_Logger.Air_Flow_Log(No_Air_Flow);
  else
    Data_Logger.Air_Flow_Log(Air_Flow);
  end if;
end;
task body Thread is
  T : Time;
  Period : Time_Span := Air_Flow_Sensor_Rtatt.Period;
begin
  T := Clock + Period;
  Initialize;
  loop
    delay until(T);
    Periodic_Code;
    T := T + Period;
  end loop;
end Thread;
end Air_Flow_Sensor;
```

El objeto "Sensor Flujo Aire"

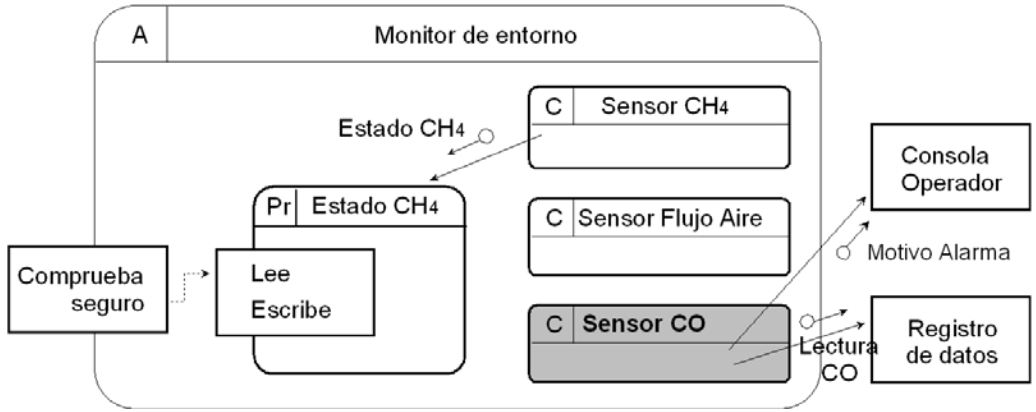
3. Implementación



Traducción a Ada

El objeto “Sensor CO”

1. Atributos de tiempo real (El objeto cíclico requiere periodo y prioridad)

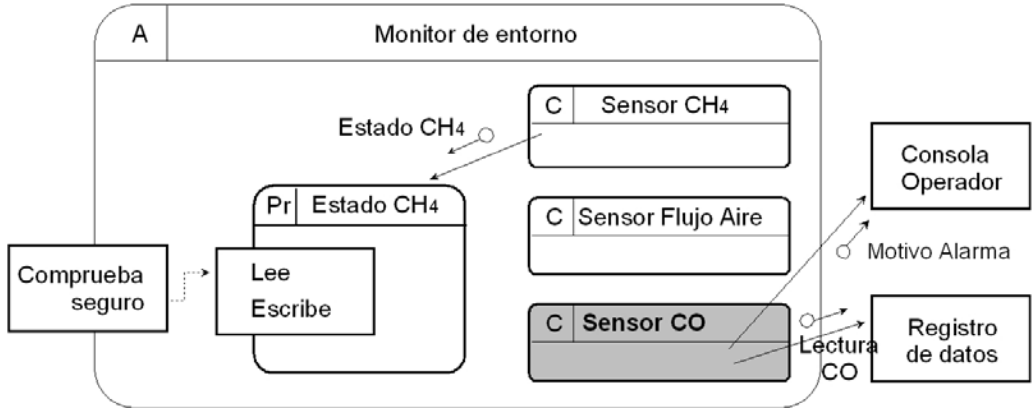



```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
package Co_Sensor_Rtatt is
   Period          : Time_Span := Milliseconds(1000);
   Thread_Priority : constant Priority := 8;
end Co_Sensor_Rtatt;
```

Traducción a Ada

El objeto “Sensor CO”

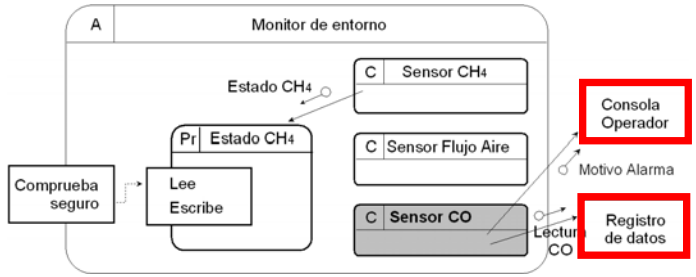
2. Interfaz




```
package Co_Sensor is  -- Ciclico
  pragma Elaborate_Body;
  type Co_Reading is new Integer range 0 .. 1023;
  Co_High : constant Co_Reading := 600;
                -- Invoca Operator_Console.Alarm
                -- Invoca Data_Logger.Co_Log
end Co_Sensor;
```

Traducción a Ada

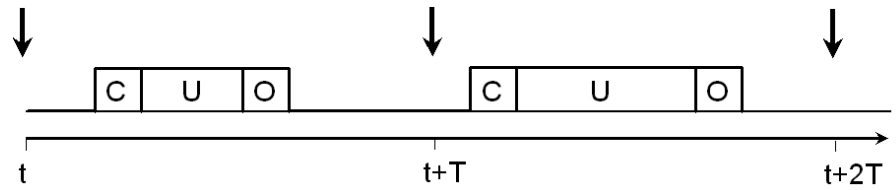
El objeto “Sensor CO”



3. Implementación

```
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;
with Co_Status_Rtatt; use Co_Status_Rtatt;
with Device_Register_Types; use Device_Register_Types;
with System.Storage_Elements; use System.Storage_Elements;
with Operator_Console; use Operator_Console;
with Data_Logger; use Data_Logger;
package body Co_Sensor is
  Co_Present : Co_Reading;
  -- Registro de Control
  Control_Reg_Addr : constant Address := To_Address(16#AA1C#);
  Cocsr : Device_Register_Types.Csr;
  for Cocsr'Address use Control_Reg_Addr;
  -- Registro de Datos
  Data_Reg_Addr : constant Address := To_Address(16#AA1E#);
  Codbr : Co_Reading;
  for Codbr'Address use Data_Reg_Addr;
```

Traducción a Ada

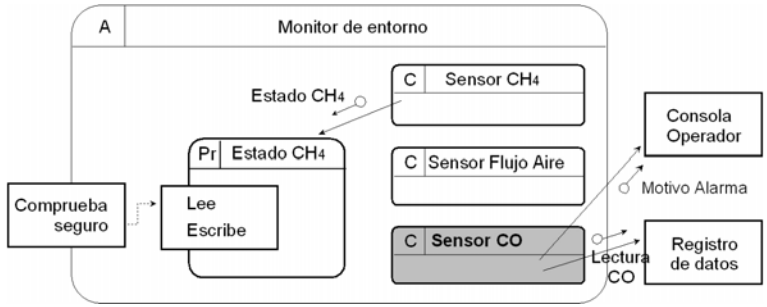


El objeto "Sensor CO"

3. Implementación

```
package body Co_Sensor is
  ... ⇑
  procedure Initialize is
  begin
    -- Enable device
    Cocsr.Device    := D_Enabled;
    Cocsr.Operation := Set;
  end Initialize;

  procedure Periodic_Code is
  begin
    if not Cocsr.Done then
      Operator_Console.Alarm(Co_Device_Error);
    else
      Co_Present := Codbr; -- Lectura de la conversión
      if Co_Present > Co_High then
        Operator_Console.Alarm(High_Co);
      end if;
      Data_Logger.Co_Log(Co_Present);
    end if;
    Cocsr.Operation := Set;  -- Comienza la conversión
  end Periodic_Code;
```

Traducción a Ada




```
package body Co_Sensor is

  ... ⇑
  task Thread is
    pragma Priority(Co_Sensor_Rtatt.Thread_Priority);
  end Thread;

  task body Thread is
    T : Time;
    Period : Time_Span := Co_Sensor_Rtatt.Period;
  begin
    T := Clock + Period;
    Initialize;
    loop
      delay until(T);
      Periodic_Code;
      T := T + Period;
    end loop;
  end Thread;
end Co_Sensor;
```

El objeto "Sensor CO"

3. Implementación

Traducción a Ada

El objeto “Registro de datos”

- No lo tratamos en detalle.
- Existe el requisito de que sólo puede retrasar las tareas (de tiempo real) durante un tiempo limitado.
- Por tanto, su interfaz debe contener objetos protegidos

```
with Co_Sensor; use Co_Sensor;
with Ch4_Sensor; use Ch4_Sensor;
with Air_Flow_Sensor; use Air_Flow_Sensor;
with Hlw_Handler; use Hlw_Handler;
with Water_Flow_Sensor; use Water_Flow_Sensor;
with Motor; use Motor;
package Data_Logger is --ACTIVO
  procedure Co_Log             (Reading : Co_Reading);
  procedure Ch4_Log            (Reading : Ch4_Reading);
  procedure Air_Flow_Log       (Reading : Air_Flow_Status);
  procedure High_Low_Water_Log(Mark    : Water_Mark);
  procedure Water_Flow_Log     (Reading : Water_Flow);
  procedure Motor_Log          (State   : Motor_State_Changes);
end Data_Logger;
```

Traducción a Ada

El objeto “Consola Operador”

- No lo tratamos en detalle.
- Existe el requisito de que sólo puede retrasar las tareas (de tiempo real) durante un tiempo limitado.
- Por tanto, su interfaz debe contener objetos protegidos

```
package Operator_Console is --ACTIVO
  type Alarm_Reason is (High_Metane, High_Co, No_Air_Flow,
                        Ch4_Device_Error, Co_Device_Error,
                        Pump_Fault, Unknown_Error);
  procedure Alarm(Reason  : Alarm_Reason,
                  Name    : String := “Unknown”,
                  Details : String := “”);
                            -- Invoca Motor.Request_Status
                            -- Invoca Motor.Set_Pump
end Operator_Console;
```

Traducción a Ada

Errores

- Si suponemos que todos los errores generan una excepción, cada tarea puede ser protegida con un manajedor "`when others`":

```
task body Thread is
  T : Time;
  Period : Time_Span := Water_Flow_Sensor_Rtatt.Period;
begin
  T := Clock;
  Initialize;
  loop
    Periodic_Code;
    T := T + Period;
    delay until(T);
  end loop;
exception
  when E: others =>
    Motor.Not_Safe;
    Ch4_Status.Write(Motor_Unsafe);
    Operator_Console.Alarm(Unknown_Error, Exception_Name(E),
                           Exception_Information(E));
end Thread;
end Water_Flow_Sensor;
```

Tareas esporádicas en POSIX/C

sporadic.h

```
#include<pthread.h>
#define SPORADIC_PARM_SIZE 256
#define PARAM_IN  0
#define PARAM_OUT 1
struct rtattr {
  int  Priority;
  int  Ceiling;
};
typedef rtattr rtattr;
typedef void *sporadic_param;
typedef void (*sporadic_frame)(sporadic_param param);
typedef void (*initialise)(void);

struct sporadic_event {
  rtatt               Rtattr;
  pthread_mutexattr_t Mattr;
  pthread_condattr_t  Cattr;
  pthread_mutex_t     Mutex;
  pthread_cond_t      Cond;
  int                 Arrived, Sense;
  char                Param[SPORADIC_PARM_SIZE];
  sporadic_frame      Frame;
  initialise          Initialise;
};
typedef struct sporadic_event sporadic_event;
```

Tareas esporádicas en POSIX/C

sporadic.h

```
void sporadic_init(sporadic_event *event,
                                    sporadic_frame frame,
                                    initialise     init,
                                    int            ceiling,
                                    int            prio,
                                    int            sense);

int  sporadic_start(sporadic_event *event, void *dato);
```

Tareas esporádicas en POSIX/C

sporadic.c

```
#include <sporadic.h>
#include <pthread.h>
#include <sched.h>

static void event_init(sporadic_event *event, sporadic_frame frame,
                                              initialise     init,
                                              int            ceiling,
                                              int            prio,
                                              int            sense) {

  pthread_attr_t     attr;
  pthread_t          thr;
  struct sched_param param;

  event->Rtattr.Ceiling  = ceiling;
  event->Rtattr.Priority = prio;
  event->Frame           = frame;
  event->Initialise      = init;
  event->Sense           = sense;
  event->Arrived         = 0;

  pthread_mutexattr_init          (&event->Mattr);
  pthread_mutexattr_setprotocol   (&event->Mattr, PTHREAD_PRIO_PROTECT);
  pthread_mutexattr_setprioceiling(&event->Mattr, event->Rtattr.Ceiling);
  pthread_mutex_init              (&event->Mutex, &event.Mattr);
  pthread_condattr_init           (&event->Cattr);
  pthread_cond_init               (&event->Cond,  &event->Cattr);
}
```

Tareas esporádicas en POSIX/C

sporadic.c

```
static void sporadic_body(sporadic_event *event)
{
  ev_param dato;

  event->Initialise();  /* Inicializacion */
  while(1) {
    pthread_mutex_lock(event->mutex);
    while(!event->Arrived)
      pthread_cond_wait(event->Cond, event->Mutex);
    dato = event->Ev_data;
    event->Arrived = 0;
    pthread_mutex_unlock(event->mutex);
    event->Frame(dato);  /* Marco esporadico */
  }
}
```

Tareas esporádicas en POSIX/C

sporadic.c

```
void sporadic_init(sporadic_event *event, sporadic_frame frame,
                                          initialise     init,
                                          int            ceiling,
                                          int            prio,
                                          int            sense)
{
  pthread_attr_t     attr;
  pthread_t          thr;
  struct sched_param param;

  event_init(event, frame, init, ceiling, prio, sense);

  pthread_attr_init           (&attr);
  pthread_attr_setscope       (&attr, PTHREAD_SCOPE_SYSTEM);
  pthread_attr_setschedpolicy (&attr, SCHED_FIFO);
  param.sched_priority = prio;
  pthread_attr_setschedparam  (&attr, &param);
  pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
  pthread_create(&thr, &attr, (void *)sporadic_body, (void *)event);
}
```

Tareas esporádicas en POSIX/C

sporadic.c

```
int sporadic_start(sporadic_event *event, void *dato)
{
  pthread_mutex_lock  (event->Mutex);
  event.Arrived = 1;
  if(event->sense == INPUT)
    memcpy(event.Param, dato, SPORADIC_PARM_SIZE);
  else
    memcpy(dato, event.Param, SPORADIC_PARM_SIZE);
  pthread_cond_signal (event->Cond);
  pthread_mutex_unlock(event->Mutex);
  return 0;
}
```

Tareas esporádicas en POSIX/C

mod_rtattr.h

```
#define MOD_SNA_CEILING         1
#define MOD_SNA_PRIORITY        9
```

Tareas esporádicas en POSIX/C

mod.h

```
typedef ... MOD_param_type;
extern void MOD_init();
extern int  MOD_start(MOD_param_type *dato);
```

Tareas esporádicas en POSIX/C

mod.c

```
#include <mod_rtattr.h>
#include <mod.h>
#include <sporadic.h>

static sporadic_event event;

static void initialise(void)
{
   ...
}

static void sporadic_frame(MOD_param_type *dato)
{
   ...
}

void MOD_init()
{
  sporadic_init(&event, sporadic_frame, initialise,
                MOD_CEILING_PRIORITY, MOD_PRIORITY, INPUT);
  /* Código de iniciicialización dependiente del objeto */
}

int MOD_start(MOD_param_type *dato)
{
  sporadic_start(&event, (void *)dato);
}
```

Tareas periódicas en POSIX/C

periodic.h

```
struct rtattr {
  int            Priority;
  struc timespec Period;
};
typedef rtattr rtattr;
typedef void (*periodic_frame)(void);
typedef void (*initialise)    (void);

struct periodic_event {
  rtatt          Rtattr;
  periodic_frame  Frame;
  initialise      Initialise;
};
typedef struct periodic_event periodic_event;

void periodic_init (periodic_event *event, periodic_frame frame,
                                           initialise     init,
                                           int            period_s,
                                           int            period_ns,
                                           int            prio);
```

Tareas periódicas en POSIX/C

periodic.c

```
#include <periodic.h>
#include <pthread.h>
#include <sched.h>
#include <signal.h>

static void event_init(periodic_event *event, periodic_frame   frame,
                                              initialise       init,
                                              struct timespec *period,
                                              int              prio)
{
  event.Rtattr.Period    = *period;
  event.Rtattr.Prioriy   =  prio;
  event.Frame            =  frame;
  event.Initialise       =  init;
}
```

Tareas periódicas en POSIX/C

periodic.c

```
static void periodic_Body(periodic_event *event)
{
  int               signum;
  sigset_t          set;
  struct sigevent   sig;
  timer_t           timer;
  struct itimerspec required, old;
  struct timespec   first, period;

  sig.sigev_notify = SIGEV_SIGNAL;
  sig.sigev_signo  = SIGRTMIN;
  if(0 > clock_gettime (CLOCK_REALTIME, &first))   error();
  first.tv_sec   = first.tv_sec + 1;
  required.it_value    = first;
  required.it_interval = event->Period;
  if(timer_create(CLOCK_REALTIME, &sig, &timer))   error();
  if(sigemptyset(&set))                            error();
  if(sigaddset(&set, SIGRTMIN))                    error();
  if(timer_settime(timer, 0, &required, &old))     error();
  event->Initialise();  /* Inicializacion */
  while(1) {
    if (sigwait(&set, &signum))                    error();
    event->Frame();  /* Marco periodico */
  }
}
```

Tareas periódicas en POSIX/C

periodic.c

```
void periodic_init(periodic_event *event, sporadic_frame frame,
                                          initialise     init,
                                          int            period_s,
                                          int            period_ns,
                                          int            prio)
{
  pthread_attr_t     attr;
  pthread_t          thr;
  struct sched_param param;
  struct timespec    period = {period_s, period_ns},

  event_init(event, frame, ceiling, prio);

  pthread_attr_init          (&attr);
  pthread_attr_setscope      (&attr, PTHREAD_SCOPE_SYSTEM);
  pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
  param.sched_priority = prio;
  pthread_attr_setschedparam(&attr, &param);
  pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
  pthread_create(&thr, &attr, (void *)sporadic_body, (void *)event);
}
```

Tareas periódicas en POSIX/C

mod_rtattr.h

```
#define MOD_PERIOD_SECONDS        0  /* 10 ms */
#define MOD_PERIOD_NANOSECONDS    10000000
#define MOD_PRIORITY              9
```

Tareas periódicas en POSIX/C

mod.h

```
void MOD_init(void);
```

Tareas periódicas en POSIX/C

mod.c

```
#include <MOD_rtattr.h>
#include <periodic.h>

static periodic_event event;

static void initialise(void)
{
   /* Código de negocio */
}
static void periodic_frame(void)
{
   /* Código de negocio */
}

void MOD_init()
{
  periodic_init(&event, periodic_frame,
                        initialise,
                        MOD_PERIOD_SECOND,
                        MOD_PERIOD_NSECOND,
                        MOD_PRIORITY);
  /* Código de negocio */
}
```

Protocolos de prioridad en POSIX/C

- Las regiones críticas (recursos) se implementan usando mutexes
- Para prevenir la inversión de prioridad en el acceso a recursos, POSIX asocia protocolos de herencia/techo de prioridad al mutex:

```
int pthread_mutexattr_setprotocol (
                    pthread_mutex_attr_t *attr,
                    int                   protocol);
```

- El protocolo `protocol` puede ser

```
PTHREAD_PRIO_INHERIT (Herencia de prioridad)
PTHREAD_PRIO_PROTECT (Techo de prioridad inmediato)
PTHREAD_PRIO_NONE    (No hay herencia de prioridad)
```

- El techo de prioridad del mutex se establece

```
int pthread_mutexattr_setprioceiling (
                    pthread_mutexattr_t *attr,
                    int                  prioceiling);
```

Protocolos de prioridad en POSIX/C

Ejemplo

```
pthread_mutex_t     m;
pthread_mutexattr_t m_attr;
int                 protocol, high_prio;

pthread_mutexattr_init(&m_attr);
pthread_mutexattr_getprotocol(&m_attr, &protocol);
pthread_mutexattr_setprotocol(&m_attr,
                              PTHREAD_PRIO_PROTECT);
pthread_mutexattr_setpriorityceiling(&m_attr,
                                     high_prio);
pthread_mutex_init(&m, &m_attr);
```

Traducción a POSIX/C

- HRT-HOOD soporta una traducción sistemática a POSIX/C
- C no tiene una estructura de módulo formal equivalente al paquete Ada. Por lo tanto un objeto HRT-HOOD se mapea a dos ficheros C diferentes, un ".h" para la interfaz del objeto y un ".c" para su implementación. A ambos los denominaremos conjuntamente como un **módulo C**
- Una operación de un objeto HRT-HOOD se mapeará a una función C del modulo que implementa el objeto
- Desafortunadamente, un fichero C no introduce un espacio de nombres separado. Por lo tanto, todas las operaciones las denominaremos `MODULO_operacion`.

Traducción a POSIX/C

- Para cada objeto terminal se generan **dos** módulos C:
  1. Tipos de datos y variables que definen los atributos de tiempo real del objeto (sólo “.h”)
  2. El código del objeto (“.h” y “.c”)

Traducción a POSIX/C

El objeto "Motor" Fichero *Device_Register_Types.h*

- Los registros del dispositivo que controla el motor están declarados en el módulo `Device_Register_Types`:

No escrito

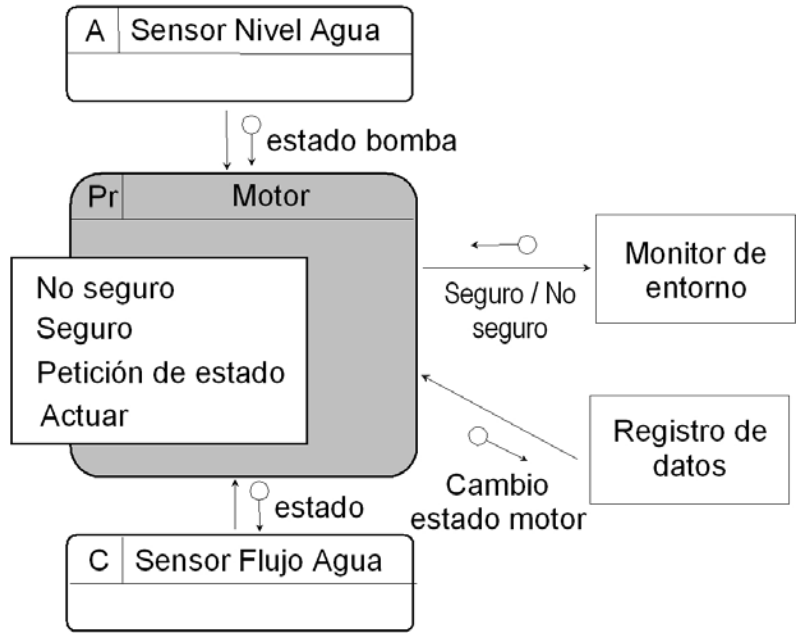
Traducción a POSIX/C

El objeto "Motor"

1. **Atributos de tiempo real** (Mutex que requiere un techo de prioridad)

Fichero *motor_rtatt.h*

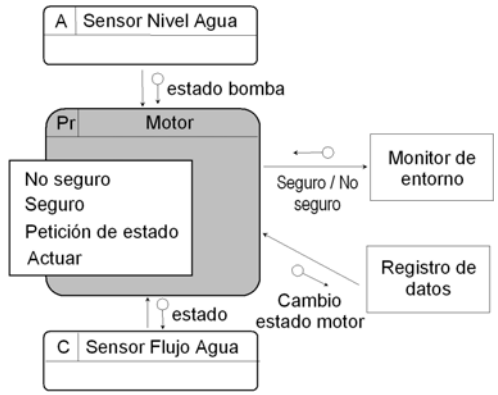
```
#define MOTOR_CEILING_PRIORITY 10
```



Traducción a POSIX/C

El objeto “Motor” 2. Interfaz Fichero *motor.h*

```
/* PROTECTED */
#define PUMP_NOT_SAFE (-1) /* Set_Pump exception */

enum Pump_Status {On = 0; Off = 1};
typedef enum Pump_Status Pump_Status;

enum Pump_Condition {Enable = 0; Disable = 1};
typedef enum Pump_Condition Pump_Condition;

enum Motor_State_Changes {Motor_Started = 0; Motor_Stopped = 1;
                          Motor_Safe= 2;     Motor_Unsafe= 3;   };
typedef enum Motor_State_Changes Motor_State_Changes;

struct Operational_Status {
  Pump_Status    Ps;
  Pump_Condition Pc;
};
typedef struct Operational_Status Operational_Status;

extern void MOTOR_init          (void);
extern void MOTOR_not_Safe      (void);
extern void MOTOR_is_Safe       (void);
extern void MOTOR_rquest_Status(Operational_Status *opStatus);
extern int  MOTOR_set_Pump      (Pump_Staus          To);
```



Traducción a POSIX/C

El objeto “Motor” 3. Implementación Fichero *motor.c*

```
#include "Motor.h"
#include "Motor_Rtatt.h"
#include <data_Logger.h>
#include <ch4_estado.h>
#include <pthread.h>

static int             motor_Status;
static int             motor_Condition;
static pthread_mutex_t mutex;

void MOTOR_init(void)
{
  pthread_mutexattr_t m_attr;

  motor_Status    = Off;
  motor_Condition = Disabled;

  pthread_mutexattr_init(&m_attr);
  pthread_mutexattr_setprotocol(&m_attr, PTHREAD_PRIO_PROTECT);
  pthread_mutexattr_setpriorityceiling(&m_attr, MOTOR_CEILING_PRIORITY);
  pthread_mutex_init(&mutex, &m_attr);
}
```

Traducción a POSIX/C

El objeto "Motor"

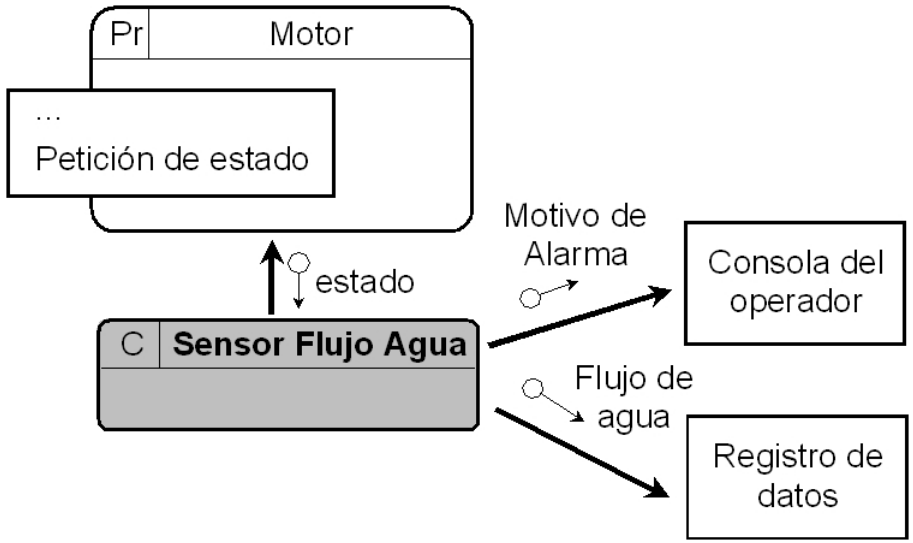
Fichero *motor.c*

```
void MOTOR_not_Safe (void)
{
  pthread_mutex_lock(&mutex);
  if(motor_Status == On) {
    /* Apaga motor (No implementado) */
    DATALOGER_motor_Log(Motor_Stopped);
  }
  motor_Condition := Disabled;
  DATALOGER_motor_Log(Motor_Unsafe);
  pthread_mutex_unlock(&mutex);
}

...
```

Traducción a POSIX/C

El objeto “Sensor Flujo Agua”

1. **Atributos de tiempo real**
   (Objeto cíclico requiere periodo y prioridad)



Fichero *flujo_agua_rtatt.h*

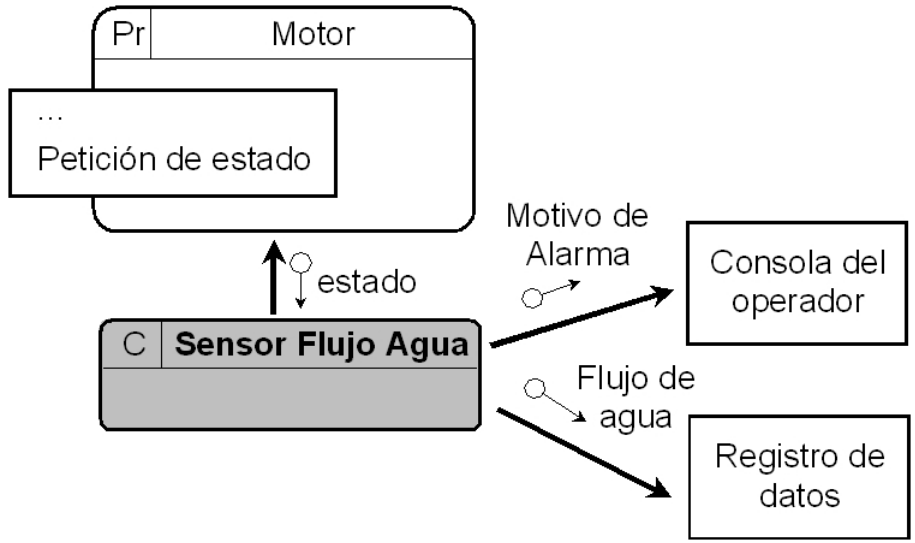
```
#define FLUJO_AGUA_PERIOD_SECONDS      1  /* 1000 ms */
#define FLUJO_AGUA_PERIOD_NANOSECONDS  0
#define FLUJO_AGUA_PRIORITY            9
```

Traducción a POSIX/C

El objeto “Sensor Flujo Agua”

2. Interfaz

No tiene interfaz al exterior. No obstante necesitamos definir un **tipo** accesible al registro de datos



Fichero *flujo_agua.h*

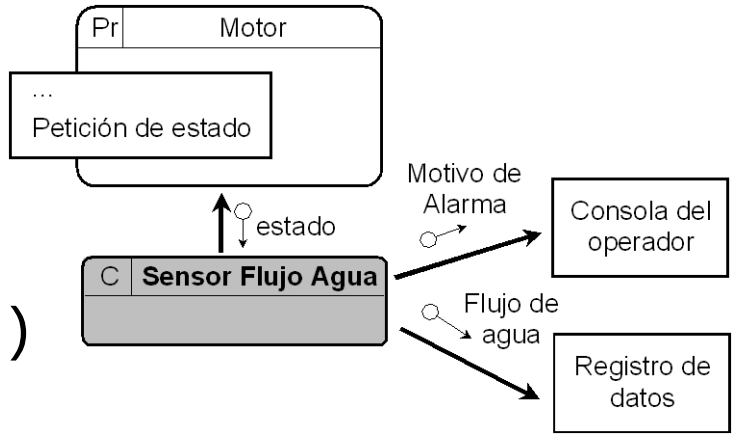
```
-- CICLICO
enum Water_Flow {No = 0; Yes = 1};
typedef enum Water_Flow Water_Flow ;
void FLUJO_AGUA_init(void);
```

Traducción a POSIX/C

El objeto “Sensor Flujo Agua”

3. Implementación Fichero *flujo_gua.c*

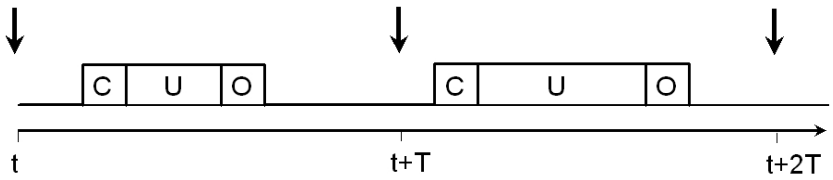
1. `initialise`, inicializa el sensor
2. `periodic_Frame`, el código periódico




```
static void initialise(void)
{
  /* Enable device */
  /* Not implemented */
}
```

Traducción a POSIX/C

El objeto “Sensor Flujo Agua”

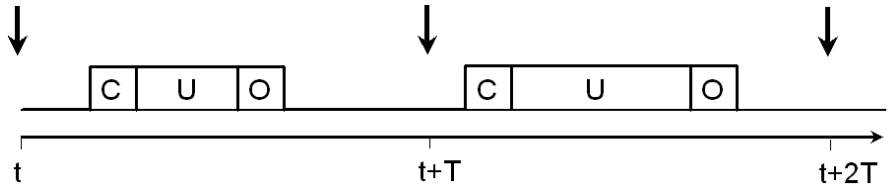



```
static void periodic_Frame (void)
{
  Pump_Status        current, last;
  Operational_Status status;
  Water_Flow         flow;

  MOTOR_request_Status(&status);
  current = status.Ps;
  flow = Get_Flow_From_HW(); /* Not implemented */
  if ((current == On) && (last == On) && (flow == No))
    OPERATOR_CONSOLE_alarm(Pump_Fault);
  else {
    if((current == Off) && (last == Off) && (flow == Yes))
      OPERATOR_CONSOLE_alarm(Pump_Fault);
  }
  last = current;
  DATALOGER.Water_Flow_Log(Flow);
}
```

Fichero *flujo_gua.c*

Traducción a POSIX/C

El objeto “Sensor Flujo Agua”




```
#include “flujo_agua.h"
#include “flujo_agua_Rtatt.h"
#include "motor.h"
#include <operator_Console.h>
#include <data_Logger.h>
#include <pthread.h>
#include <time.h>

static void error() {exit(1);}
static void initialise(void) ...
static void periodic_Frame(void) ...

void AGUA_init()
{
  struc timespec period;
  period.tv_sec  = FLUJO_AGUA_PERIOD_SECONDS;
  period.tv_nsec = FLUJO_AGUA_PERIOD_NANOSECONDS;
  periodic_init(&period, FLUJO_AGUA_PRIORITY);
  /* Código de negocio */
}
```

Fichero *flujo_agua.c*

Traducción a POSIX/C

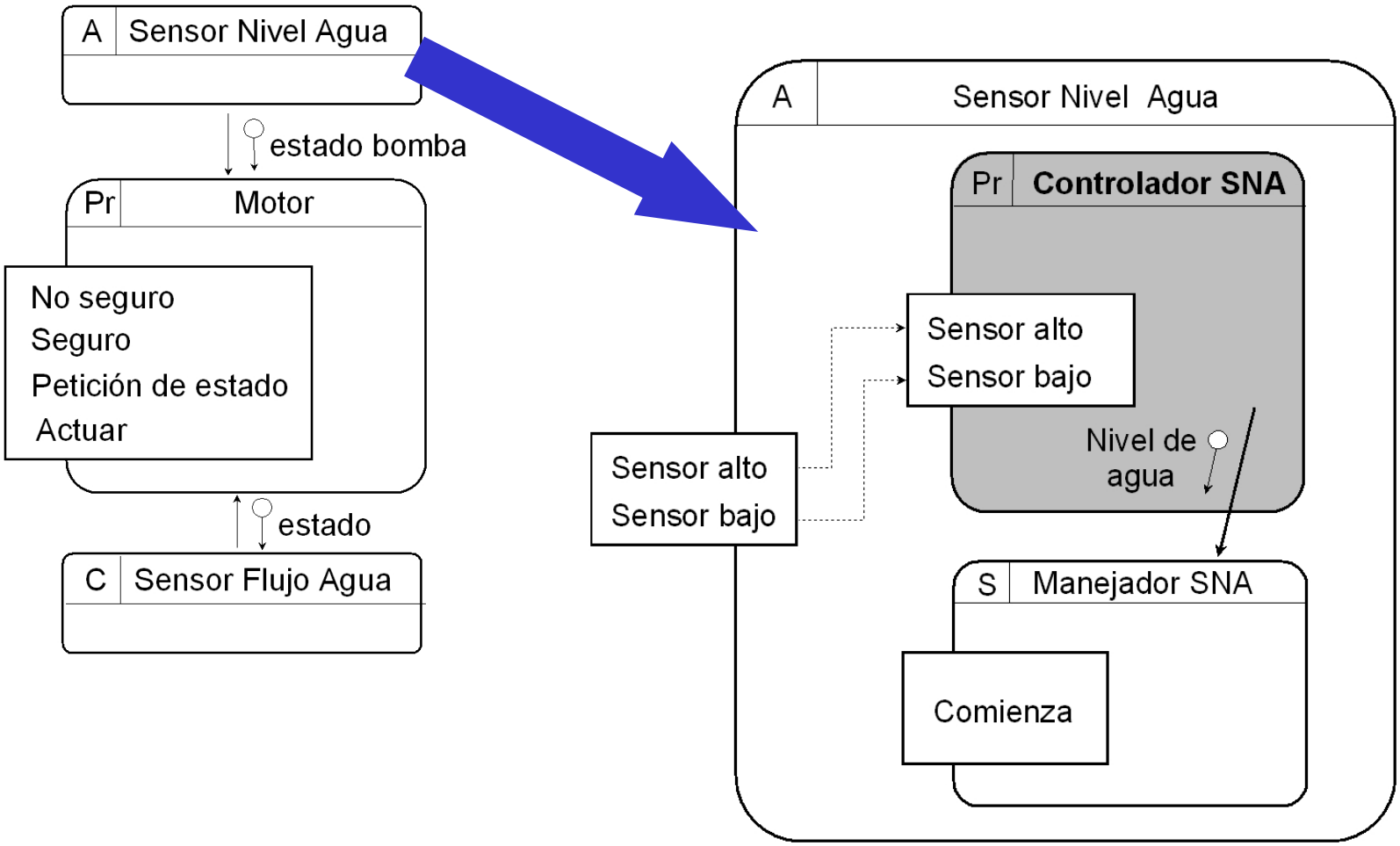
El objeto “Sensor Flujo Agua”

Fichero *flujo_agua.c*

```
void periodic_Body()
{
  int               signum;
  sigset_t          set;
  struct sigevent   sig;
  timer_t           timer;
  struct itimerspec required, old;
  struct timespec   first, period;

  sig.sigev_notify = SIGEV_SIGNAL;
  sig.sigev_signo  = SIGRTMIN;
  if(0 > clock_gettime (CLOCK_REALTIME, &first)) error();
  first.tv_sec   = first.tv_sec + 1;
  period.tv_sec  = FLUJO_AGUA_PERIOD_SECONDS;
  period.tv_nsec = FLUJO_AGUA_PERIOD_NANOSECONDS; /* 1000 ms */
  required.it_value    = first;
  required.it_interval = period;
  if(timer_create(CLOCK_REALTIME,&sig,&timer))   error();
  if(sigemptyset(&set))                          error();
  if(sigaddset(&set, SIGRTMIN))                  error();
  if(timer_settime(timer, 0, &required, &old))   error();
  initialise()
  while(1) {
    if (sigwait(&set, &signum))                  error();
    periodic_Frame();
  }
}
```

Traducción a POSIX/C

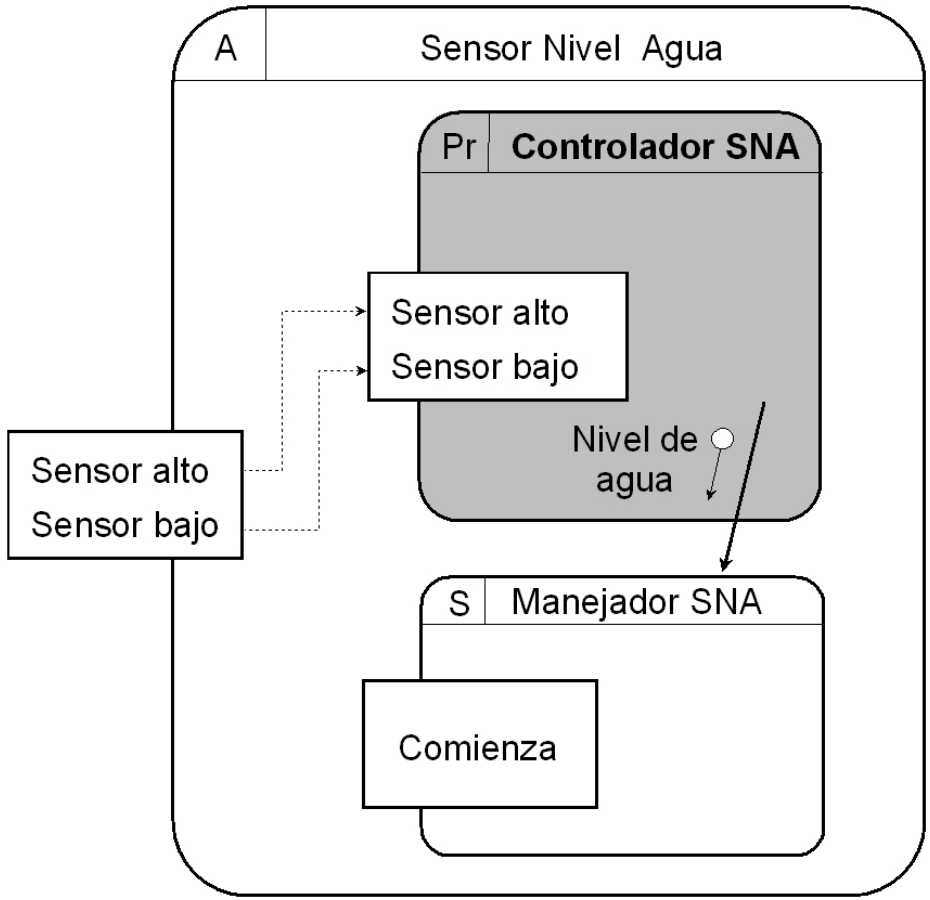
El objeto “Controlador Sensor Nivel Agua (SNA)”



Traducción a POSIX/C

El objeto “Controlador SNA”

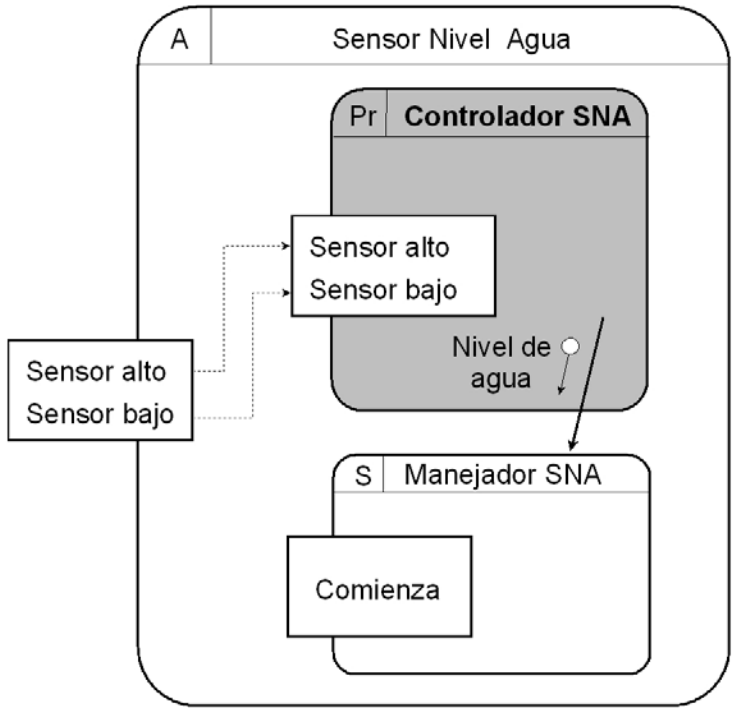
- Es un objeto protegido que encapsula las *rutinas de interrupción* de los sensores de nivel
- HRT_HOOD no permite que un objeto esporádico sea invocado por más de una operación de arranque
- Su **objetivo** es mapear dos rutinas de interrupción en una única operación "Comienza" del objeto “Manejador SNA” con el parámetro de nivel correspondiente



Traducción a POSIX/C

El objeto “Controlador SNA”

1. **Atributos de tiempo real**
   (Mutex que requiere un techo de prioridad)



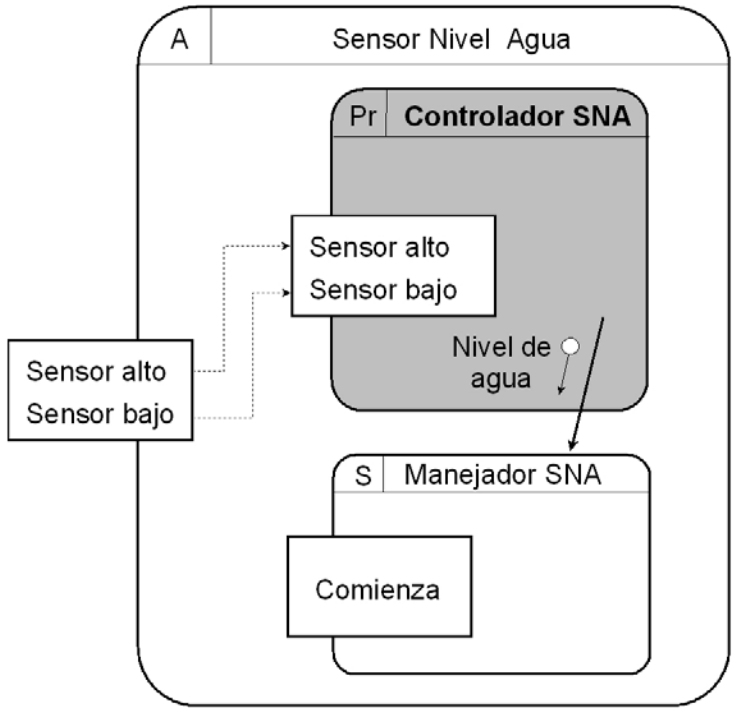
Fichero *contr_Sna_rtatt.h*

```
#define CONT_SNA_CEILING_PRIORITY 11
```

Traducción a POSIX/C

El objeto “Controlador SNA”

2. Interfaz

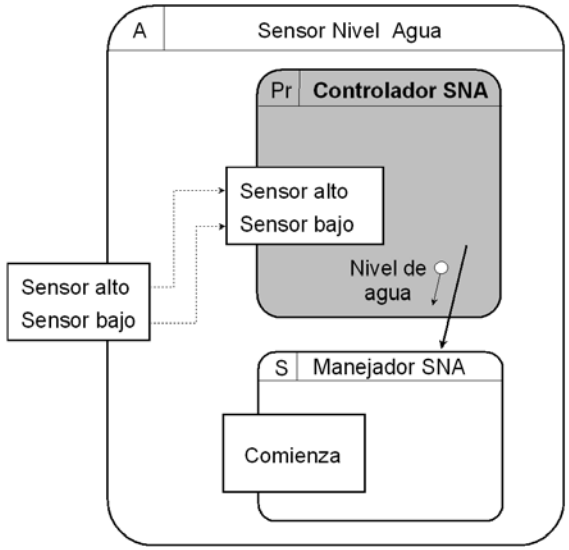


Fichero *contr_Sna.h*

```
/* PROTECTED */
extern void CONTR_SNA_high(void);
extern void CONTR_SNA_low (void);
```

Traducción a POSIX/C

El objeto “Controlador SNA”



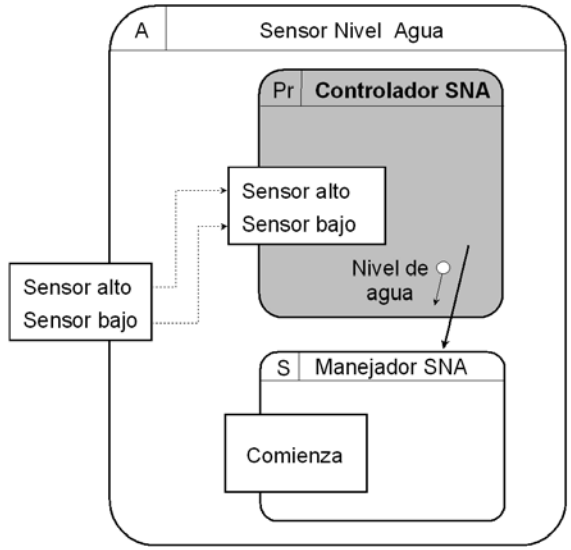
3. Implementación

Fichero *contr_Sna.c*

```
#include “contr_Sna.h"
#include “contr_Sna_Rtatt.h"
#include <data_Logger.h>
#include <pthread.h>

static pthread_mutex_t mutex;

void CONTR_SNA_init(void)
{
  pthread_mutex_init(&mutex);
}
```

Traducción a POSIX/C

El objeto “Controlador SNA”



3. Implementación

Fichero *contr_Sna.c*

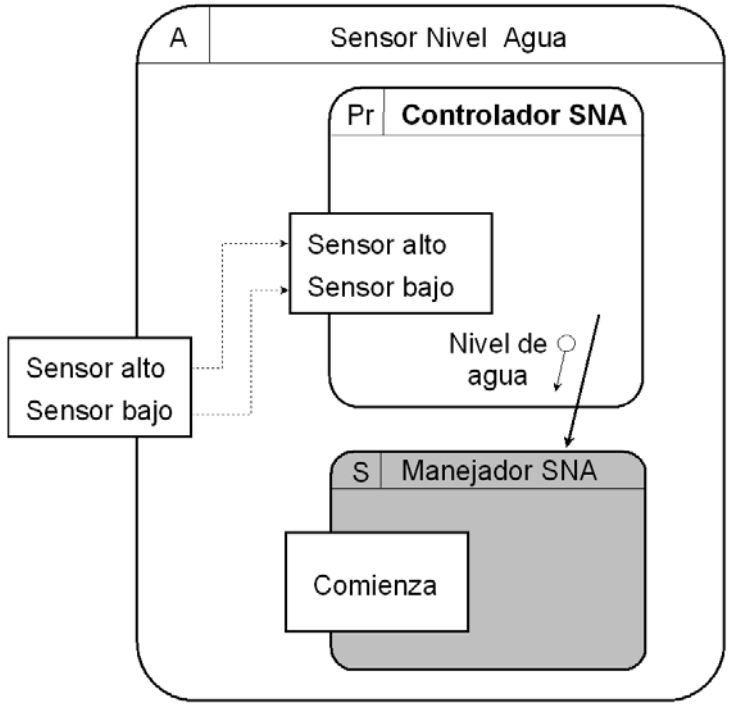
```
void CONTR_SNA_high (void)
{
  pthread_mutex_lock(&mutex);
  HANDL_SNA_start(High);
  pthread_mutex_unlock(&mutex);
}

void CONTR_SNA_low (void)
{
  pthread_mutex_lock(&mutex);
  HANDL_SNA_start(Low);
  pthread_mutex_unlock(&mutex);
}
```

Traducción a POSIX/C

El objeto "Manejador SNA"

1. **Atributos de tiempo real**

   El objeto protegido *interno* requiere un techo de prioridad y la tarea esporádica una prioridad



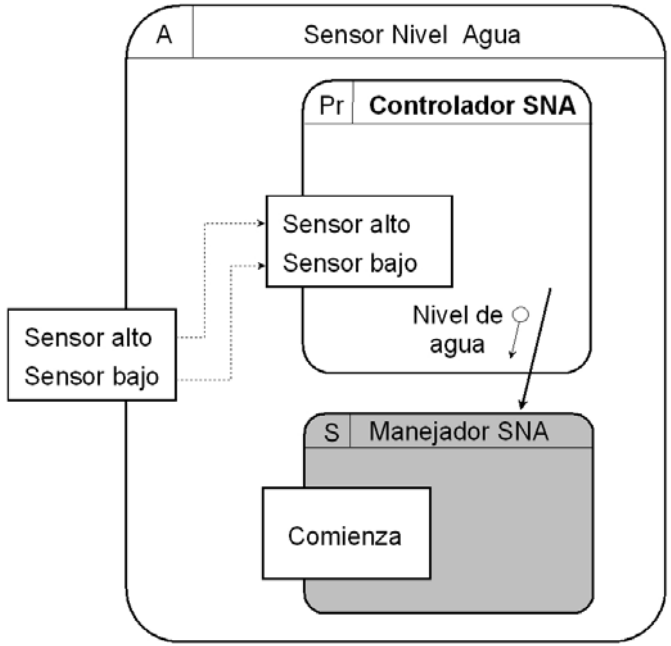
Fichero *handler_Sna_rtatt.h*

```
#define HANDLER_SNA_CEILING        1
#define HANDLER_SNA_PRIORITY       9
```

Traducción a POSIX/C

El objeto “Manejador SNA”

2. Interfaz



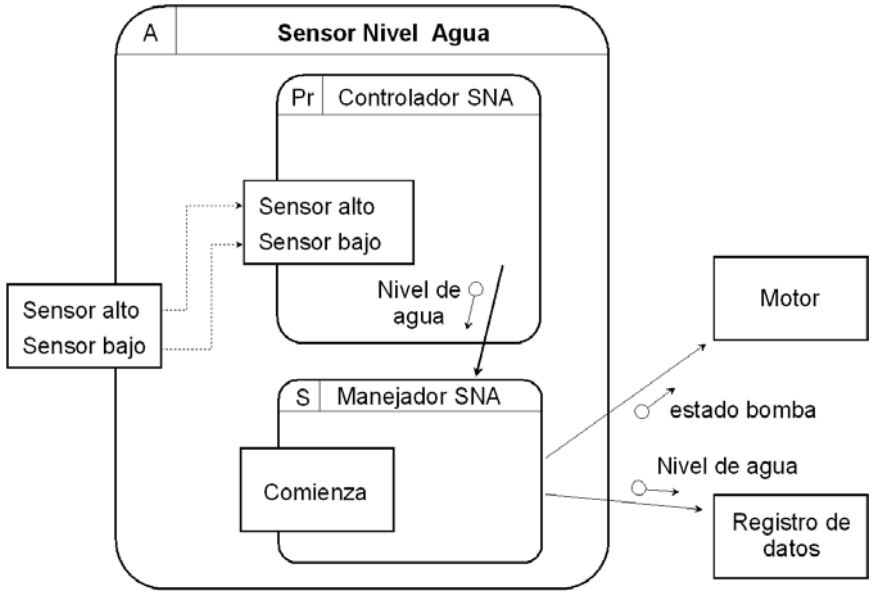
Fichero *handler_Sna.h*

```
/* Esporadico */
enum Water_Mark {High = 0; Low = 1};
typedef enum Water_Mark Water_Mark;
int HANDLER_SNA_start(Water_Mark *int);
```

Traducción a POSIX/C

El objeto "Manejador SNA" 3. Implementación

```
#include motor.h
#include data_Logger.h
static void initialise(void)
{
  /* Enable device (Not implemented) */
}
static void sporadic_frame (void)
{
  Pump_Status        current, last;
  Operational_Status status;
  Water_Flow         flow;
  MOTOR_request_Status(&status);
  current = status.Ps;
  flow = Get_Flow_From_HW(); /* Not implemented */
  if ((current == On) && (last == On) && (flow == No))
    OPERATOR_CONSOLE_alarm(Pump_Fault);
  else
    if((current == Off) && (last == Off) && (flow == Yes))
      OPERATOR_CONSOLE_alarm(Pump_Fault);
  last = current;
  Data_Logger.Water_Flow_Log(Flow);
}
```

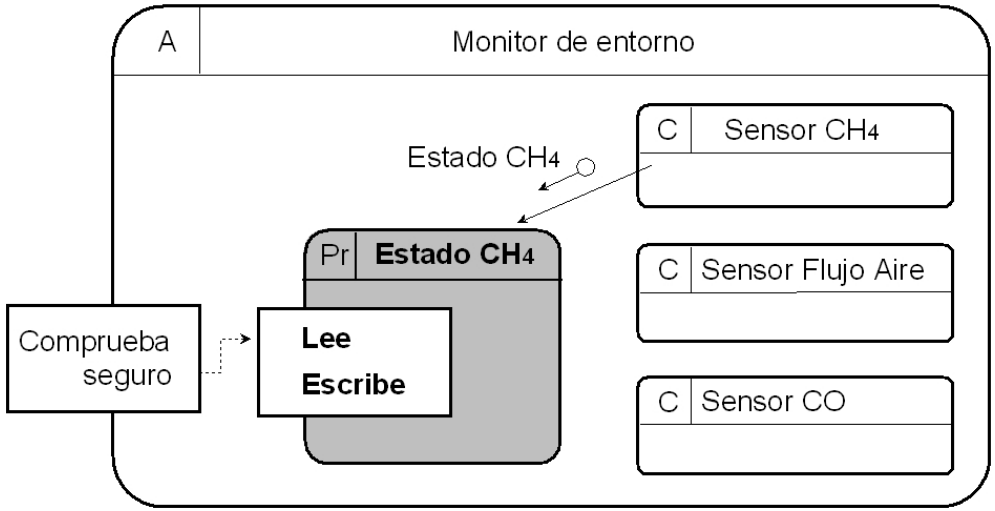


Fichero *handler_Sna.c*

Traducción a POSIX/C

El objeto "Estado CH4"

1. **Atributos de tiempo real** (Mutex que requiere un techo de prioridad)

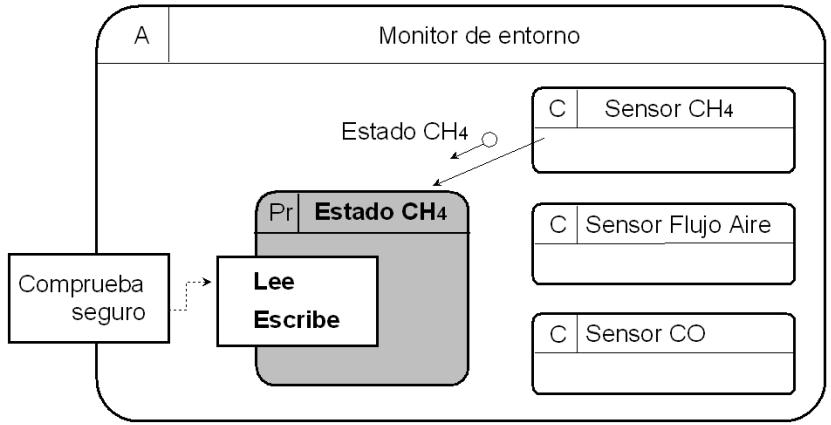


Fichero *ch4_estado_rtatt.h*

```
#define CH4_ESTADO_CEILING_PRIORITY 10
```

Traducción a POSIX/C

El objeto “Estado CH4”



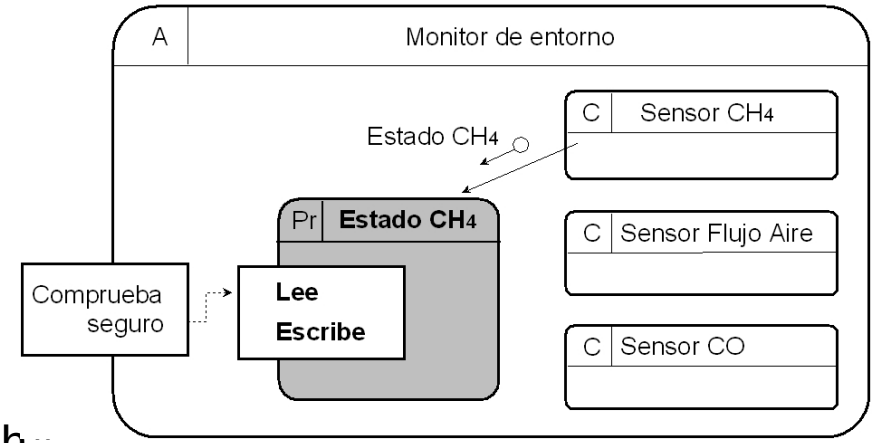
2. Interfaz

Fichero *ch4_estado.h*

```
/* PROTECTED */
enum Status {Motor_Safe; Motor_Unsafe};
typedef enum Status Methane_Status;
extern void   CH4_ESTADO_init (void);
extern status CH4_ESTADO_read (void);
extern void   CH4_ESTADO_write(Status status);
```

Traducción a POSIX/C

El objeto “Estado CH4”



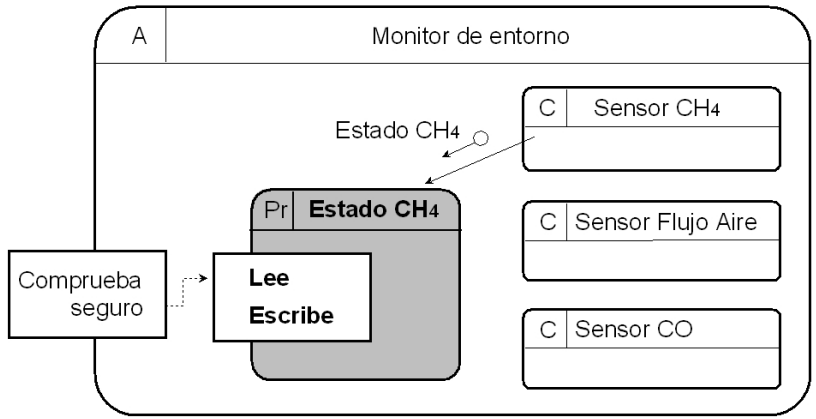
3. Implementación

Fichero *ch4_estado.c*

```
#include "ch4_estado.h"
#include "ch4_estado_rtattr.h
#include <pthread.h>

static status           status;
static pthread_mutex_t mutex;

void CH4_estado_init(void)
{
  status = Motor_Unsafe;
  pthread_mutex_init(&mutex);
}
```

Traducción a POSIX/C

El objeto “Estado CH4”



Fichero *ch4_estado.c*

```
status CH4_estado_read(void)
{
  status stat;
  pthread_mutex_lock(&mutex);
  stat = status;
  pthread_mutex_unlock(&mutex);
  return(stat);
}

void CH4_estado_write(status stat)
{
  pthread_mutex_lock(&mutex);
  status = stat;
  pthread_mutex_unlock(&mutex);
  return;
}
```