

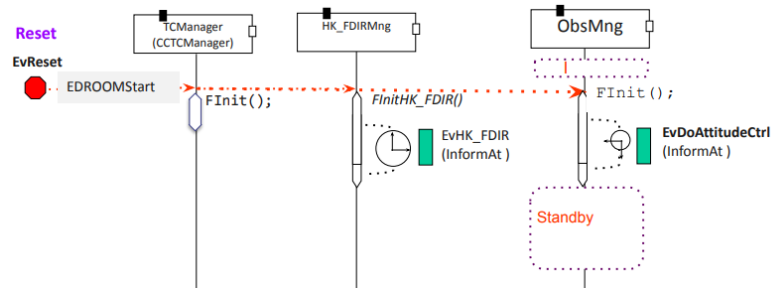
# **ENTREGABLES:**

**DESARROLLO E IMPLEMENTACIÓN**  
**DEL MODELO EDROOM**

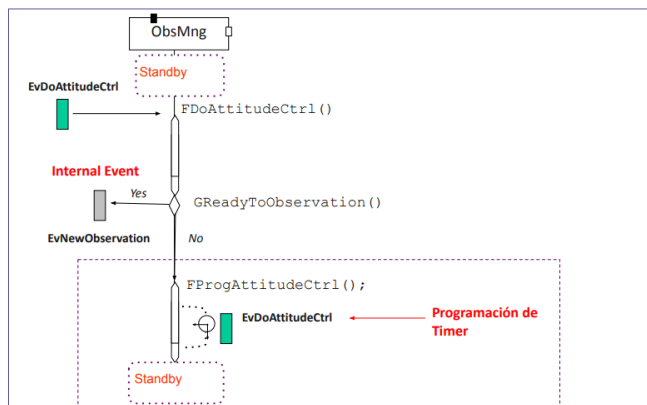
## I. Escenarios en los que está involucrado el nuevo componente ObsMng que controla la observación

Tal y como se refleja tanto en el Diagrama de Estados como en el modelo de Edroom, el componente ObsMng se encuentra implicado en los siguientes 4 escenarios:

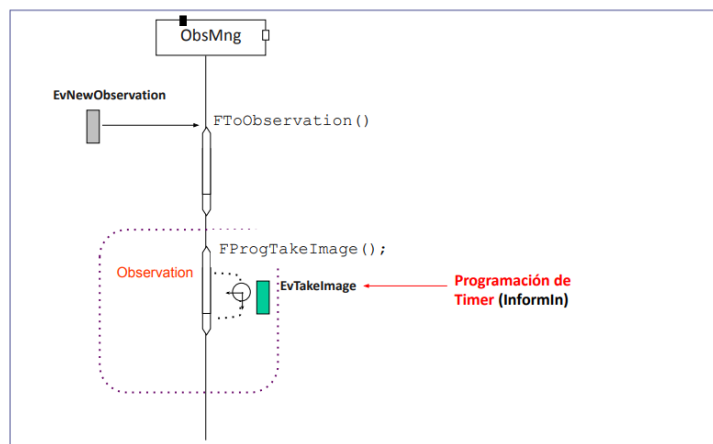
- **Inicialización:** En este escenario es donde se configura el *timer* que controla el sistema de actitud.



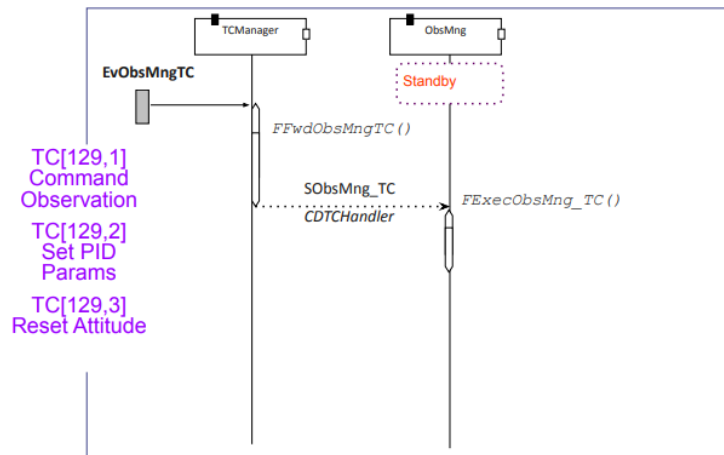
- **Standby (Espera):** Este escenario supervisa el control de actitud hasta que se alcanza el valor deseado, momento en el cual se activa el estado Observation. Si la observación no está lista, o no ha sido programada ninguna observación se regresará al estado Standby



- **Observación (Captura de imágenes):** Si, por el contrario, la observación sí está lista, se ejecutará la toma de imágenes como parte de la tarea de observación, tras lo cual retorna automáticamente al estado Standby.



- **Conexión con el TCManager:** El componente ObsMng ejecutará todos los telecomandos del servicio ST[129] que se envían desde TCManager



## 2. **Definición de la clase protocolo a añadir al Modelo EDROOM, aportando toda la información de cada mensaje**

Protocol Class Edition

Name: CPObsCtrl

Design | Analysis

Input Messages:

- SObsMng\_TC

Output Messages:

Protocol Brief

Message Edition Box

Signal Name: SObsMng\_TC

Data Class: CDTCHandler

☐ Synchronous Invoke ☒ Asynchronous

☐ Synchronous Reply To --->

Edit Message

New Input Message New Output Message

Delete Message

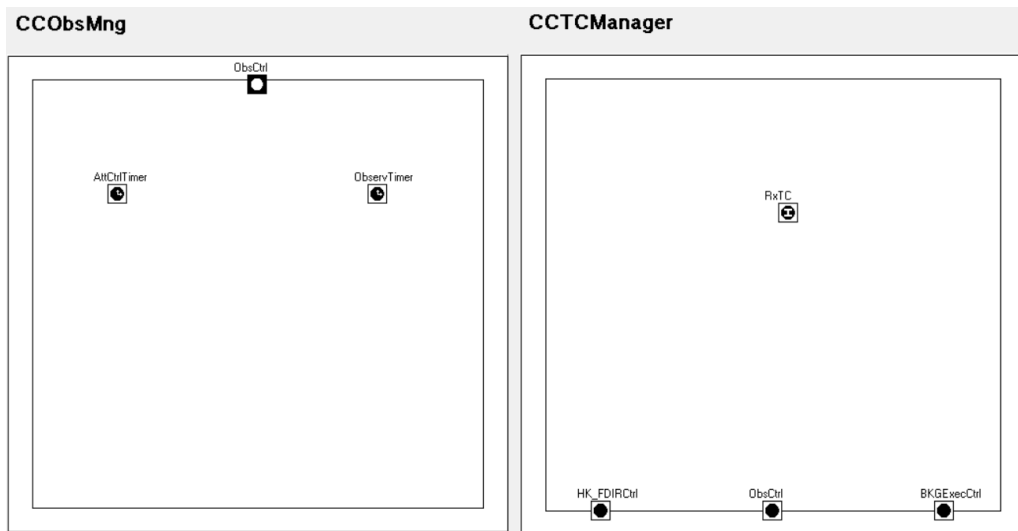
Modify Cancel

Tal y como se muestra en la imagen, la nueva clase de protocolo definida en el modelo, denominada CPObsCtrl, incluye únicamente un mensaje de entrada de tipo asíncrono con los siguientes parámetros:

- **Nombre de señal:** SObsMng\_TC
- **Clase de datos asociada:** CDTCHandler

Esta configuración permite que el componente receptor procese comandos desde el TC Manager mediante una señal directa al componente ObsMng.

### 3. Diseño de la interfaz de la clase componente CCObsMng



La clase componente CCObsMng presenta una interfaz compuesta por tres elementos principales:

- **AttCtrlTimer**: Es el temporizador responsable del control de actitud, encargado de gestionar los intervalos de activación del sistema de orientación.

Port Configuration

Name: AttCtrlTimer

☐ Conjugated

Type: Internal

Protocol Class: EDROOMTimingSAP

Modify Cancel

- **ObservTimer**: Temporizador dedicado a la toma de imágenes, se encarga de regular la frecuencia con la que se ejecuta la función de observación.

Port Configuration

Name: ObservTimer

☐ Conjugated

Type: Internal

Protocol Class: EDROOMTimingSAP

Modify Cancel

- **ObsCtrl**: Corresponde con el puerto de comunicación nominal vinculado al protocolo CPObsCtrl, el cual permite la conexión directa entre CCObsMng y CCTCManager (conjugado), facilitando la recepción de telecomandos.

Port Configuration

Name: ObsCtrl

☐ Conjugated

Type: External

Protocol Class: CPObsCtrl

Modify Cancel

Port Configuration

Name: ObsCtrl

☒ Conjugated

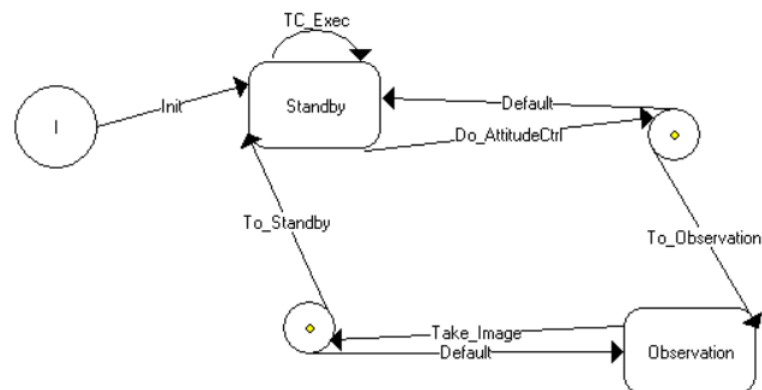
Type: External

Protocol Class: CPObsCtrl

Modify Cancel

## 4. Diseño del comportamiento de la clase componente CCObsMng

### 4.1 La máquina de estados de la clase componente



### 4.2 Variables y Constantes de la clase componente



En la clase componente se han definido los siguientes elementos

- **VNextTimeOut**: Variable de tipo Pr\_Time utilizada para gestionar y actualizar los valores de temporización asociados al componente AttCtrlTimer.

Variables and Constants Edition

Name:

Init Value:

Class:

☐ Constant ☒ Variable

Array ☐

Dimension

☒ OK ☐ Cancel

- **CImageInterval**: Constante de tipo Pr\_Time con valor inicial 0.5, que establece el intervalo temporal empleado por el temporizador ObservTimer durante las fases de captura de imágenes.

Variables and Constants Edition

Name:

Init Value:

Class:

☒ Constant ☐ Variable

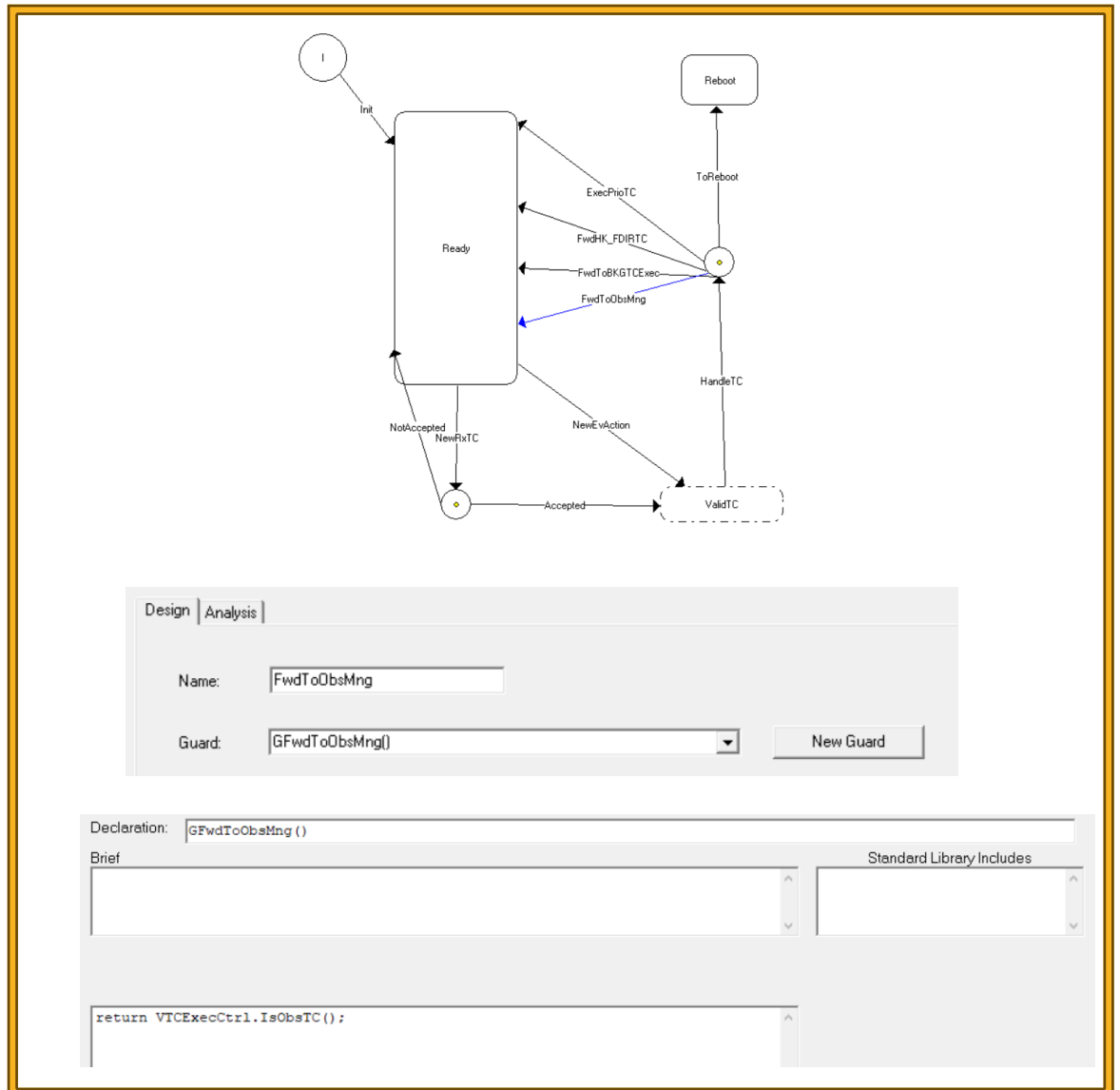
Array ☐

Dimension

☒ OK ☐ Cancel

### 4.3 Triggers y Guardas (CCTCManager)

- **FwdToObsMng:** Añadimos en la guarda una rama para enviar el telecomando al ObsMng En el caso de esta función, al no ser una transición sino una rama, no presenta ningún trigger.



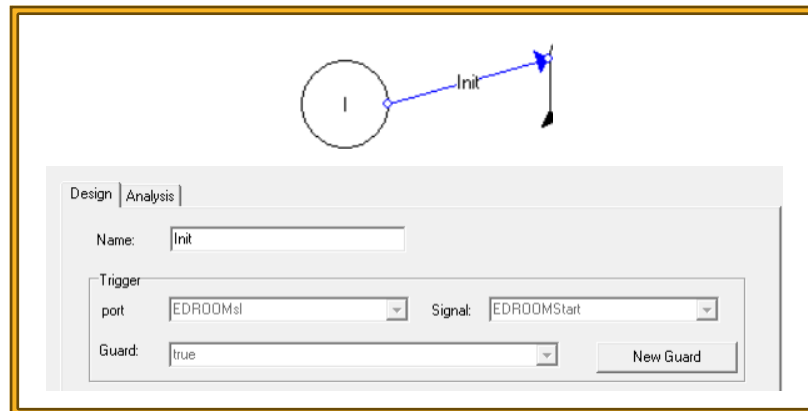
Para la implementación de la guarda se ha seguido el mismo procedimiento que en el resto de ramas.

Se ha elegido el nombre “IsObsTC” siguiendo el esquema del código eclipse.

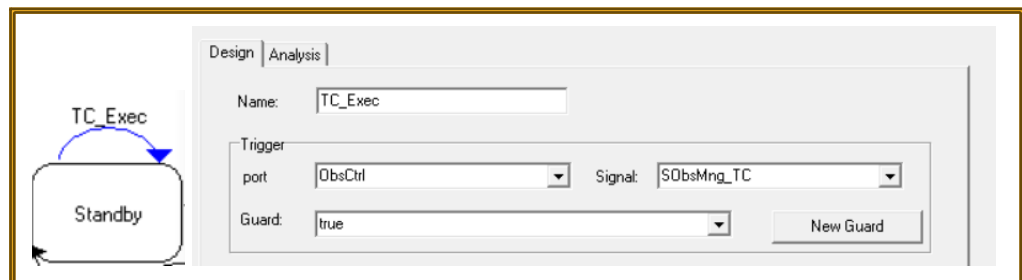
```
//TODO DONE Add in EDROOM model the guard to handle ST[129] TCs  
bool IsObsTC(){return (ExecCtrlObservTC==mExecCtrl);};
```

#### 4.4 Triggers y Guardas (CCObsMng)

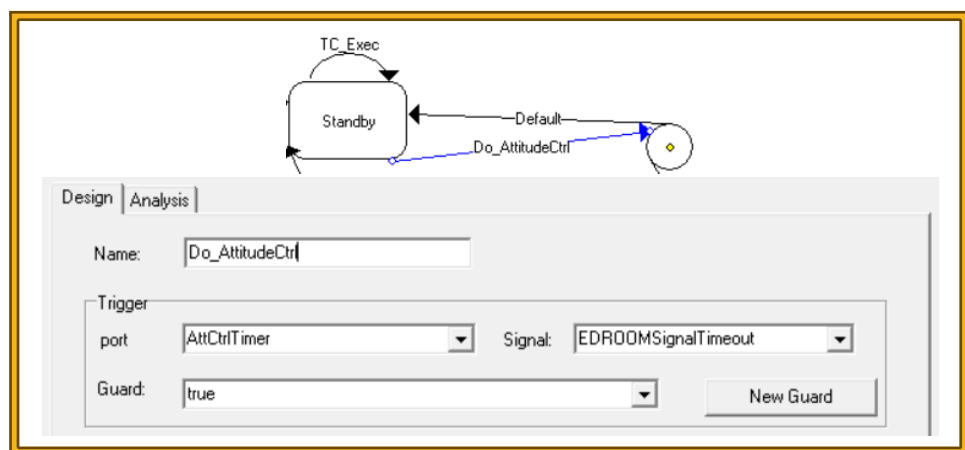
- **Init:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger.



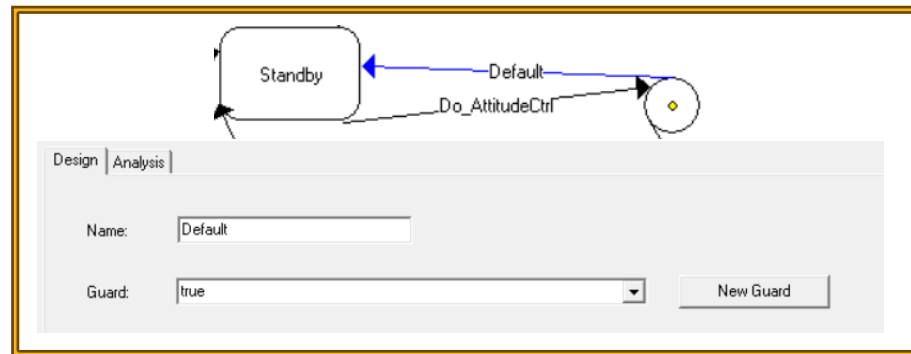
- **TC Exec:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger.



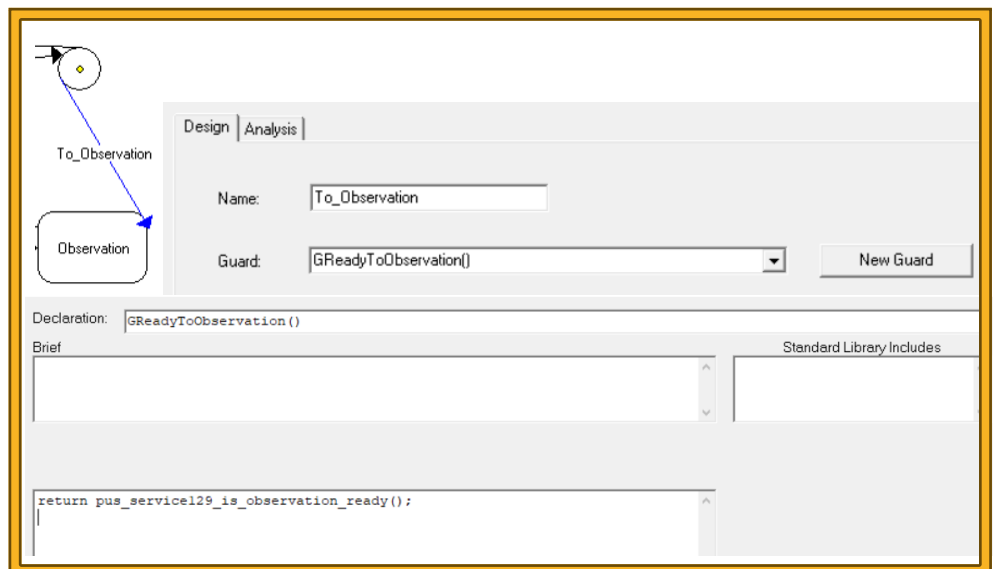
- **Do AttitudeCtrl:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger.



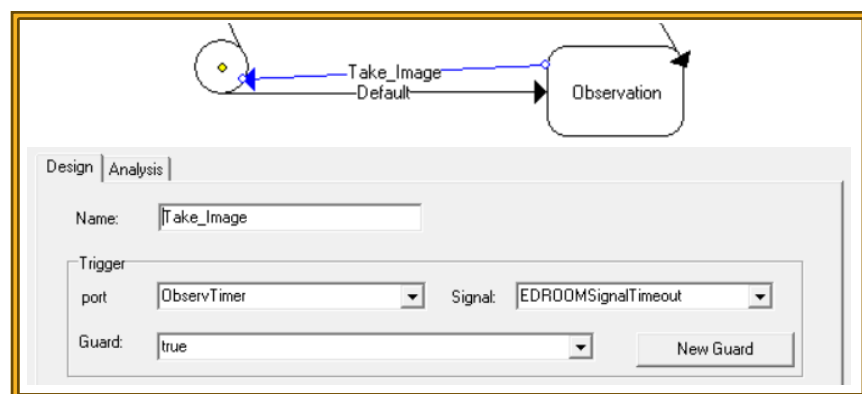
- **Default:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger.



- **To Observation:** Esta rama evalúa la guarda `GReadyToObservation()` que indica que se debe iniciar la observación programada. En caso de no cumplirse la condición de la guarda, se regresaría al estado Standby por medio del Default

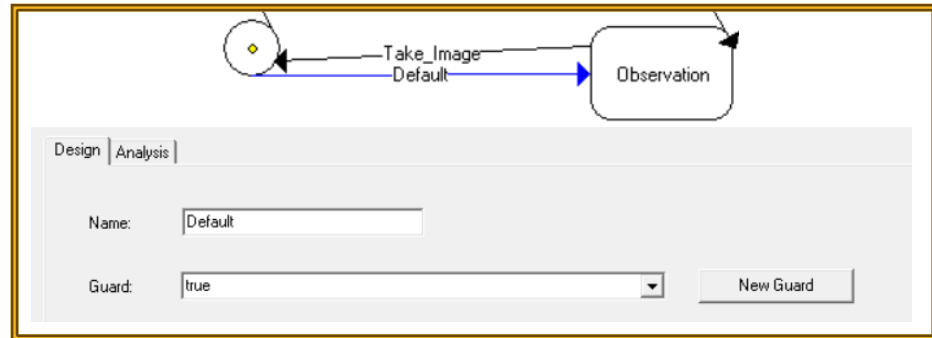


- **Take Image:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger

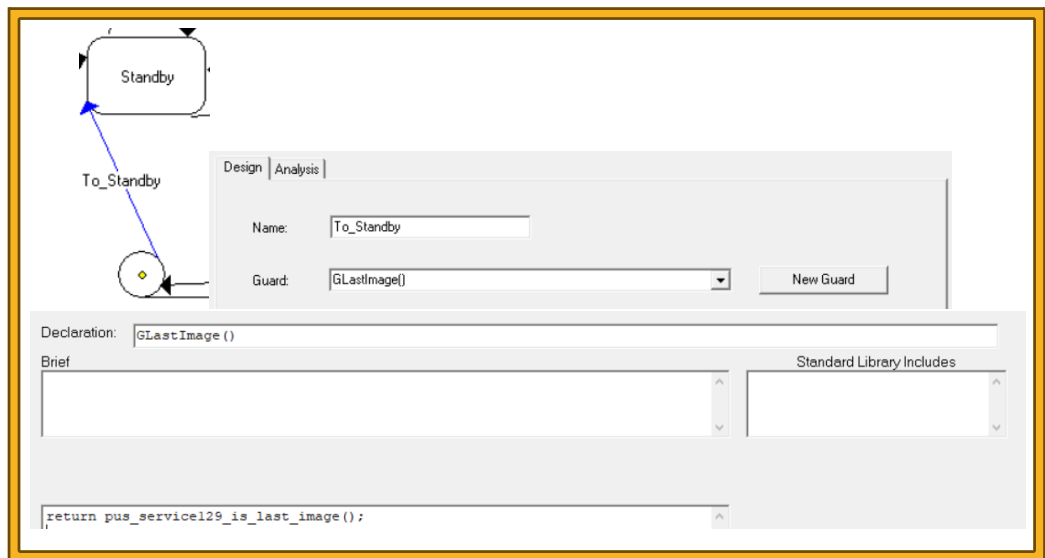




- **Default:** Al ser una transición no requiere de guarda porque no tiene ninguna condición que cumplir más allá del propio trigger



- **To Standby:** Tras la toma de imágenes el componente analiza si es la última imagen de la observación mediante la guarda `GLastImage()`. Si la guarda `GLastImage()` retorna true, se regresa al estado Standby



## 4.5 Actions (CCTCManager)

- **FwdToObsMng → FFwdToObsMng (Send)**: Confirma el envío del evento desde el TcManager al ObsMng.

Branch Handler

Trans MsgBack->Data Handler :  New MsgBackDataH

Action: :  New Action

Send : FFwdToObsMng() New Send

Invoke :  New Invoke

Branch MsgBack->Data Handler :  New MsgBackDataH

Declaration: FFwdToObsMng()

Brief

```
{
    CDTCHandler * pSObsMng_TC_Data = EDROOMPoolCDTCHandler.AllocData();

    // Complete Data
    *pSObsMng_TC_Data = VCurrentTC;

    ObsCtrl1.send(SObsMng_TC, pSObsMng_TC_Data, &EDROOMPoolCDTCHandler);
}
```

Standard Library Includes

EDROOM Service

send

Port  ObsCtrl

Signal  SObsMng\_TC

Data Class  CDTCHandler

Service Request

## 4.6 Actions (CCObsMng)

- **Init → FInit\_Obs (Inform InAt):** Ejecuta en la inicialización del componente antes de alcanzar el estado Standby con un periodo de ejecución de 100 ms

The screenshot displays the IDE interface for configuring the **FInit\_Obs** action. The top panel, titled "Transition Handler", contains several dropdown menus and buttons for configuring the action's behavior:

- Msg->Data Handler:** A dropdown menu with a semicolon (;) selected, and a "New MsgDataHand" button.
- Action:** A dropdown menu with "FInit\_Obs();" selected, and buttons for "New Action" and "New Inform(In,At)".
- Send :** A dropdown menu with a semicolon (;) selected, and a "New Send" button.
- Invoke :** A dropdown menu with a semicolon (;) selected, and a "New Invoke" button.
- MsgBack->Data Handler:** A dropdown menu with a semicolon (;) selected, and a "New MsgBackDataH" button.

The bottom panel shows the code editor for the **FInit\_Obs** action. The declaration is `FInit_Obs ()`. The code includes a timing service call and a service request:

```
{
  Pr_Time time;

  //Timing Service useful methods
  time.GetTime(); // Get current monotonic time
  time += Pr_Time(0,100000); // Add X sec + Y microsec
  VNextTimeout=time;

  AttCtrlTimer.InformAt( time );
}
```

On the right side of the code editor, the "EDROOM Service" configuration panel is visible, showing the following settings:

- InformAt:** A dropdown menu.
- Port:** A dropdown menu with "AttCtrlTimer" selected.
- Signal:** A dropdown menu with "EDROOMSignalTimeout" selected.
- Data Class:** A dropdown menu with "CDTCHandler" selected.
- Service Request:** A button.

- **TC Exec → FObsCtrl\_exec (MsgDataHandler):** Procesa la ejecución de los telecomandos enviados desde el TCMnager

Transition Handler

Msg->Data Handler:

FObsCtrl\_exec();

New MsgDataHand

Action:

:

New Action

Send:

:

New Inform(In,At)

Invoke

:

New Send

MsgBack->Data Handler:

:

New Invoke

New MsgBackDataH

Declaration: FObsCtrl\_exec()

Brief

Standard Library Includes

```

{
    CDTCHandler & varSObsMng_TC = *(CDTCHandler *)Msg->data;

    // Data access
    varSObsMng_TC.ExecTC;
}

```

EDROOM Service

Msg->data

Port

ObsCtrl

Signal

SObsMng\_TC

Data Class

CDTCHandler

Service Request

- **Do AttitudeCtrl → FDoAttitudeCtrl (Action):** Por medio de la llamada al servicio 129, se aplica el algoritmo de control de actitud de acuerdo a la orientación objetivo y actualiza los parámetros del system data pool que gestiona este servicio

The screenshot shows the configuration window for the **FDoAttitudeCtrl** service. The **Transition Handler** section contains several dropdown menus and buttons:

- Msg->Data Handler:** A dropdown menu with a colon in the first position, and a **New MsgDataHand** button.
- Action:** A dropdown menu with **FDoAttitudeCtrl()** selected, and **New Action** and **New Inform(In,At)** buttons.
- Send :** A dropdown menu with a colon in the first position, and a **New Send** button.
- Invoke :** A dropdown menu with a colon in the first position, and a **New Invoke** button.
- MsgBack->Data Handler:** A dropdown menu with a colon in the first position, and a **New MsgBackDataH** button.

Below the transition handler section, the **Declaration:** field contains `FDoAttitudeCtrl()`. The **Brief** field is empty. The **Standard Library Includes** field is also empty. At the bottom, a code editor shows the following code:

```
pus_service129_do_attitude_ctrl();
```

- **Default → FProgAttitudeCtrl (Inform InAt):** Debido al que el “timer” no se programa en el Standby este debe de ejecutarse en la propia transición

The screenshot shows the configuration window for the **FProgAttitudeCtrl** service. The **Branch Handler** section contains several dropdown menus and buttons:

- Trans MsgBack->Data Handler:** A dropdown menu with a colon in the first position, and a **New MsgBackDataH** button.
- Action:** A dropdown menu with **FProgAttitudeCtrl()** selected, and **New Action** and **New Inform(In,At)** buttons.
- Send :** A dropdown menu with a colon in the first position, and a **New Send** button.
- Invoke :** A dropdown menu with a colon in the first position, and a **New Invoke** button.
- Branch MsgBack->Data Handler:** A dropdown menu with a colon in the first position, and a **New MsgBackDataH** button.

Below the branch handler section, the **Declaration:** field contains `FProgAttitudeCtrl()`. The **Brief** field is empty. The **Standard Library Includes** field is also empty. The code editor shows the following code:

```
{
    Pr_Time time;

    //Timing Service useful methods
    VNextTimeout += Pr_Time(0,100000); // Add X sec + Y microsec
    time = VNextTimeout

    AttCtrlTimer.InformAt( time );
}
```

On the right side of the code editor, there is a **EDROOM Service** panel with the following settings:

- InformAt:** A dropdown menu with **InformAt** selected.
- Port:** A dropdown menu with **AttCtrlTimer** selected.
- Signal:** A dropdown menu with **EDROOMSignalTimeout** selected.
- Data Class:** A dropdown menu with **CDTCHandler** selected.
- Service Request:** A button.

- **To Observation → FToObservation (Action):** Por medio de la guarda GReadyToObservation se analiza si la observación programada está lista para realizarse, en caso afirmativo el evento que se dispara se gestionará con la ejecución de esta acción

- **Observation → FProgTakeImage (Inform InAt):** Asociada a la constante CImageInterval, programa la toma de una imagen mediante el servicio InformIn. Cada vez que se pasa por el estado, este se actualiza el timer.

- **Take Image → FTakeImage (Action):** Su ejecución permite la toma de imágenes, su código es una llamada a `pus_service129_take_image()` que genera la telemetría `TM[129,4]` con la imagen tomada.

The screenshot displays the UML State Machine Editor interface for configuring a transition. The top section, titled 'Transition Handler', contains several fields and buttons:

- Msg->Data Handler:** A dropdown menu with a colon symbol and a 'New MsgDataHand' button.
- Action:** A dropdown menu containing 'FTakeImage();' with 'New Action' and 'New Inform(In,At)' buttons.
- Send :** A dropdown menu with a colon symbol and a 'New Send' button.
- Invoke**: A dropdown menu with a colon symbol and a 'New Invoke' button.
- MsgBack->Data Handler:** A dropdown menu with a colon symbol and a 'New MsgBackDataH' button.

Below this section, the 'Declaration:' field contains the text `FTakeImage ()`. The 'Brief' section is empty. The 'Standard Library Includes' section is also empty. At the bottom, a code editor shows the function call `pus_service129_take_image();`.

- **Default:** Si no se han tomado todas las imágenes regresamos al estado Observation mediante otra transición default. En este caso, la transición no requiere de ninguna acción, pues es el propio estado Observation el encargado de actualizar y programar el timer cada vez que se pasa por él.

The screenshot shows the UML State Machine Editor interface for configuring a 'Default' transition. The top section has the following fields and buttons:

- Name:** A text field containing 'Default'.
- Guard:** A dropdown menu with 'true' selected and a 'New Guard' button.

The 'Branch Handler' section contains several fields and buttons:

- Trans MsgBack->Data Handler**: A dropdown menu with a colon symbol and a 'New MsgBackDataH' button.
- Action:** A dropdown menu with a blue cursor icon and a 'New Action' button.
- Send :** A dropdown menu with a colon symbol and a 'New Send' button.
- Invoke**: A dropdown menu with a colon symbol and a 'New Invoke' button.
- Branch MsgBack->Data Handler:** A dropdown menu with a colon symbol and a 'New MsgBackDataH' button.

- **To Standby → F Standby:** Si se han tomado todas las imágenes, la vuelta al estado Standby se gestiona con la ejecución secuencial de dos acciones, la FEndObservation y la FProgAttitudeCtrl

Ambas acciones marcan el fin de la observación y son las encargadas de devolver el sistema al estado Standby y prepararlo así para una futura observación.