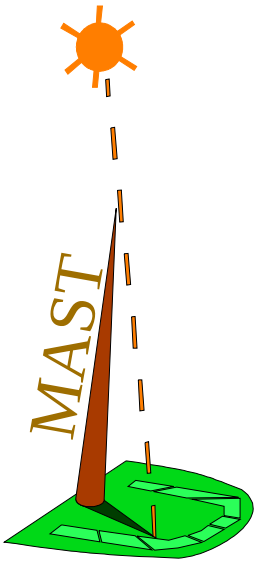
GRUPO DE COMPUTADORES Y TIEMPO REAL

UNIVERSIDAD DE CANTABRIA

ROBOT TELEOPERADO

(EJEMPLO UML MAST)



José M. Drake
Julio L. Medina
Santander, febrero, 2001

TABLE OF CONTENTS

I DESCRIPCIÓN DEL SISTEMA: ROBOT TELEOPERADO.

Este ejemplo muestra la aplicación de la metodología de modelado y análisis de sistemas de tiempo real que propone UML-Mast.

El sistema es un robot teleoperado y consiste en un sistema distribuido constituido por dos procesadores comunicados entre si por un bus CAN. El primero (Station) es una estación de teleoperación basada en una GUI (Grafical User Interface) desde la que el operador monitoriza y controla los movimientos del robot. Es un procesador de tipo PC que opera sobre Windows NT. El segundo (Controller) es un microprocesador embarcado que controla los servos y sensores del robot a través de un bus VME. Este procesador ejecuta una aplicación embarcada desarrollada en Ada'95 sobre un núcleo de tiempo real mínimo tal como Marte_OS. (Marte OS: http://ctrpc17.ctr.unican.es/marte.html).

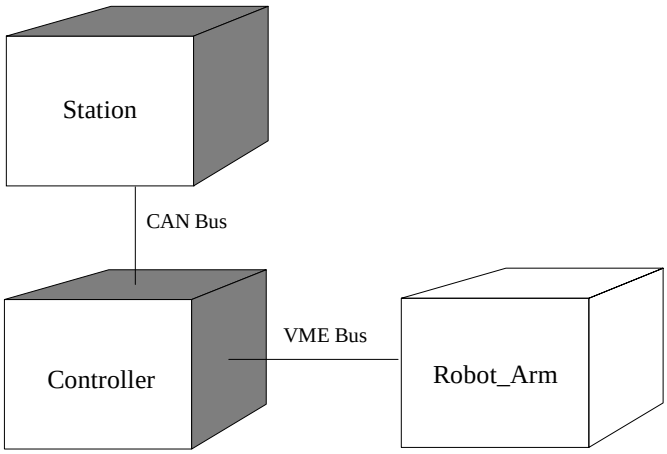


Diagrama de despliegue del robot teleoperado.

Es un sistema distribuido de tiempo real que realiza básicamente tres operaciones:

- Atiende los comandos que el operador introduce por la GUI (teclado y ratón) y los traduce en secuencias de consignas temporizadas que el robot debe seguir para ejecutar el comando.

- Periódicamente actualiza los campos textuales y gráficos presentes en la GUI con información obtenida del hardware del robot y de sus sensores.

- Mantiene el control en bucle cerrado de los servos del robot a través de un proceso de control periódico.

II DISEÑO LÓGICO.

La arquitectura software del sistema se muestra con el diagrama de clases de la página 5. El software del Controller está constituido por tres clases activas y una clase pasiva a través de la que se comunican. La clase Servos_Controller implementa el algorítmo de control de los servos y la monitorización de los sensores. Es una tarea periódica disparada por el timer. La tarea Reporter es también una tarea periódica que transfiere el status del robot a la estación de teleoperación. La tarea Command_Manager es una tarea que se ejecuta con la llegada de un mensaje a través del bus CAN. Su función es interpretar el mensaje recibido y transformarlo en secuencias de consignas para los servos y encolarlas para ser aplicadas en los tiempos que correspondan. Estas tres tareas son concurrentes entre sí y se comunican a través del objeto protegido Servos_Data. El software de la estación es el típico de una aplicación GUI. La tarea Command_Interpreter gestiona los eventos que el operador introduce sobre la GUI. Transforma los eventos en comandos que transfiere hacia el Controller. La tarea Display_Refresher actualiza los datos de la GUI de acuerdo con los mensajes de status que recibe de Reporter. El acceso a la clase pasiva Display_Data se realiza con exclusión mutua por lo que se implementa a través de un objeto protegido.

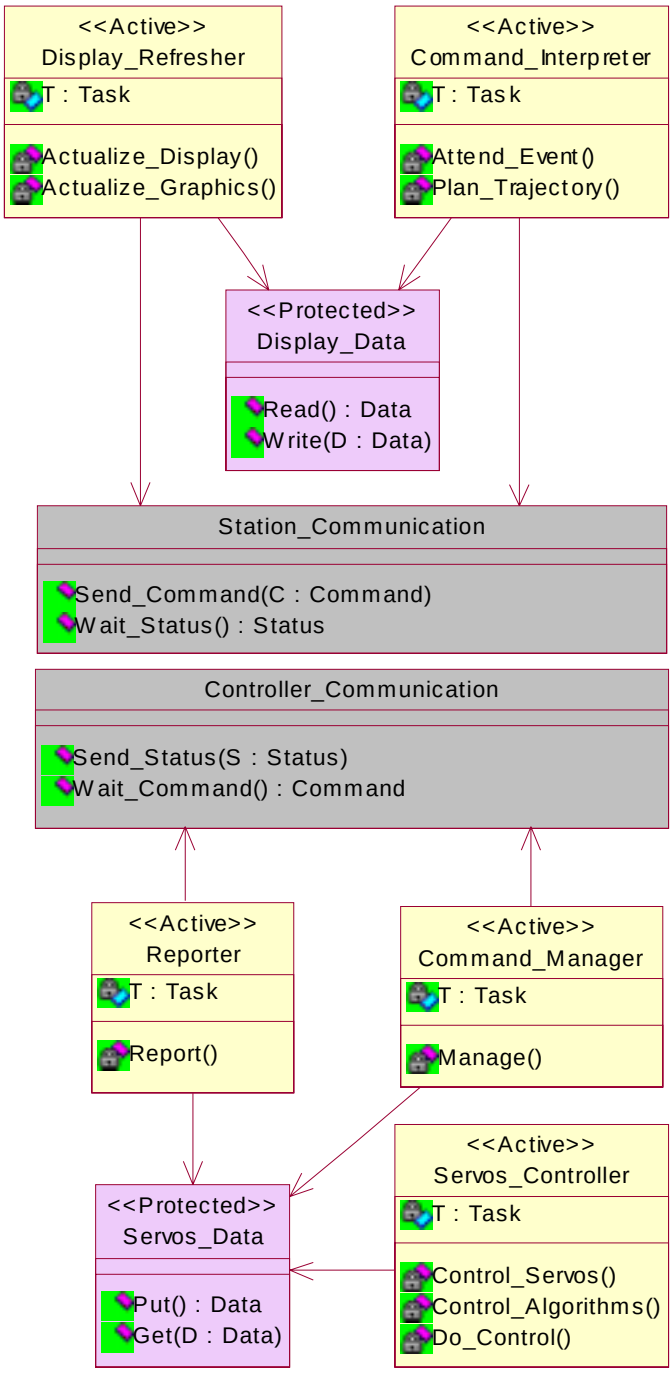
El escenario Control_Teleoperado que se analiza, corresponde a las operaciones que se producen cuando el operador monitoriza y controla el trabajo del robot a través de la estación de teleoperación (Controller y GUI). Este escenario está constituido por tres transacciones asíncronas que interfieren entre sí por compartir los recursos de procesamiento (procesadores y canal de comunicación) y por necesitar sincronizar sus accesos a los objetos pasivos. Las tres transacciones tienen requerimientos de tiempo real.

- La transacción **Control_Process** ejecuta el control en bucle cerrado de los servos, lo que se realiza mediante una tarea periódica con periodo y deadline de 5 ms.
- La transacción **Report_Process** obtiene de los servos y de los sensores la información sobre el status actual del robot y la transfiere a través del Bus CAN para actualizar el display. La actualización se realiza periódicamente con periodo y deadline de 100 ms.
- La transacción **Command_Process** atiende los comandos del operador y los traduce en secuencias temporizadas de consignas del robot. Es un proceso de naturaleza esporádica y su frecuencia máxima se limita a un comando por segundo.

Con los tres diagramas de secuencias que se muestran en las figuras de las páginas 10, 11 y 12 se describen gráficamente las tres transacciones del escenario Control_Teleoperado. En el diagrama de la página 13 se muestran los componentes de software que se desarrollan para cada uno de los procesadores. (Solo se incluyen los paquetes de especificación Ada de los componentes)

II.1 Logical_View: Arquitectura del Software.

La carpeta Logical View contien los diagramas de clases con el diseño lógico de la aplicación. Contiene la arquitectura y la definición de los componentes lógicos que la implementa.



Display_Refresher

Clase activa con sólo una instancia en el sistema. Su función es esperar la llegada de un mensaje de tipo Status por el bus CAN procedente del Controller y actualizar de acuerdo con los valores que contiene los campos del display de la GUI. Actualiza también la imagen que se representa del robot.

Private Attributes:

T : Task

Tarea que introduce el thread que se requiere para planificar la operación de actualización de los campos de datos e imágenes de la GUI.

Private Operations:

Actualize_Display () :

Actualiza los campos de información y display de la GUI, de acuerdo con el estado de operación en que se encuentra, de la información ya presente, y de la nueva información que llega procedente del Controller.

Actualize_Graphics () :

Actualiza el gráfico que representa el estado del robot en la GUI.

Command_Interpreter

Clase que incluye la atención y ejecución de los comandos que el operador puede ordenar a través de la GUI mediante eventos de teclado o de ratón.

Private Attributes:

T : Task

Introduce el thread en el que se planifica la atención y ejecución de los eventos que el operador genera en la GUI.

Private Operations:

Attend_Event () :

Atiende el evento que genera el operador a través de la GUI. Cuando el comando es sólo relativo al estado de la propia GUI, se resuelve directamente. Si el comando se refiere a operaciones del robot, hace uso del método Plan_Trajectory para generar la secuencia de consignas que debe ejecutar y envía un mensaje tipo Command con ellas al Controller.

Plan_Trajectory () :

Procedimiento que se invoca cuando el comando establecido por el operador se refiere a un movimiento del robot. Procesa el movimiento que se requiere y lo descompone en secuencias de consignas que deben ser cumplidas por los servos que controlan el movimiento del robot.

Display_Data

Objeto protegido que proporciona acceso seguro a los datos de la GUI a los dos threads que acceden concurrentemente a ella.

Public Operations:

Read () : Data
Acceso en modo de lectura a los datos de la GUI protegidos.

Write (D : Data) :
Acceso en modo de escritura a los datos protegidos de la GUI.

Station_Communication

Librería con métodos de acceso al bus CAN específicos de la aplicación y propios del terminal Station.

Public Operations:

Send_Command (C : Command) :
Procedimiento que gestiona el envío de un mensaje de tipo Command por el bus CAN. El procedimiento descompone el mensaje en paquetes y los encola en el driver del bus para que sean enviados de acuerdo con la disponibilidad del bus y con su prioridad.

Wait_Status () : Status
Función bloqueante que espera la llegada de un mensaje de Status. Cuando lo recibe, retorna el control con la información contenida en el mensaje.

Command_Manager

Espera la llegada por el bus CAN de un mensaje tipo Command, lo procesa y lo traduce en secuencias de consignas que almacena en Servos_Data.

Private Attributes:

T : Task
Introduce el thread dentro del que se planifican las actividades de interpretación del mensaje tipo Command recibido.

Private Operations:

Manage () :
Procedimiento que procesa el mensaje Command, lo traduce a secuencias de consignas de los servos y almacena estas en Servos_Data.

Reporter

Proceso periódico que se activa por el timer cada 100 ms, y cuya función es monitorizar la información relativa al estado del robot y de los sensores, así como enviarlos hacia la estación de monitorización a través de un mensaje tipo Status.

Private Attributes:

T : Task
Introduce el thread que se requiere para planificar la tarea de monitorización que lleva a cabo la clase.

Private Operations:

Report () :

Procedimiento que recupera la información, y la codifica en forma de un mensaje de tipo Status.

Servos_Controller

Tarea periódica que realiza el control de los servos del robot. Cada 5 ms lee las consignas y el estado de los servos, aplica el algoritmo de control y establece las entradas de los servos. Finaliza actualizando la información de acuerdo con los datos recibidos de los servos y de los sensores del robot.

Private Attributes:

T : Task

Introduce el thread que se necesita para planificar la tarea de control.

Private Operations:

Control_Servos () :

Invocada por una interrupción hardware del timer cada 5 ms, realiza la operación de control y monitorización de los servos. Este procedimiento invoca los procedimientos Control_Algorithms y Do_Control.

Control_Algorithms () :

Ejecuta el algoritmo de control de los servos.

Do_Control () :

Accede a los registros del hardware del robot para establecer las consignas de los servos y leer los sensores.

Servos_Data

Conjuntos de datos relativos a las consignas, al estado del robot y al estado de los sensores. Se implementa como un objeto protegido a fin de que sea seguro el acceso concurrente por los tres proceso que lo utilizan.

Public Operations:

Put () : Data

Acceso en modo de escritura a la información protegida.

Get (D : Data) :

Acceso en modo de lectura a los datos protegidos.

Controller_Communication

Librería con métodos de acceso al bus CAN específicos de la aplicación y propios del Controller.

Public Operations:

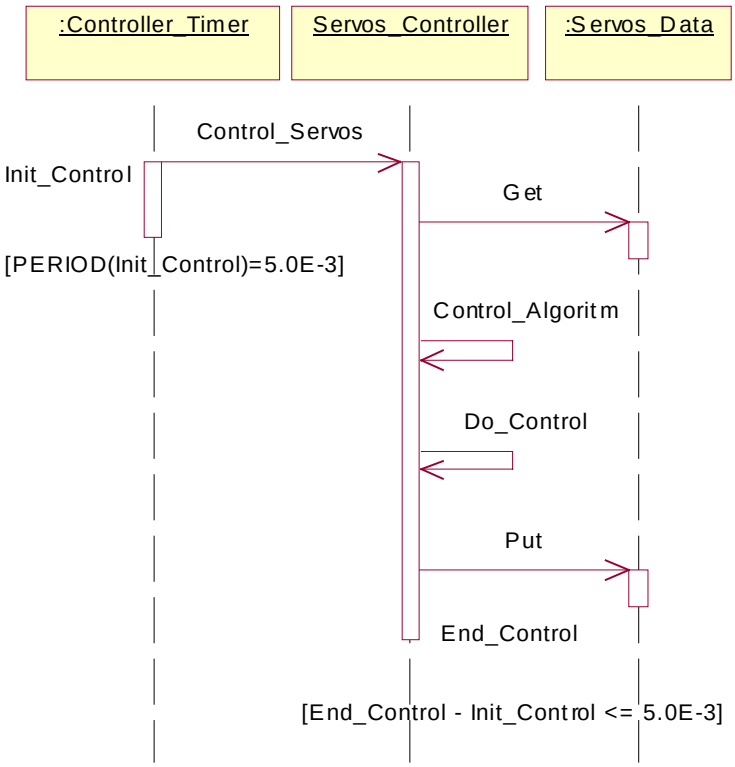
Send_Status (S : Status) :

Procedimiento que gestiona el envío de un mensaje de tipo Status por el bus CAN. El procedimiento descompone el mensaje en paquetes y los encola en el driver del bus para que sean enviados de acuerdo con la disponibilidad del bus y con su prioridad.

Wait_Command () : Command

Procedimiento bloqueante que espera a la llegada de un mensaje de tipo Command. Cuando este se recibe, retorna el control y la información transferida por el mensaje.

II.2 Logical View: Main transactions

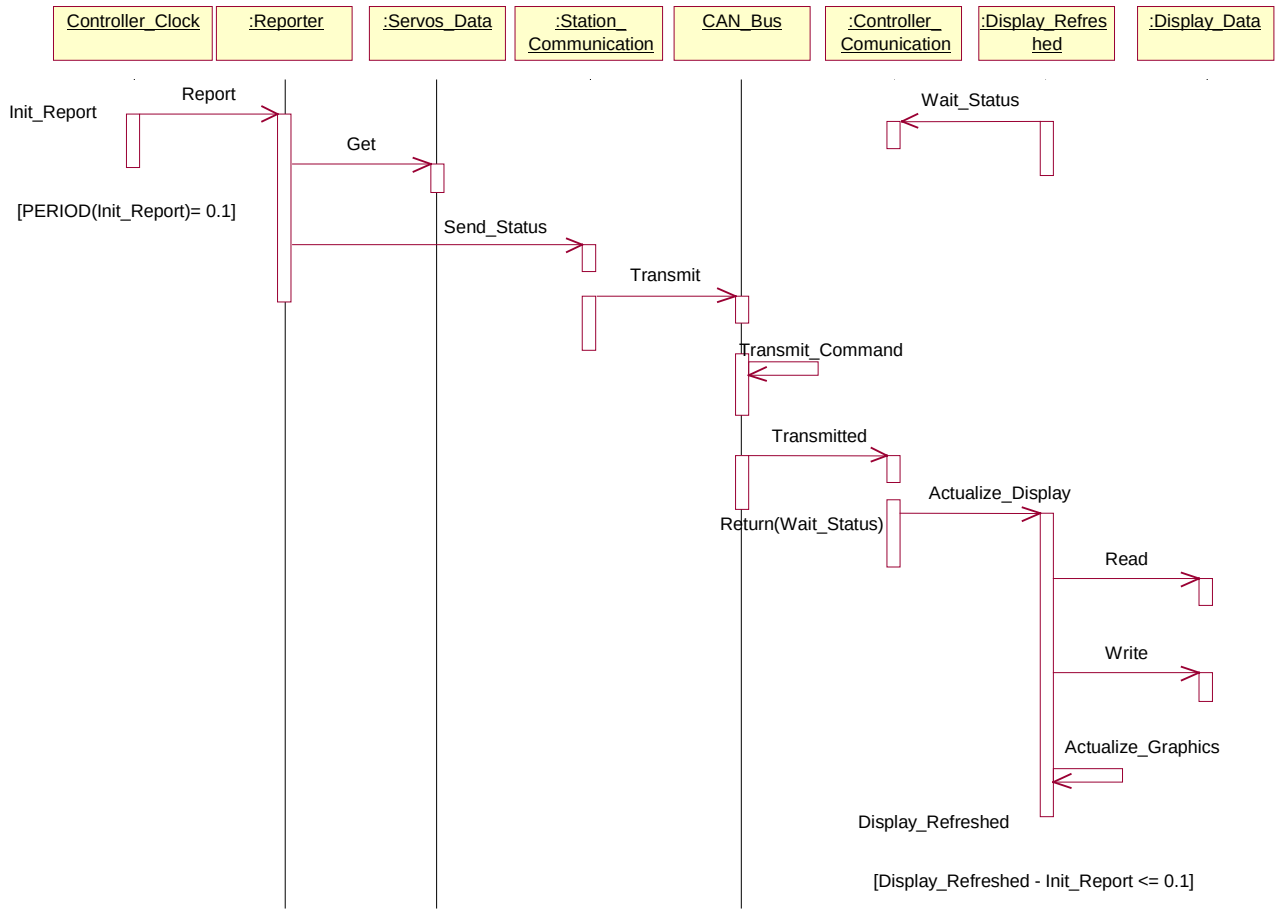


Transacción Control_Servos

Se inicia periódicamente cada 5 ms a través de una interrupción hardware (Init_Control) generada por el timer del Controller.

En el procesador Controller se lee el estado actual de los servos y las consignas que en ese instante deben satisfacerse, se ejecuta el algoritmo de control para calcular las entradas de los servos, y se transfieren a los registros hardware a través del Bus VME. Finalmente se actualiza la información relativa al estado de los servos.

Se requiere que finalice (End_Control) antes de que se inicie el siguiente ciclo de control (5 ms).



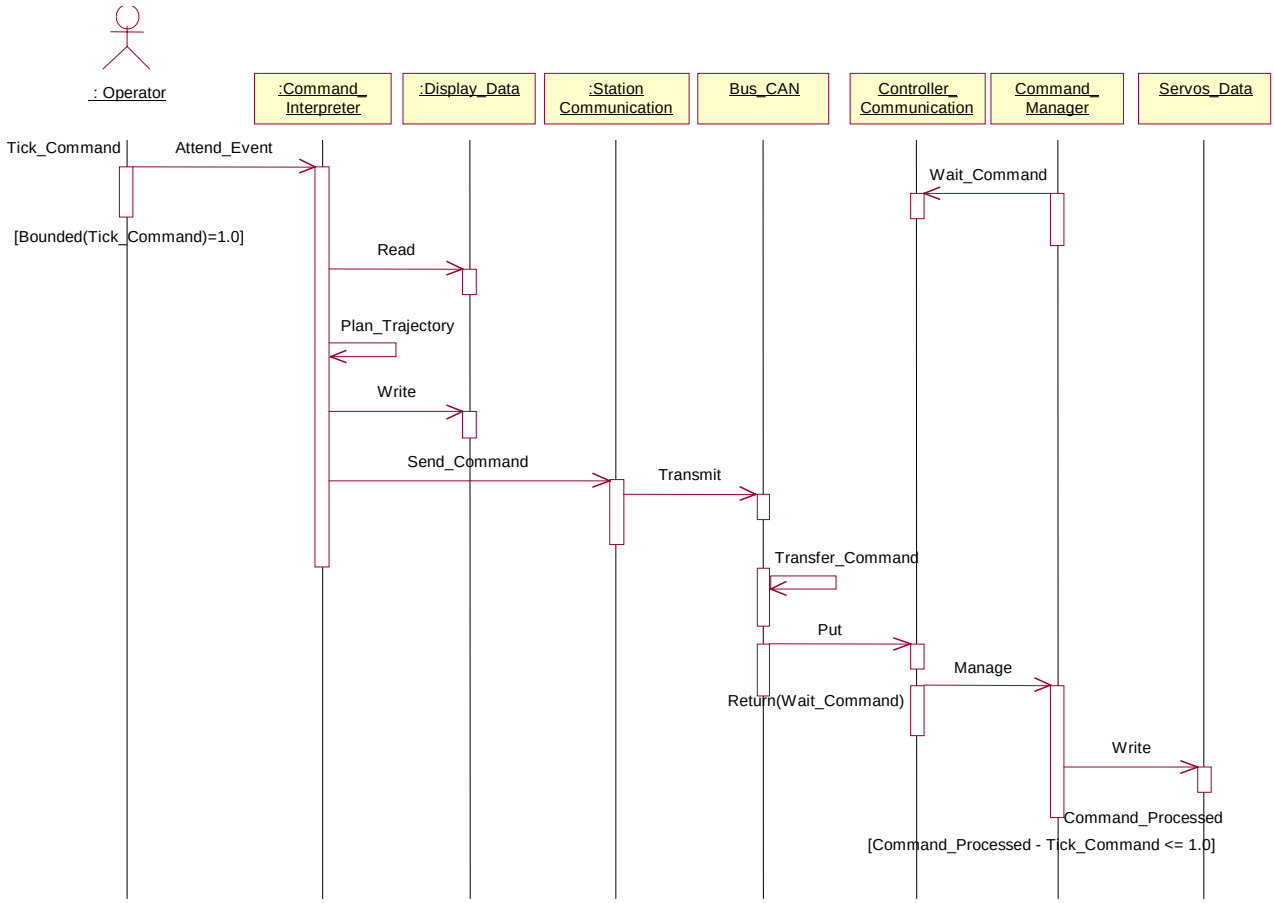
Transacción Report_Process

Se inicia periódicamente por interrupción hardware (Init_Report) del timer del Controller cada 100 ms.

En el procesador Controller codifica el estado del robot en un mensaje de status y lo envía por el bus CAN.

En el procesador Station se recibe el mensaje status y con él actualiza la información textual y gráfica de la GUI.

Se requiere que finalice con un deadline de 100 ms, esto es, antes de que se inicie el siguiente ciclo de refresco.



Transacción Command_Process

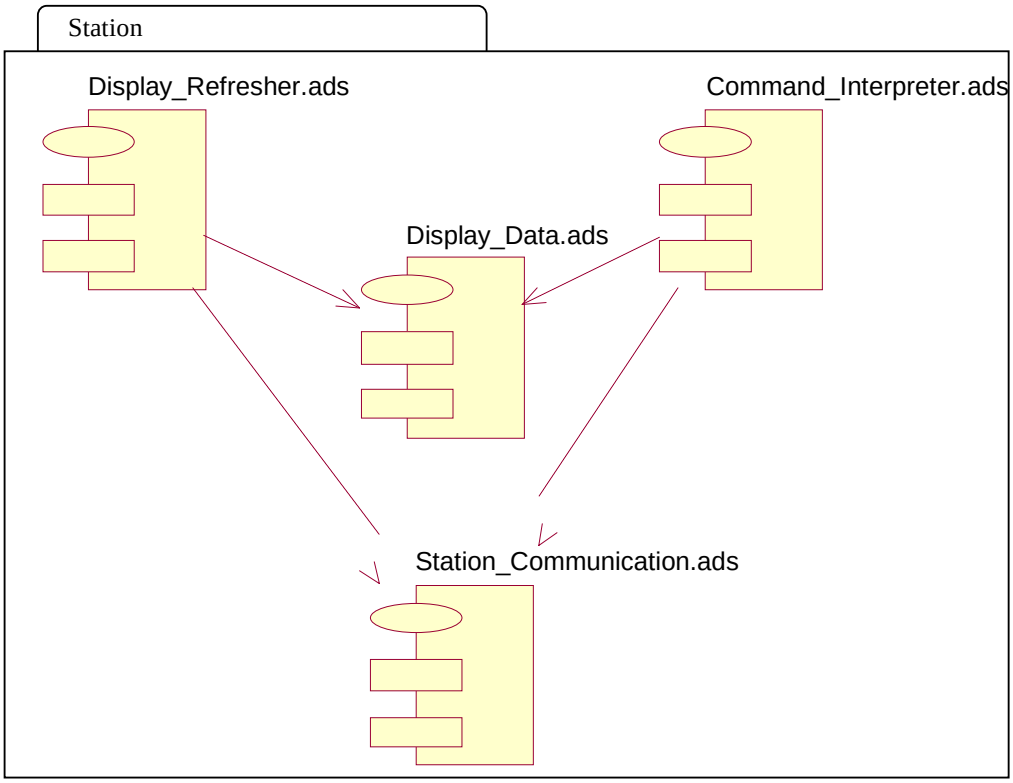
Se inicia cada vez que el operador ordena un comando a través de la GUI (Tick_Command). Este es un evento esporádico y a efecto de que sea planificable se limita su cadencia a 1 segundo.

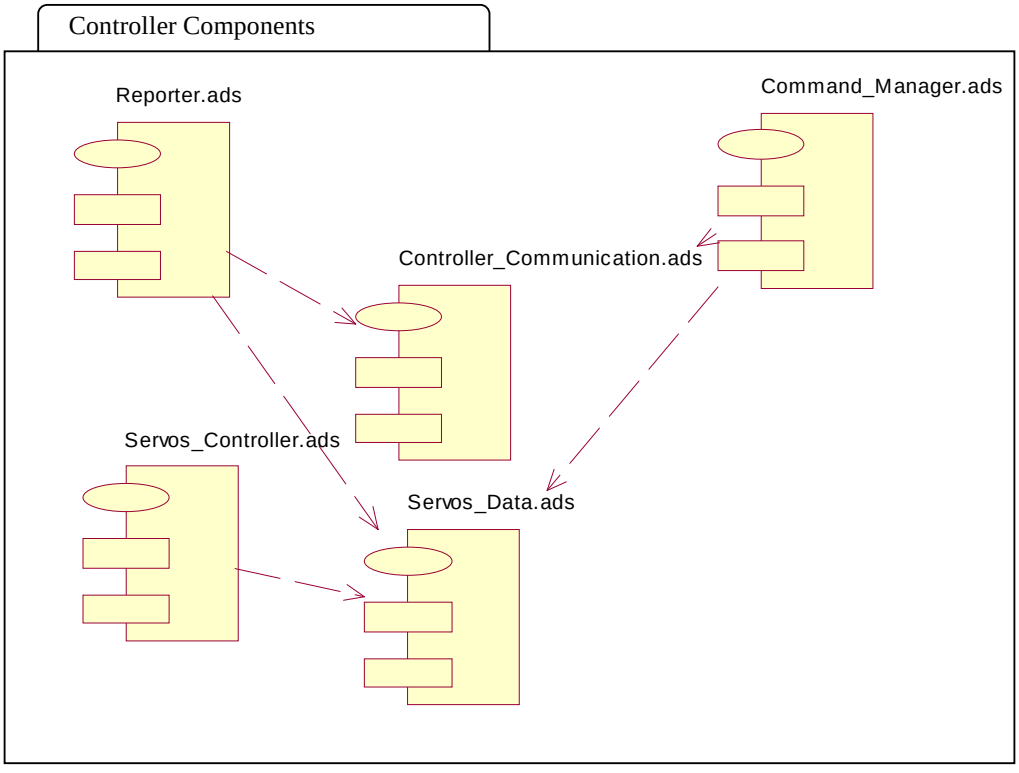
En el procesador Station se atiende el evento de la GUI, se interpreta de acuerdo con el estado de operación, se procesa para planificar el movimiento del robot, se modifica el estado del sistema si procede, y se envía el comando por el Bus CAN.

En el procesador Controller se interpreta el mensaje que se recibe y se encola como secuencia de consignas que en su tiempo deberán ser conseguidas.

Se requiere que finalice en 1 segundo, esto es, antes de que el operador pueda iniciar el siguiente comando.

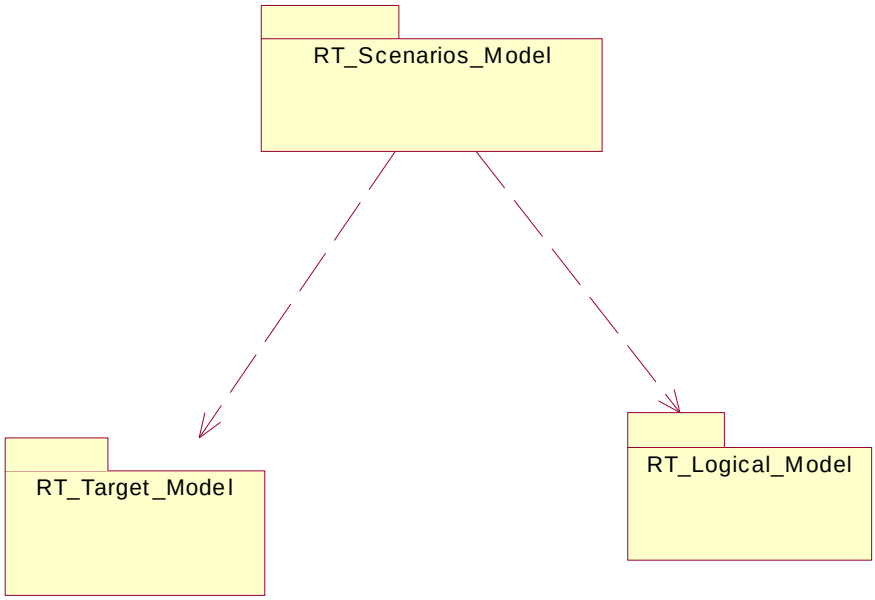
III COMPONENT VIEW: DECLARACION DE COMPONENTES .





IV MAST_RT_VIEW

Carpeta que contiene la Mast RT View que constituye el modelo de tiempo real de la aplicación.



La vista de tiempo real Mast modela el comportamiento de tiempo real de la aplicación . Se compone de:

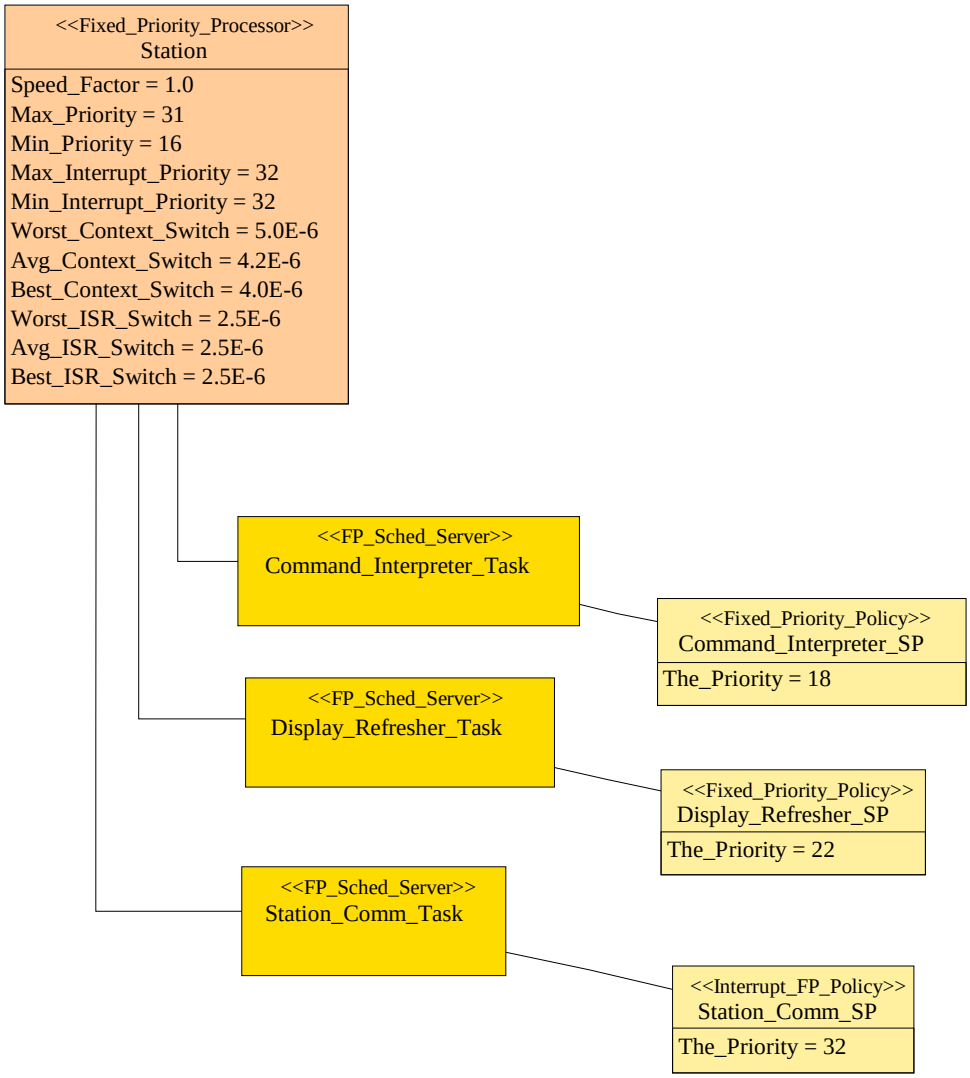
Modelo de tiempo real de la plataforma: Describe la capacidad de procesamiento de la plataforma hardware y software sobre la que se ejecuta la aplicación.

Modelo de tiempo real de los componentes lógicos: Describe la temporización y los requerimientos de procesamiento que requieren los componentes lógicos que constituyen la aplicación.

Modelo de los escenarios de tiempo real: Describe las secuencias de eventos externos o de temporización que conducen las transacciones que constituyen la aplicación, las secuencias de actividades que se desencadenan y los requerimientos temporales que establece la especificación de tiempo real.

IV.1 RT_Target_Model

Describe la capacidad de procesamiento de la plataforma sobre la que se ejecuta la aplicación. Describe la capacidad de procesamiento de los dos procesadores (Station y Processor) de la plataforma, los thread que se han introducido en cada uno de ellos para planificar las actividades y la capacidad de transferencia de información del canal de comunicación (Bus CAN).



Station

El procesador Station es una plataforma de tipo PC sobre la que opera un sistema operativo Window NT. Todos los procesos que se ejecutan en él lo hacen en el rango de prioridades de tiempo real (16..31), para los que se realiza una

planificación basada en prioridades fijas. Por ello, en la Mast_RT_View se modela mediante un componente del tipo Fixed_Priority_Processor

Private Attributes:

Speed_Factor : = 1.0
El Speed_Factor es 1.0 ya que la evaluación de los tiempos de los componentes lógicos se ha realizado en el propio procesador Station o en un procesador con una capacidad de procesamiento similar.

Max_Priority : = 31
En Windows NT las prioridades de "tiempo real" en las que la planificación se realiza por prioridades estáticas están en el rango 16..31.

Min_Priority : = 16
En Windows NT las prioridades de "tiempo real" en las que la planificación se realiza por prioridades estáticas están en el rango 16..31.

Max_Interrupt_Priority : = 32
En los sistema de tipo PC, las prioridades de interrupción son establecidas por la programación del dispositivo hardware PIC (Programable Interrupt Controller) y deben ser mapeadas a las prioridades del modelo MAST que son globales para todo el sistema. En este caso sólo interviene la interrupción que genera el controlador del Bus CAN y se le ha asignado arbitrariamente el valor de prioridad 32, que es un valor superior al del rango de las prioridades de los threads de la aplicación.

Min_Interrupt_Priority : = 32
En los sistema de tipo PC, las prioridades de interrupción son establecidas por la programación del dispositivo hardware PIC (Programable Interrupt Controller) y deben ser mapeadas a las prioridades del modelo MAST que son globales para todo el sistema. En este caso sólo interviene la interrupción que genera el controlador del Bus CAN y se le ha asignado arbitrariamente el valor de prioridad 32, que es un valor superior al del rango de las prioridades de los threads de la aplicación.

Worst_Context_Switch : = 5.0E-6
El cambio de contexto entre dos threads del procesador se realiza en un tiempo inferior a 5 us.

Avg_Context_Switch : = 4.2E-6
El cambio de contexto entre dos threads del procesador se realiza con un tiempo promedio de 4.2 us.

Best_Context_Switch : = 4.0E-6
El cambio de contexto entre dos threads del procesador se realiza siempre en un tiempo igual o superior a 4.0 us.

Worst_ISR_Switch : = 2.5E-6
El tiempo de cambio de contexto desde un thread de la aplicación a la rutina de atención de una interrupción es de 2.5 us.

Avg_ISR_Switch : = 2.5E-6
El tiempo de cambio de contexto desde un thread de la aplicación a la rutina de atención de una interrupción es de 2.5 us.

Best_ISR_Switch : = 2.5E-6
El tiempo de cambio de contexto desde un thread de la aplicación a la rutina de atención de una interrupción es de 2.5 us.

Command_Interpreter_Task

Thread que introduce la instancia de la clase Command_Interpreter. Este thread planifica la atención a los eventos generados por el operador sobre la GUI, ya sea por teclado o por ratón.

Command_Interpreter_SP

El Server planifica sus actividades siguiendo una política de prioridad fija y con una prioridad 18.

Private Attributes:

The_Priority : = 18

Display_Refresher_Task

Thread que introduce la instancia de la clase Display_Refresher. Se utiliza para planificar el procesado de los mensajes de tipo Status, que corresponde a las actualizaciones periódicas de la información textual y gráfica de la GUI.

Display_Refresher_SP

El Server planifica sus actividades siguiendo una política de prioridad fija y con una prioridad 22.

Private Attributes:

The_Priority : = 22

Station_Comm_Task

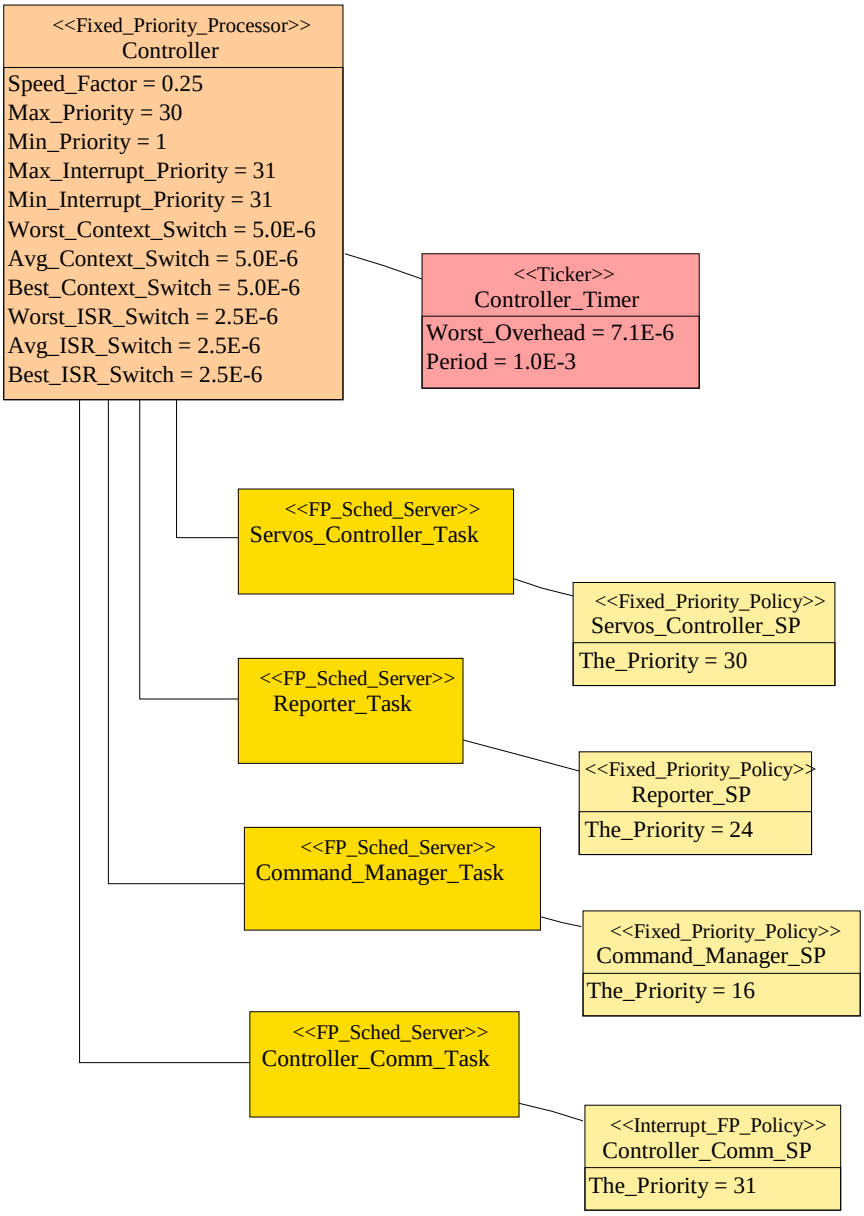
Thread en el que se atienden las interrupciones provocadas por el controlador hardware del Bus_CAN cada vez que llega un paquete. Es el thread sobre el que opera el driver del bus CAN.

Station_Comm_SP

El Server planifica sus actividades siguiendo una política de prioridad fija en el rango de prioridades de interrupción y con una prioridad 32.

Private Attributes:

The_Priority : = 32



Controller

El procesador Controller es un sistema embarcado que ejecuta un programa Ada'95 directamente sobre el núcleo mínimo Marte OS. En estas condiciones el procesador se modela mediante un componente Fixed_Priority_Processor.

Private Attributes:

Speed_Factor : = 0.25

La velocidad de procesamiento del procesador es 4 veces menor que la del procesador (p.e. Station) en que se han estimado los tiempos de ejecución de los componentes lógicos.

Max_Priority : = 30
En el entorno Marte OS, con programas Ada'95 desarrollados con el compilador GNAT (sin modificar) el rango de aplicaciones de aplicación es 1..30.

Min_Priority : = 1
En el entorno Marte OS, con programas Ada'95 desarrollados con el compilador GNAT (sin modificar) el rango de aplicaciones de aplicación es 1..30.

Max_Interrupt_Priority : = 31
En el entorno Marte OS, con programas Ada'95 desarrollados por el compilador GNAT, los manejadores de interrupción se ejecutan con una prioridad de interrupción 31.

Min_Interrupt_Priority : = 31
Los manejadores de interrupción se ejecutan con una prioridad de interrupción 31.

Worst_Context_Switch : = 5.0E-6
El tiempo de conmutación entre dos threads de aplicación es aproximadamente constante y de valor 5.0 us.

Avg_Context_Switch : = 5.0E-6
El tiempo de conmutación entre dos threads de aplicación es aproximadamente constante y de valor 5.0 us.s.

Best_Context_Switch : = 5.0E-6
El tiempo de conmutación entre dos threads de aplicación es aproximadamente constante y de valor 5.0 us.

Worst_ISR_Switch : = 2.5E-6
El tiempo de conmutación entre un thread de aplicación y una rutina de atención de interrupción es aproximadamente constante y de valor 2.5 us.

Avg_ISR_Switch : = 2.5E-6
El tiempo de conmutación entre un thread de aplicación y una rutina de atención de interrupción es aproximadamente constante y de valor 2.5 us.

Best_ISR_Switch : = 2.5E-6
El tiempo de conmutación entre un thread de aplicación y una rutina de atención de interrupción es aproximadamente constante y de valor 2.5 us.

Controller_Timer

El procesador Controller dispone de un timer hardware que le proporciona el reloj interno y la capacidad de temporización. Este reloj es de tipo Ticker, con periodo 1 ms. Este componente modela la carga de procesamiento que se requiere del procesador Controller para atender en background y con alta prioridad las interrupciones del timer.

Private Attributes:

Worst_Overhead : = 7.1E-6

Tiempo que se requiere del procesador para atender una interrupción del timer.

Period : = 1.0E-3
Periodo del timer ticker.

Servos_Controller_Task

Modela el thread que introduce la tarea Ada del objeto activo Servos_Controller. Es una tarea periodica de 5.0E-3 segundos de periodo

Servos_Controller_SP

Esta tarea es planificada de acuerdo con una política de planificación de prioridad estática fija y con prioridad 30.

Private Attributes:

The_Priority : = 30

Reporter_Task

Modela el thread que introduce la tarea Ada del objeto activo Reporter. Es una tarea periódica de 0.1 segundos de periodo

Reporter_SP

Esta tarea es planificada de acuerdo con una política de planificación de prioridad estática fija y con prioridad 24.

Private Attributes:

The_Priority : = 24

Command_Manager_Task

Modela el thread que introduce la tarea Ada del objeto activo Command_Manager. Es una tarea gobernada por evento. Se ejecuta cada vez que se recibe un mensaje tipo Command por el bus CAN.

Command_Manager_SP

Esta tarea es planificada de acuerdo con una política de planificación de prioridad estática fija y con prioridad 16.

Private Attributes:

The_Priority : = 16

Controller_Comm_Task

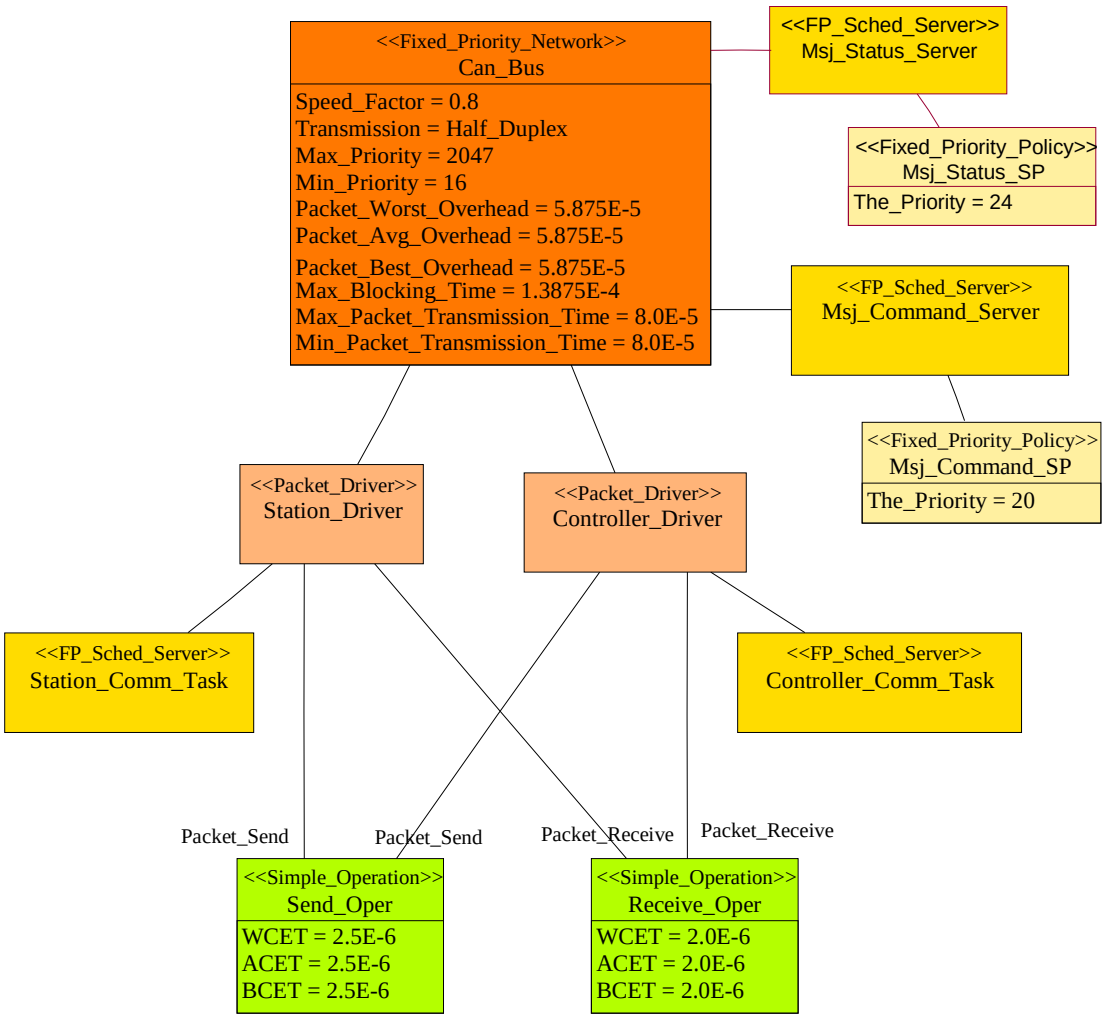
Modela el thread en que se ejecuta la rutina de interrupción que atiende el driver del bus CAN. Es una tarea gobernada por eventos y se ejecuta cada vez que se produce una interrupción hardware en el driver del bus CAN.

Controller_Comm_SP

Es una rutina de interrupción y es atendida con prioridad estática de nivel 31.

Private Attributes:

The_Priority : = 31



Can_Bus

El Bus_CAN es el canal de comunicación que comunica el procesador Station y el procesador Controller. Es un canal de tipo half_duplex, esto es, con capacidad de transferir mensajes en ambos sentidos, pero sin que puedan coincidir en el tiempo. Es un bus orientado a paquetes y cada paquete puede estar compuesto de 1 a 8 bytes. La velocidad de transferencia del bus es de 100 Kbyte/s (800000 bits/seg).

La transferencia de los mensajes es priorizada, esto es, no se transfiere ningún paquete de un mensaje de una prioridad dada si aún quedan pendientes de transferencia paquetes de un mensaje de mayor prioridad.

El modelo del canal de comunicación se describe mediante componentes Mast de los tipos Fixed_Priority_Network, Packet_Driver y FP_Sched_Server.

El componente **CAN_Bus** es un Fixed_Priority_Network que describe la capacidad del canal.

Private Attributes:

Speed_Factor : = 0.8
El factor de velocidad de transferencia del canal es 0.8, ya que tiene una velocidad de transmisión de 100 KBytes/s y los tiempos de transferencia se han estimado sobre un canal equivalente de 1 Mbit/s.

Transmission : = Half_Duplex
El Bus CAN opera en modo Half Duplex, esto es, con capacidad de transferir mensajes en ambos sentidos pero sin que puedan coincidir en el tiempo.

Max_Priority : = 2047
Los paquetes se envían por el Bus CAN con prioridades en el rango de 16 a 2047.

Min_Priority : = 16
Los paquetes se envían por el Bus CAN con prioridades en el rango de 16 a 2047.

Packet_Worst_Overhead : = 58.75
Tiempo extra que se requiere para enviar un mensaje por el canal, bien por necesidad de intercambio de mensajes de control o por la inclusión de cabecera de control en los paquetes. En esta aplicación la comunicación es punto a punto y se transfiere un único tipo de mensaje en cada dirección, por lo que el overhead corresponde tan sólo a los 47 bits extra del protocolo básico del bus can.

Packet_Avg_Overhead : = 58.75
En esta aplicación la comunicación es punto a punto y se transfiere un único tipo de mensaje en cada dirección, por lo que el overhead corresponde tan sólo a los 47 bits extra del protocolo básico del bus can.

Packet_Best_Overhead : = 58.75
En esta aplicación la comunicación es punto a punto y se transfiere un único tipo de mensaje en cada dirección, por lo que el overhead corresponde tan sólo a los 47 bits extra del protocolo básico del bus can.

Max_Blocking_Time : = 1.3875E-4
Tiempo máximo que el canal puede permanecer ocupado sin posibilidad de ser interrumpido. En este caso, el Bus CAN opera con una velocidad de 100 KBytes/s la longitud máxima del paquete y las cabeceras de control es de 8 bytes + 47 bits, por lo que el tiempo de bloqueo es de 138.75 us.

Max_Packet_Transmission_Time : = 8.0E-5
Tiempo máximo que el Bus CAN utiliza para transferir un paquete. En este caso, el paquete máximo es de 8 bytes y la velocidad de transferencia es de 100 Kbytes/s por lo que el su valor es de 80 us.

Min_Packet_Transmission_Time : = 8.0E-5
Tiempo mínimo que el Bus CAN utiliza para transferir un paquete. En este caso, el paquete mínimo es de 1 bytes y la velocidad de transferencia es de 100 KBytes/s por lo que su valor sería de 10 us.

Sin embargo en esta aplicación todos los paquetes que se utilizan son de 8 bytes, por lo que se considera que el tiempo de transmisión de paquete es de 80 us. Con ello se evita un análisis excesivamente pesimista.

Msj_Status_Server

Servidor de planificación de paquetes dentro del Bus CAN que se utiliza para planificar los paquetes correspondientes a los mensajes de tipo Status.

Msj_Status_SP

Los paquetes se planifican de acuerdo con una política de prioridad fija y con una prioridad de 24

Private Attributes:

The_Priority : = 24

Msj_Command_Server

Servidor de planificación de paquetes dentro del Bus CAN que se utiliza para planificar los paquetes correspondientes a los mensajes de tipo Command.

Msj_Command_SP

Los paquetes se planifican de acuerdo con una política de prioridad fija y con una prioridad de 20.

Private Attributes:

The_Priority : = 20

Controller_Driver

El driver modela el tiempo de procesamiento que se requiere del procesador Controller para atender las interrupciones que provoca la gestión de los paquete que se envian o reciben por el Bus_CAN. Como se trata de una comunicación orientada a paquetes se modela con un componente del tipo Packet_Driver.

Station_Driver

El driver modela el tiempo de procesamiento que se requiere del procesador Station para atender las interrupciones que provoca la gestión de los paquete que se envían o reciben por el Bus_CAN. Como se trata de una comunicación orientada a paquetes se modela con un componente del tipo Packet_Driver.

Receive_Oper

Modela el tiempo que emplea el procesador en atender la recepción de un paquete por el Bus CAN. Los tiempos son normalizados, esto es, están referidos a un procesador de Speed_Factor 1.0.

Private Attributes:

WCET : = 2.0E-6

El tiempo normalizado de peor caso que requiere la recepción de un paquete por el bus CAN es aproximadamente constante y de valor 2.0E-6.

ACET : = 2.0E-6

El tiempo normalizado medio que requiere la recepción de un paquete por el bus CAN es aproximadamente constante y de valor 2.0E-6.

BCET : = 2.0E-6
En el mejor caso, el tiempo normalizado que requiere la recepción de un paquete por el bus CAN es aproximadamente constante y de valor 2.0E-6.

Send_Oper

Modela el tiempo que emplea el procesador en activar el envío de un paquete por el Bus CAN. Los tiempos son normalizados, esto es, están referidos a un procesador de Speed_Factor 1.0.

Private Attributes:

WCET : = 2.5E-6
El tiempo normalizado de peor caso que requiere el envío de un paquete por el bus CAN es aproximadamente constante y de valor 2.5E-6.

ACET : = 2.5E-6
El tiempo normalizado medio que requiere el envío de un paquete por el bus CAN es aproximadamente constante y de valor 2.5E-6.

BCET : = 2.5E-6
En el mejor caso, el tiempo normalizado que requiere el envío de un paquete por el bus CAN es aproximadamente constante y de valor 2.5E-6.

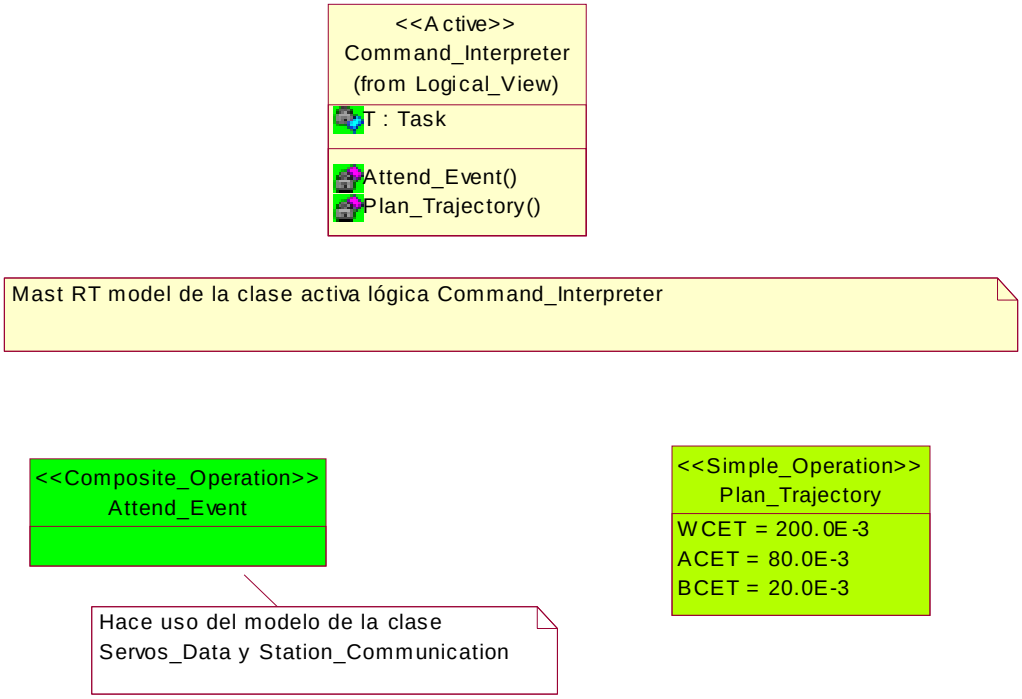
IV.2 RT_Logical_Model

El modelo de tiempo real de los componentes lógicos modela el comportamiento temporal de los componentes funcionales (clases, métodos, procedimientos, operaciones, etc.) que están definidos en el sistema y cuyos tiempos de ejecución condicionan el cumplimiento de los requerimientos temporales definidos en los escenarios de tiempo real que van a analizarse. El modelo de cada componente lógico describe los dos aspectos que condicionan su tiempo de ejecución: el tiempo que requiere la ejecución de su código, que es función de la complejidad de los algoritmos que contiene y los bloqueos que puede sufrir su ejecución como consecuencia de que necesita acceder en régimen exclusivo a recursos que también son requeridos por otros componentes lógicos que se ejecutan concurrentemente con él.

El modelo de tiempo real de los componentes lógicos se establece con las siguientes características:

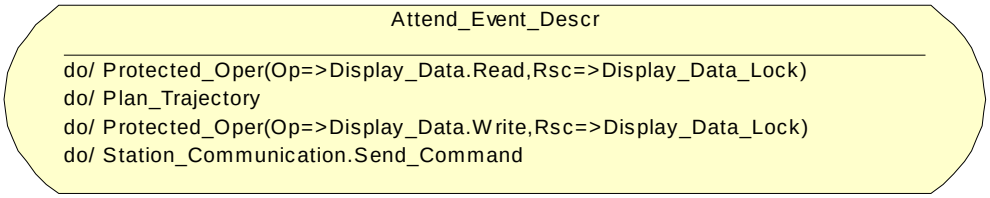
Los tiempos de ejecución de las operaciones se definen normalizados, esto es, se formulan con parámetros que son independientes de la plataforma en que se van a ejecutar.
El modelo de interacción entre componentes lógicos se expresa de forma parametrizada, identificando los recursos que son potenciales causas de bloqueos (Shared_Resource) y dejando hasta la descripción del escenario la declaración de los componentes lógicos concretos con los que va a interferir.
El modelo de tiempo real de los componentes lógicos, se formula con una modularidad paralela a la modularidad que en la vista lógica ofrecen los componentes lógicos que se modelan.

En los siguientes diagramas se describen los modelos de tiempo real de los componentes lógicos de esta aplicación. A efecto meramente explicativo, se adjunta a cada diagrama del RT_Logical_Model la clase lógica que se modela en el diagrama.



Attend_Event

Modelo del procedimiento Attend_Event de la clase lógica Command_Interpreter. Es una operación compuesta descrita por su diagrama de actividad.

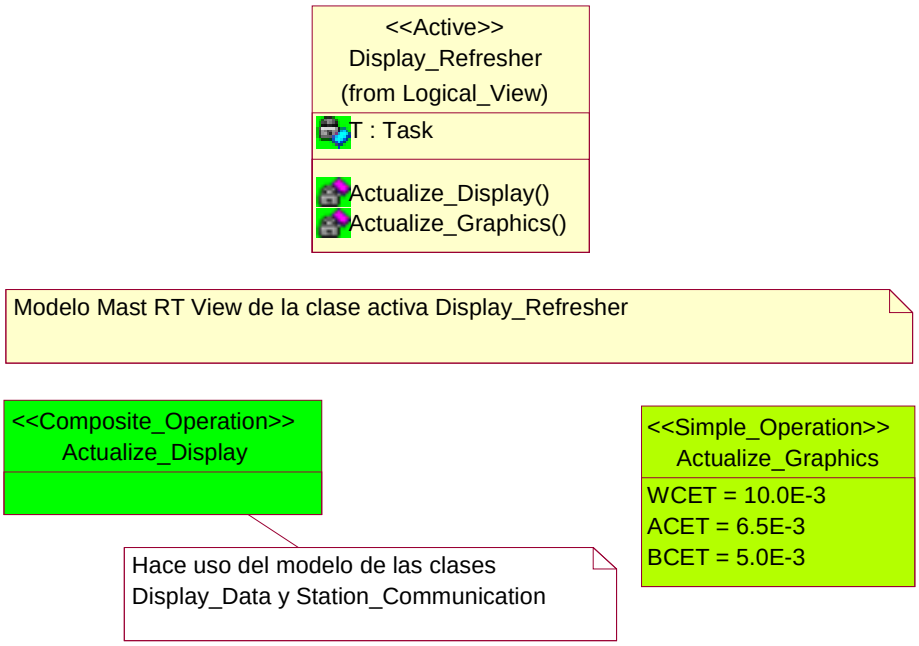


Plan_Trajectory

Modelo de la operacion de procesado de un mensaje Command para generar la secuencia de consignas del robot que le corresponde. Se modela mediante una operación simple, cuya duración es muy diferente (20,0 ms a 200.0 ms) dependiendo de la naturaleza del comando.

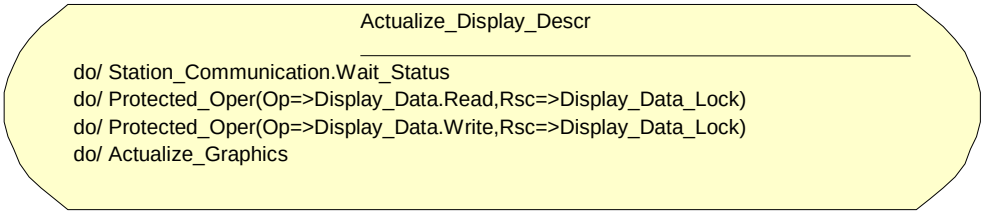
Private Attributes:

WCET : = 200.0E-3 **ACET : = 80.0E-3** **BCET : = 20.0E-3**



Actualize_Display

Modela la secuencia de operaciones que se ejecutan tras ser recibido por el CAN Bus un mensaje de Status. Estas operaciones están destinadas a actualizar la información del GUI. Se modela como una operación compuesta descrita mediante un diagrama de actividad.

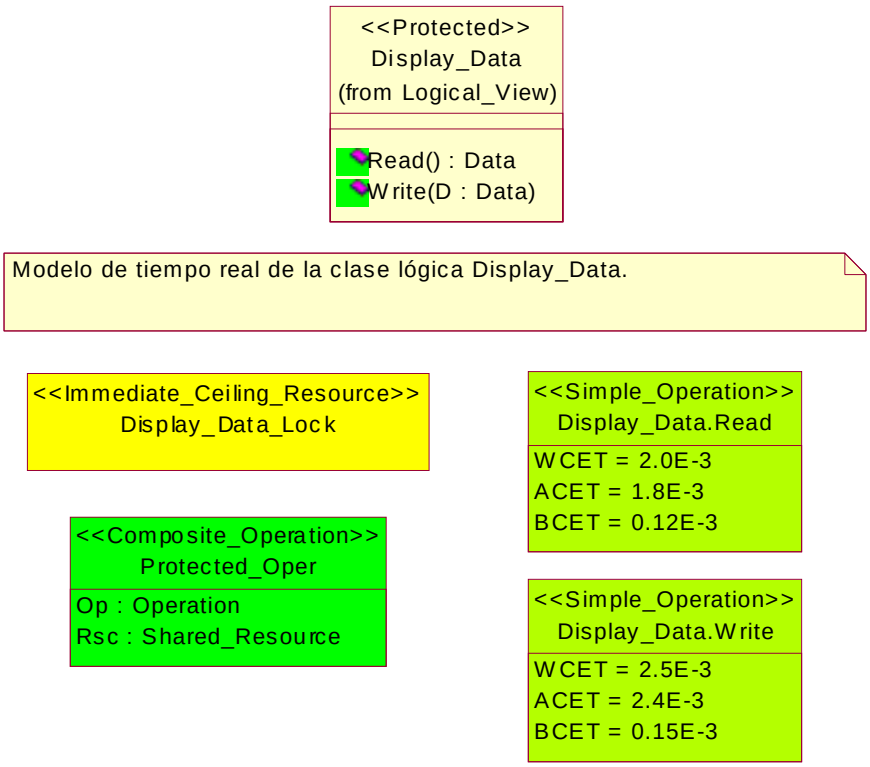


Actualize_Graphics

Modela la actualización de la información gráfica presente en la GUI.

Private Attributes:

WCET : = 10.0E-3 **ACET : = 6.5E-3** **BCET : = 5.0E-3**

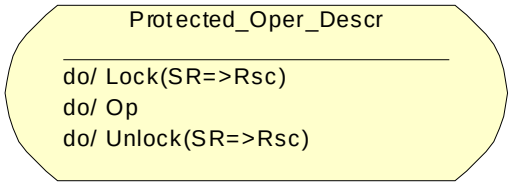


Display_Data_Lock

Representa el recurso asociado al objeto protegido, y que debe ser accedido por cualquier operación protegida sobre el objeto antes de poder ejecutarse.

Protected_Oper

Operación compuesta parametrizada que se utiliza para invocar una operación de cualquier objeto protegido. El parámetro Op representa la operación que se invoca.. Rsc representa el recurso compartido que debe ser accedido en régimen exclusivo antes de que se ejecute la operación. Al concluir la ejecución de la operación, libera el recurso.



Private Attributes:

Op : Operation
Operación que se ejecuta en modo protegido.

Rsc : Shared_Resource
Recurso compartido respecto del que se ejecuta la operación protegida.

Display_Data.Read

Modela el acceso al recurso en modo lectura para adquirir y retornar cierta información propia del objeto protegido.

Private Attributes:

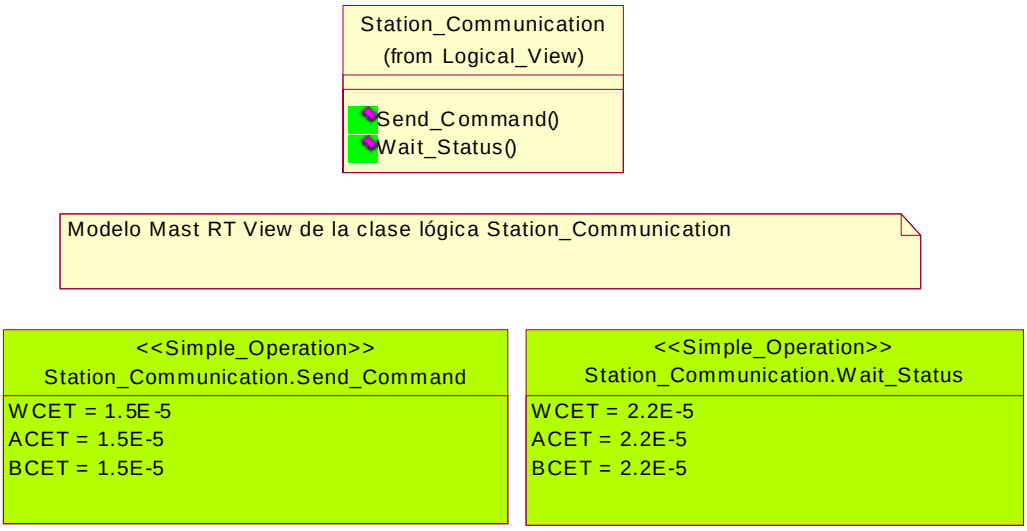
WCET : = 2.0E-3 **ACET : = 1.8E-3** **BCET : = 0.12E-3**

Display_Data.Write

Modela el acceso al recurso en modo escritura para actualizar información del objeto protegido.

Private Attributes:

WCET : = 2.5E-3 **ACET : = 2.4E-3** **BCET : = 0.15E-3**



Station_Communication.Send_Command

Operación simple que modela el tiempo que tarda la aplicación en transferir un mensaje al controlador del Bus_CAN a fin de que el controlador planifique la transferencia de los paquetes en que se descompone. Esta operación modela la operación que se ejecuta en el thread de la actividad que invoca su transferencia. Lleva a cabo la descomposición del mensaje en paquetes y la inserción de estos en la lista de paquetes pendientes de ser transferidos. La transferencia de los paquetes por el canal la realiza el driver en background y en su propio thread.

Public Attributes:

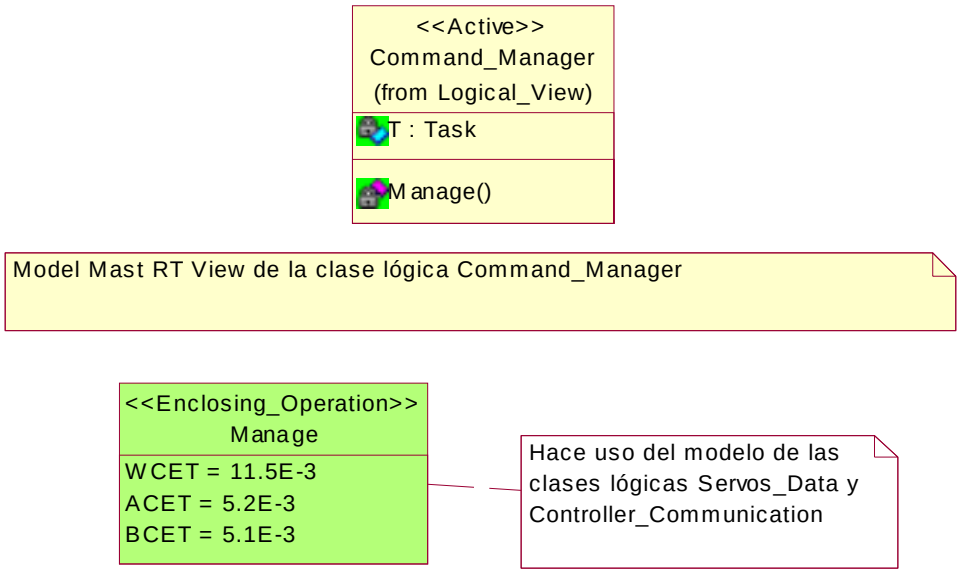
WCET : = 1.5E-5 **ACET : = 1.5E-5** **BCET : = 1.5E-5**

Station_Communication.Wait_Status

El procedimiento lógico Wait_Command es un procedimiento bloqueante, que suspende el thread que lo invoca hasta que el driver detecta la llegada de un mensaje de tipo Command completo. Esta operación modela la actividad que se ejecuta en el thread que lo invoca cuando el conjunto de los paquetes que constituyen el mensaje ha sido recibido. Básicamente modela el procesamiento que se requiere para reconstruir el mensaje a partir de los paquetes recibidos.

Public Attributes:

WCET : = 2.2E-5 **ACET : = 2.2E-5** **BCET : = 2.2E-5**

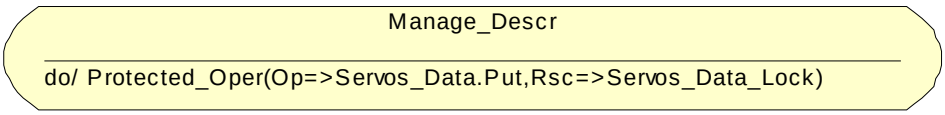


Manage

Modela la ejecución del procedimiento Manage de la clase Command_Manager. Se modela como una Enclosing_Operation, ya que la estimación de su duración se ha realizado sobre toda la secuencia de operaciones que implica:

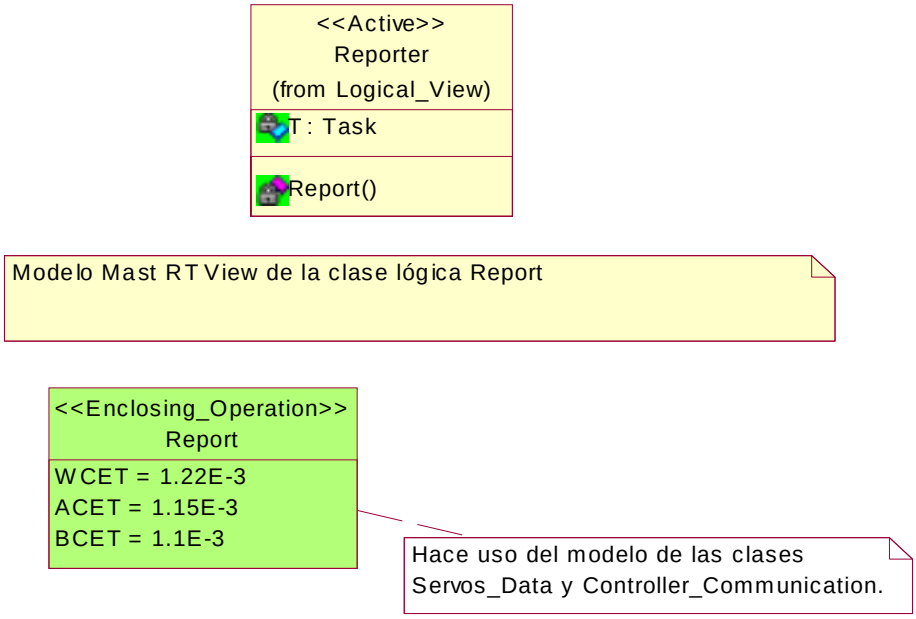
- Controller_Communication.Wait_Command
- Command_Process
- Servos_Data.Put

Sin embargo, como la descripción de la temporización se ha establecido a nivel global de toda la secuencia, sólo se deben incluir en el diagrama de actividad que describe la operación aquellas que pueden dar lugar a bloqueo por hacer uso de un recurso compartido (Servos_Data.Put).



Private Attributes:

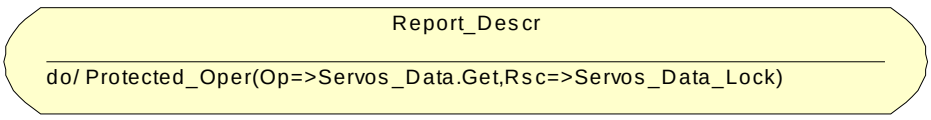
WCET : = 11.5E-3 **ACET : = 5.2E-3** **BCET : = 5.1E-3**



Report

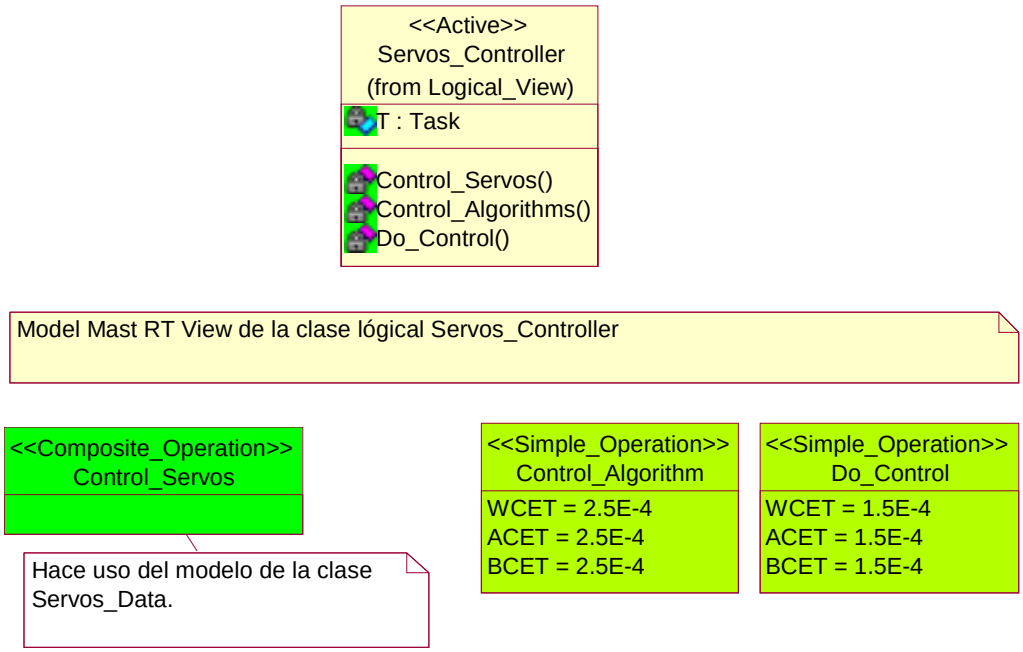
Modela la operación que se produce cada 100 ms que lee el estado de los servos, el estado de los sensores y transfiere un mensaje Status con la información monitorizada hacia la estación de teleoperación para tener actualizada la GUI.

Se modela como una Enclosing_Operación, en la que el tiempo de ejecución se ha evaluado para el conjunto de las operaciones que conlleva su ejecución. Por esta razón en su diagrama de actividad sólo se describen las operaciones que implican acceso o liberación de recursos compartidos.



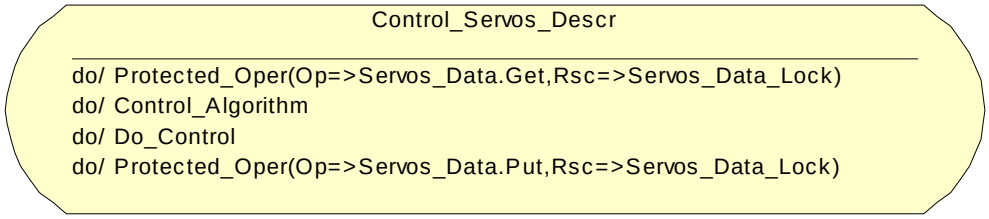
Private Attributes:

WCET : = 1.22E-3 **ACET : = 1.15E-3** **BCET : = 1.1E-3**



Control_Servos

Modela la operación que se ejecuta cada 5 ms para controlar los servos y leer su estado y el de los sensores. Se modela como una Composite_Operation que ejecuta:



Control_Algorithm

Modela el procedimiento que ejecuta el algoritmo de control en el que se calcula la entrada de los servos en función de las consignas y del estado en que se encuentran.

Public Attributes:

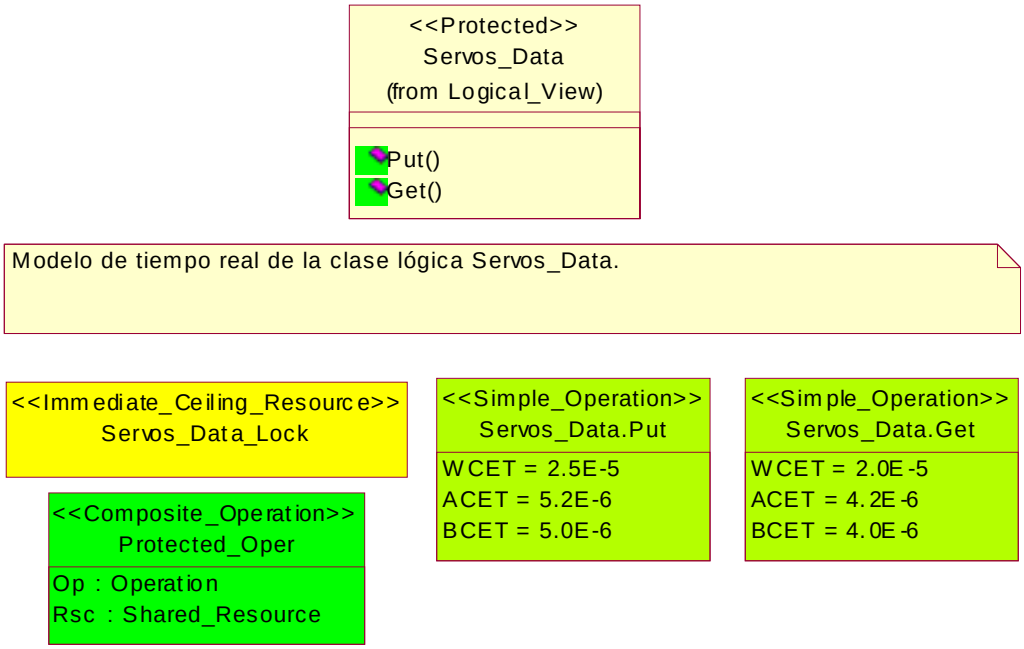
WCET : = 2.5E-4 **ACET : = 2.5E-4** **BCET : = 2.5E-4**

Do_Control

Modela el procedimiento que accede por el Bus VME a los registros del controlador hardware de los servos del robot y de los sensores. Este procedimiento no sufre ninguna contención y se modela mediante una operación simple.

Public Attributes:

WCET : = 1.5E-4 **ACET : = 1.5E-4** **BCET : = 1.5E-4**



Servos_Data_Lock

Representa el recurso compartido que debe ser accedido por cualquier operación protegida que accede al objeto Sevos_Data.

Servos_Data.Get

Modela el acceso en modo lectura para obtener información del objeto protegido Servos_Data.

Private Attributes:

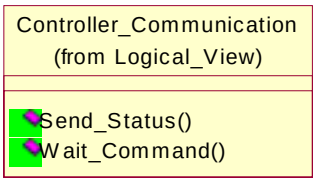
WCET : = 2.0E-5 **ACET : = 4.2E-6** **BCET : = 4.0E-6**

Servos_Data.Put

Representa la operación de acceso en modo escritura sobre el objeto protegido Servos_Data.

Private Attributes:

WCET : = 2.5E-5 **ACET : = 5.2E-6** **BCET : = 5.0E-6**



Modelo Mast RT View de la clase Controller_Communicaction

<<Simple_Operation>> Controller_Communication.Send_Status
WCET = 3.1E-5 ACET = 3.1E-5 BCET = 3.1E-5

<<Simple_Operation>> Controller_Communication.Wait_Command
WCET = 0.20E-5 ACET = 0.2E-5 BCET = 0.2E-5

Controller_Communication.Send_Status

Operación simple que modela el envío de un mensaje a través de la librería de control del Bus CAN. La operación se ejecuta en el thread de la actividad que invoca su transferencia y consiste en la descomposición del mensaje en paquetes y la inserción de estos en la lista de paquetes pendientes de ser transferidos. La transferencia de los paquetes por el canal la realiza el driver en background y en su propio thread.

Public Attributes:

WCET : = 3.1E-5 **ACET : = 3.1E-5** **BCET : = 3.1E-5**

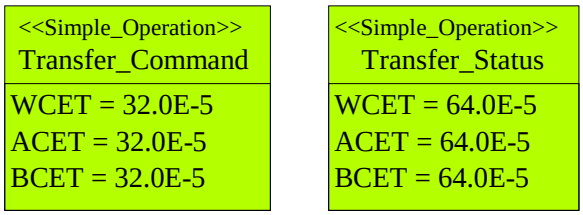
Controller_Communication.Wait_Command

El procedimiento lógico Wait_Status es un procedimiento bloqueante, que suspende el thread que lo invoca hasta que el driver detecta la llegada de un mensaje completo de Status. Esta operación modela la actividad que se ejecuta en el thread que lo invoca cuando el conjunto de los paquetes que constituyen el mensaje ha sido recibido. Básicamente modela el procesamiento que se requiere para reconstruir el mensaje a partir de los paquetes recibidos.

Public Attributes:

WCET : = 0.20E-5 **ACET : = 0.2E-5** **BCET : = 0.2E-5**

Operaciones de transferencia de información que ejecuta el network CAN_Bus. (No representan la ejecución de un código.



Transfer_Command

Representa la transmisión fisica del mensaje Command a través del Bus_CAN. La operación representa el tiempo que el canal está ocupado por la transferencia de los paquetes en que se descompone el mensaje Command.

Como el mensaje Command está compuesto por 40 bytes, y la velocidad de transferencia de un canal de Speed_Factor 1.0 es de 1 MBit/s, se requieren 320 us para su transferencia.

Private Attributes:

WCET : = 32.0E-5 **ACET : = 32.0E-5** **BCET : = 32.0E-5**

Transfer_Status

Representa la transmisión física del mensaje Status a través del Bus_CAN. La operación representa el tiempo que el canal está ocupado por la transferencia de los paquetes en que se descompone el mensaje Status.

Como el mensaje Status está compuesto por 80 bytes, y la velocidad de transferencia de un canal de Speed_Factor=1.0 es 1 MBit/s, se requieren 640 us para su transferencia.

Private Attributes:

WCET : = 64.0E-5 **ACET : = 64.0E-5** **BCET : = 64.0E-5**

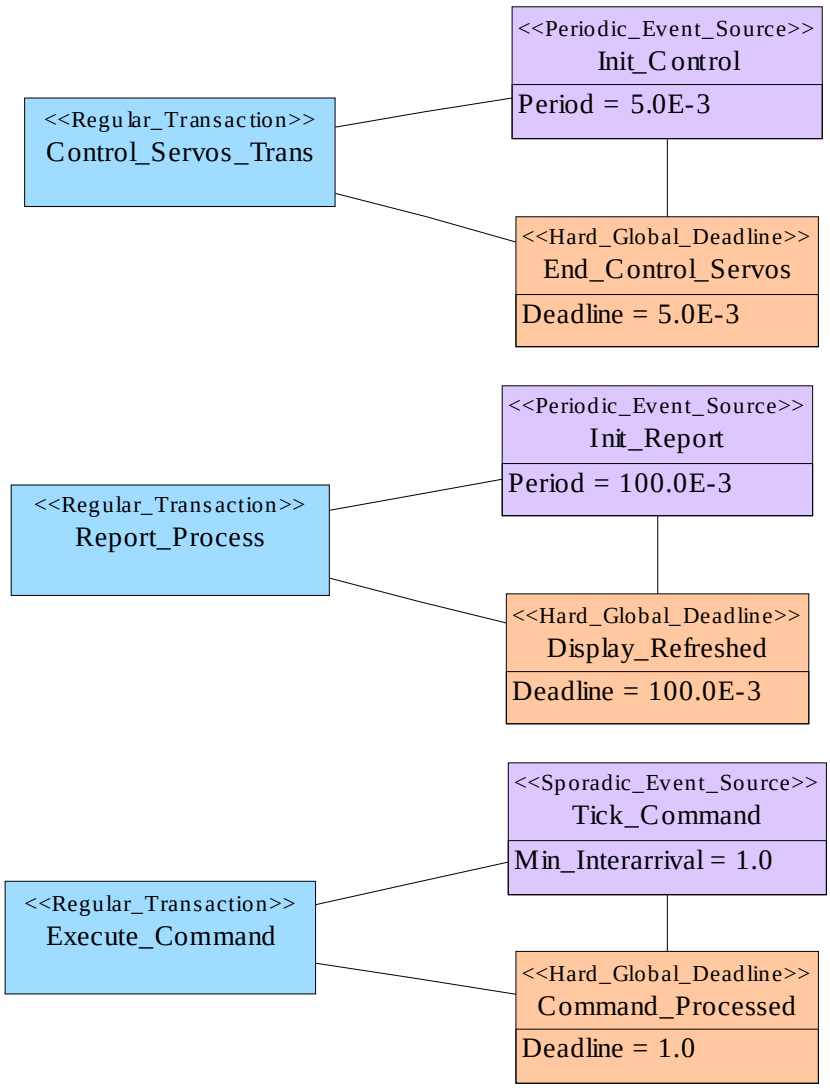
IV.3 RT_Scenarios_Model

Contiene los escenarios de tiempo real que son los modos o configuraciones de operación hardware/software para los que el sistema tiene definidos requerimientos de tiempo real. En este sistema sólo hay definido un escenario de tiempo real denominado Control_Teleoperado.

Control_Teleoperado

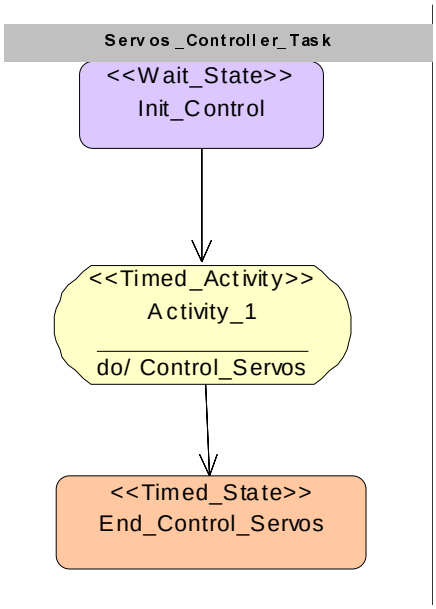
Contiene el único escenario de tiempo real que está definido en el sistema. Corresponde al control interactivo del robot desde la estación de teleoperación:

- Los comandos que el operador introduce por la GUI se traducen y hacen llegar hasta el robot.
- Se monitoriza el estado del robot y de los sensores y se representan en el display de la GUI.
- Mediante un bucle de control periódico se mantienen los servos en sus consignas.


<<Periodic_Event_Source>>
Init_Control
Period = 5.0E-3
<<Regular_Transaction>>
Control_Servos_Trans
<<Hard_Global_Deadline>>
End_Control_Servos
Deadline = 5.0E-3
<<Periodic_Event_Source>>
Init_Report
Period = 100.0E-3
<<Regular_Transaction>>
Report_Process
<<Hard_Global_Deadline>>
Display_Refreshed
Deadline = 100.0E-3
<<Sporadic_Event_Source>>
Tick_Command
Min_Interarrival = 1.0
<<Regular_Transaction>>
Execute_Command
<<Hard_Global_Deadline>>
Command_Processed
Deadline = 1.0

Control_Servos_Trans

Modela la transacción de tiempo real de control de los servos. Con periodo de 5 ms el controller ejecuta la operación Control_Servos. Se requiere que esta operación finalice antes de que se deba ejecutar de nuevo.



En su diagrama de actividad se describe la secuencia de actividades que debe ejecutar:

La actividad Control_Servos es del tipo "Timed_Activity" ya que es iniciada mediante la interrupción hardware del Timer. Esto representa que con independencia de la prioridad que se le asigne hay un segmento que se ejecuta con prioridad de interrupción hardware.

Init_Control

Dispara la transacción Control_Servos_Trans y establece que la transacción debe iniciarse periódicamente cada 5 ms.

Private Attributes:

Period : = 5.0E-3

End_Control_Servos

Establece que la transacción Control_Servos_Trans debe concluir con un Deadline de 5 ms relativo al evento Init_Control que la inicia.

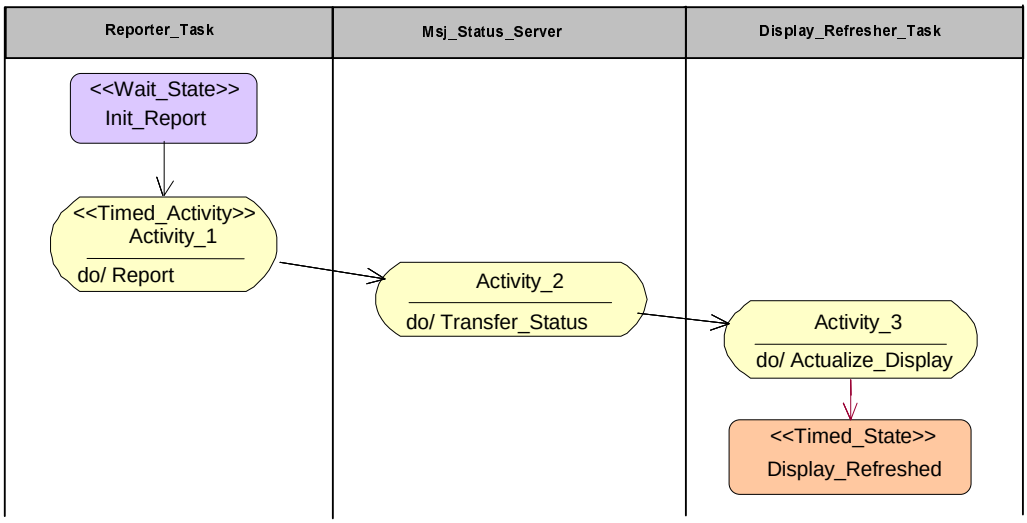
Private Attributes:

Deadline : = 5.0E-3

Report_Process

Modela la secuencia de actividades destinadas a actualizar el display de la GUI con el estado actual de los servos del robot y de los sensores. Se activa periódicamente cada 100 ms.

En su diagrama de actividad se describe la secuencia de actividades que ejecuta.



Init_Report

Describe el patrón de generación de eventos externos que inician la transacción Report_Process. Son eventos periódicos generados por el timer cada 100 ms.

Private Attributes:

Period : = 100.0E-3

Display_Refreshed

Establece que la transacción Report_Process debe finalizar con un deadline de 100 ms relativo a su inicio por el evento Init_Report.

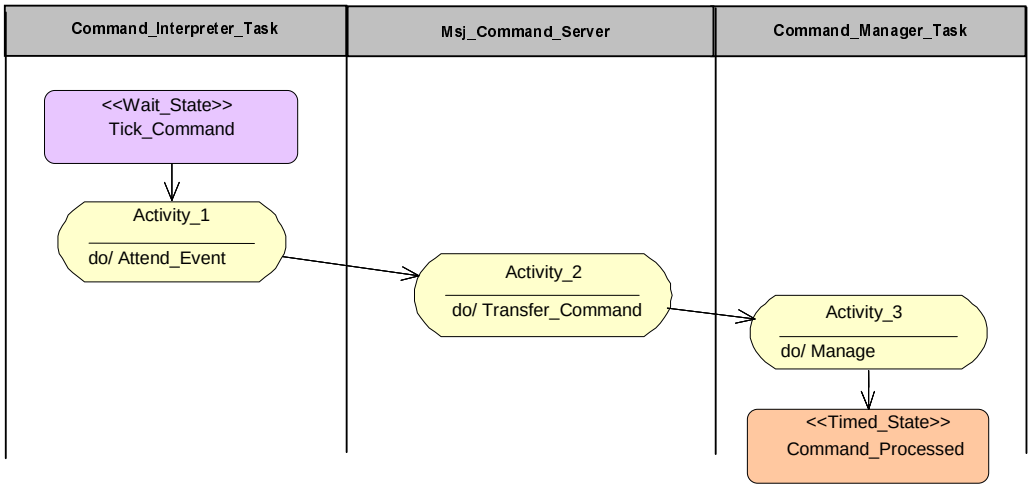
Private Attributes:

Deadline : = 100.0E-3

Execute_Command

Describe la secuencia de actividades que se genera cuando el operador introduce con el teclado o con el ratón un comando por la GUI. Es un evento esporádico, pero a fin de que el sistema sea planificable, se define como acotado (Bounded) a 1 sg, esto es, se limita a 1 sg el intervalo de tiempo que el operador debe respetar entre comando y comando. La GUI puede eventualmente forzar este límite.

En su diagrama de actividad se describe la secuencia de actividades de que consta.



Tick_Command

Se establece que el evento que representa que el operador ha introducido un comando a través de la GUI, debe generarse con un intervalo mínimo entre eventos de 1 segundo.

Private Attributes:

Min_Interarrival : = 1.0

Command_Processed

Establece que la transacción Execute_Command debe concluir con un deadline de 1 seg. relativo al evento Tick_Command que la inició.

Private Attributes:

Deadline : = 1.0

V ANALISIS DEL MODELO DE TIEMPO REAL

El análisis se realiza desde la propia herramienta ROSE. En la figura se muestran las opciones de análisis de tiempo real que se incorporan a la opción de menú Tools de la herramienta, tras haber sido incorporado el framework UML-Mast.

Check UML-MAST Componets

Permite verificar la definición de los componentes de la Mast RT View y la consistencia de sus asociaciones que constituyen el modelo de tiempo real.

Los resultados del chequeo se proporcionan en una ventana específica “Mast Real-Time View components check”.

Compile Model into Mast File

Compila la Mast RT View en un modelo Mast formulado mediante su correspondiente fichero textual. Por cada uno de los escenario del modelo se genera un modelo Mast independiente.

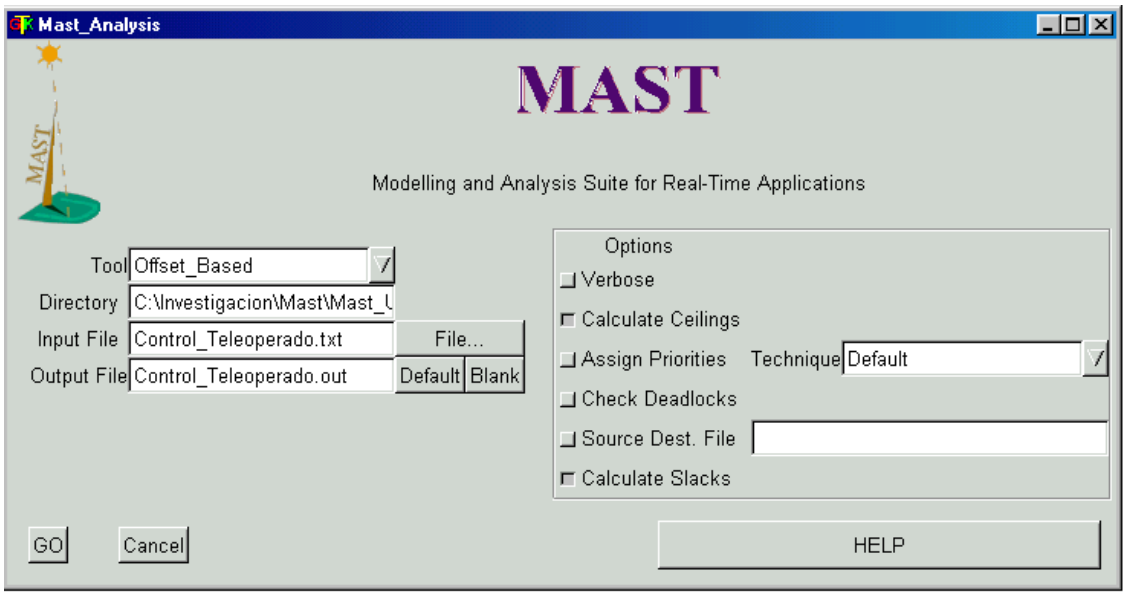
La compilación del modelo Teleoperated Robot genera en el direcctorio en el que está almacenado el modelo “Teoperated_Robot.mdl”, un nuevo directorio con el nombre “Teleoperated_Robot_Mast_Files”. En el nuevo directorio se almacena el fichero de texto Mast resultante de la compilación “Control_Teleoperado.txt”, que tiene como nombre el del escenario analizado (que en esta caso es el único del modelo).

El contenido de este fichero se muestra en el Apendice I.

Analysis Tools

Da acceso a la ventana de invocación de las herramientas del entorno Mast. Desde ella se elige el tipo de herramienta, los aspectos que se desean analizar y el directorio y el fichero en que se encuentra el modelo Mast que se analiza.

El resultado del análisis se almacena en el fichero Control_Teleoperated.out.



El análisis del modelo Teleoperated_Robot genera como resultado el fichero "..\Teleoperated_Robot_Mast_Files\Control_Teleoperado.out" que se incluye en el Apendice II.

Resultados del análisis:

Planificabilidad:

El sistema es planificable: Si el sistema no fuese planificable lo indicaría y el análisis finalizaría.

Requerimientos temporales:

1)	Transacción	=> Control_Servos_Trans
	Evento con requerimiento temporal	=> End_Control_Servos
	Tipo de requerimiento temporal	=> Global_Hard_Deadline
	Evento de referencia	=> Init_Control
	Deadline	=> 0.005
Resultado del análisis:		
	Worst_Global_Response_Times	=> 0.003191

Best_Global_Response_Times => 0.001646
Jitters => 0.001545
Requerimiento cumplido

2) Transacción => Report_Process
Evento con requerimiento temporal => Display_Refreshed
Tipo de requerimiento temporal => Global_Hard_Deadline
Evento de referencia => Init_Report
Deadline => 0.1
Resultado del análisis:
Worst_Global_Response_Times => 0.032053
Best_Global_Response_Times => 0.011097
Jitters => 0.020956
Requerimiento cumplido

3) Transacción => Execute_Command
Evento con requerimiento temporal => Command_Processed
Tipo de requerimiento temporal => Global_Hard_Deadline
Evento de referencia => Tick_Command
Deadline => 1.0
Resultado del análisis:
Worst_Global_Response_Times => 0.370071
Best_Global_Response_Times => 0.041397
Jitters => 0.328674
Requerimiento cumplido

Cálculo de holguras (slack)

La holgura se mide como el tanto por ciento en que se tiene que incrementar uniformemente los tiempos de las operaciones que intervienen en la transacción, para que deje de ser planificable

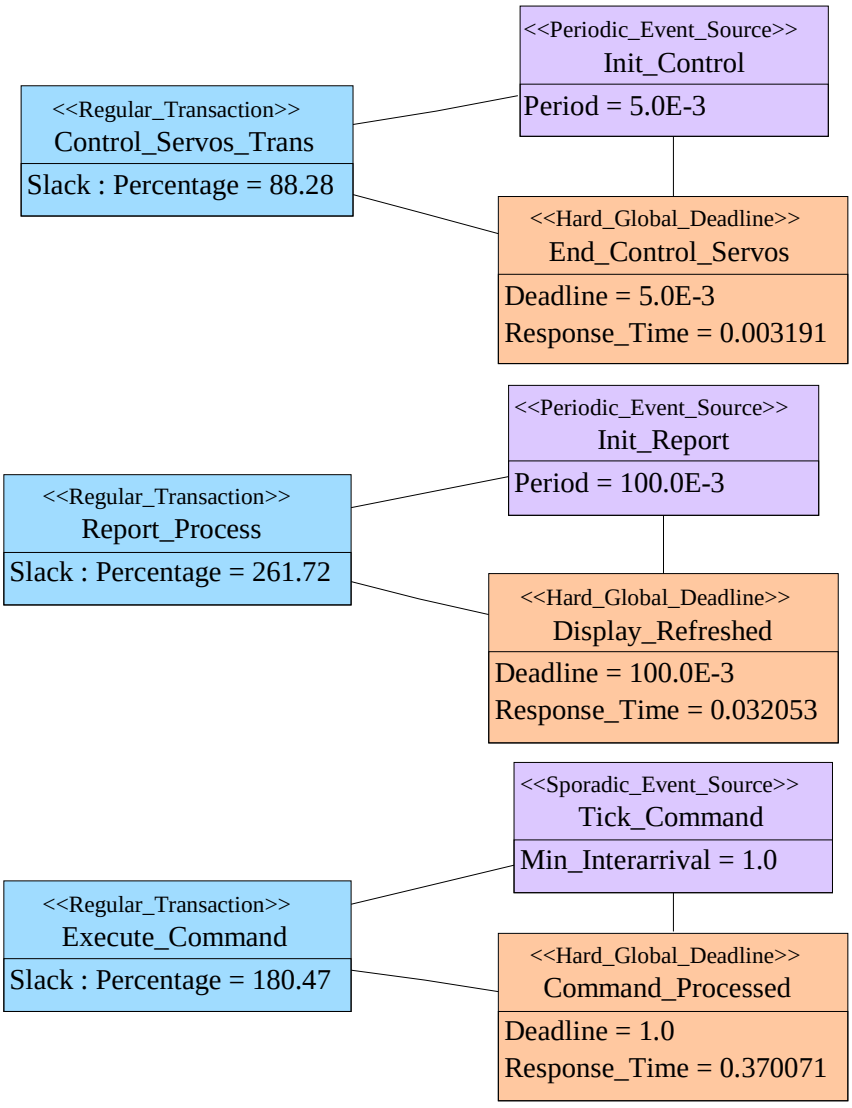
- Transacción => Control_Servos_Trans
Slack => 88.28 %

- Transacción => Report_Process
Slack => 261.72 %

- Transacción => Execute_Command
Slack => 180.47 %

Insert Analysis Results into Scenario Model

Pulsando esta opción del menú, se incorpora a la Mast_RT_View los resultados mas relevantes del análisis. En el diagrama se muestran estos resultados incorporados.



APENDICE I: MODELO MAST GENERADO POR EL COMPILADOR.

FILE: `..\Teleoperated_Robot_Mast_Files\Control_Teleoperado.txt`

```
Processing_Resource (
    Type                          => Fixed_Priority_Processor,
    Name                          => Station,
    Max_Priority                  => 31,
    Min_Priority                  => 16,
    Max_Interrupt_Priority        => 32,
    Min_Interrupt_Priority        => 32,
    Worst_Context_Switch          => 5.0E-6,
    Avg_Context_Switch            => 4.2E-6,
    Best_Context_Switch           => 4.0E-6,
    Worst_ISR_Switch              => 2.5E-6,
    Avg_ISR_Switch                => 2.5E-6,
    Best_ISR_Switch               => 2.5E-6,
    Speed_Factor                  => 1.0);
Processing_Resource (
    Type                          => Fixed_Priority_Processor,
    Name                          => Controller,
    Max_Priority                  => 30,
    Min_Priority                  => 1,
    Max_Interrupt_Priority        => 31,
    Min_Interrupt_Priority        => 31,
    Worst_Context_Switch          => 5.0E-6,
    Avg_Context_Switch            => 5.0E-6,
    Best_Context_Switch           => 5.0E-6,
    Worst_ISR_Switch              => 2.5E-6,
    Avg_ISR_Switch                => 2.5E-6,
    Best_ISR_Switch               => 2.5E-6,
    System_Timer                  => (
          Type           => Ticker,
          Worst_Overhead => 7.1E-6,
          Period         => 1.0E-3),
          Speed_Factor                  => 0.25);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Display_Refresher_Task,
    Server_Sched_Parameters       => (
         Type => Fixed_Priority_Policy,
         The_Priority           => 22),
    Server_Processing_Resource    => Station);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Command_Interpreter_Task,
    Server_Sched_Parameters       => (
         Type => Fixed_Priority_Policy,
         The_Priority           => 18),
    Server_Processing_Resource    => Station);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Station_Comm_Task,
```

```
    Server_Sched_Parameters       => (
         Type => Interrupt_FP_Policy,
         The_Priority           => 32),
    Server_Processing_Resource    => Station);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Reporter_Task,
    Server_Sched_Parameters       => (
         Type => Fixed_Priority_Policy,
         The_Priority               => 24),
    Server_Processing_Resource    => Controller);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Command_Manager_Task,
    Server_Sched_Parameters       => (
         Type => Fixed_Priority_Policy,
         The_Priority               => 16),
    Server_Processing_Resource    => Controller);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Servos_Controller_Task,
    Server_Sched_Parameters       => (
         Type => Fixed_Priority_Policy,
         The_Priority               => 30),
    Server_Processing_Resource    => Controller);
Scheduling_Server (
    Type                          => Fixed_Priority,
    Name                          => Controller_Comm_Task,
    Server_Sched_Parameters       => (
         Type => Interrupt_FP_Policy,
         The_Priority               => 31),
    Server_Processing_Resource    => Controller);
Operation       (Type                      => Simple,
          Name                         => Null_Operation,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0);
Processing_Resource (
    Type                          => Fixed_Priority_Network,
    Name                          => Can_Bus,
    Max_Priority                  => 2047,
    Min_Priority                  => 16,
    Packet_Worst_Overhead         => 58.75E-6,
    Packet_Avg_Overhead           => 58.75E-6,
    Packet_Best_Overhead          => 58.75E-6,
    Transmission                  => Half_Duplex,
    Max_Blocking                  => 138.75E-6,
    Max_Packet_Transmission_Time  => 8.0E-5,
    Min_Packet_Transmission_Time  => 8.0E-5,
    Speed_Factor                  => 0.8,
    List_of_Drivers               => (
          (Type                      => Packet_Driver,
          Packet_Server              =>
               (Type => Fixed_Priority,
                Name => Station_Comm_Task,
                Server_Sched_Parameters => (
                   Type => Interrupt_FP_Policy,
```

```
                The_Priority              => 32),
              Server_Processing_Resource => Station),
          Packet_Send_Operation      => (
              Type => Simple,
              Name                       => Send_Oper,
              Worst_Case_Execution_Time => 2.5E-6,
              Avg_Case_Execution_Time   => 2.5E-6,
              Best_Case_Execution_Time  => 2.5E-6),
          Packet_Receive_Operation   => (
              Type                       => Simple,
              Name                       => Receive_Oper,
              Worst_Case_Execution_Time => 2.0E-6,
              Avg_Case_Execution_Time   => 2.0E-6,
              Best_Case_Execution_Time  => 2.0E-6)),
         (Type                        => Packet_Driver,
          Packet_Server               =>
               ( Type => Fixed_Priority,
                 Name => Controller_Comm_Task,
                 Server_Sched_Parameters => (
                   Type              => Interrupt_FP_Policy,
                   The_Priority      => 31),
                 Server_Processing_Resource => Controller),
          Packet_Send_Operation      => (
              Type                       => Simple,
              Name                       => Send_Oper,
              Worst_Case_Execution_Time => 2.5E-6,
              Avg_Case_Execution_Time   => 2.5E-6,
              Best_Case_Execution_Time  => 2.5E-6),
           Packet_Receive_Operation   => (
              Type                       => Simple,
              Name                       => Receive_Oper,
              Worst_Case_Execution_Time => 2.0E-6,
              Avg_Case_Execution_Time   => 2.0E-6,
              Best_Case_Execution_Time  => 2.0E-6))));
Scheduling_Server (
    Type                           => Fixed_Priority,
    Name                           => Msj_Command_Server,
    Server_Sched_Parameters        => (
          Type                       => Fixed_Priority_Policy,
          The_Priority               => 20),
    Server_Processing_Resource     => Can_Bus);
Scheduling_Server (
    Type                           => Fixed_Priority,
    Name                           => Msj_Status_Server,
    Server_Sched_Parameters        => (
          Type                       => Fixed_Priority_Policy,
          The_Priority               => 24),
    Server_Processing_Resource     => Can_Bus);
Shared_Resource (
    Type                           => Immediate_Ceiling_Resource,
    Name                           => Display_Data_Lock);
Shared_Resource (
    Type                           => Immediate_Ceiling_Resource,
    Name                           => Servos_Data_Lock);
Operation (
           Type                      => Simple,
           Name                      => Display_Data.Read,
```

```
          Worst_Case_Execution_Time => 2.0E-3,
          Avg_Case_Execution_Time   => 1.8E-3,
          Best_Case_Execution_Time  => 0.12E-3);
Operation (
          Type                      => Simple,
          Name                      => Display_Data.Write,
          Worst_Case_Execution_Time => 2.5E-3,
          Avg_Case_Execution_Time   => 2.4E-3,
          Best_Case_Execution_Time  => 0.15E-3);
Operation (
          Type                      => Simple,
          Name                      => Servos_Data.Get,
          Worst_Case_Execution_Time => 2.0E-5,
          Avg_Case_Execution_Time   => 4.2E-6,
          Best_Case_Execution_Time  => 4.0E-6);
Operation (
          Type                      => Simple,
          Name                      => Servos_Data.Put,
          Worst_Case_Execution_Time => 2.5E-5,
          Avg_Case_Execution_Time   => 5.2E-6,
          Best_Case_Execution_Time  => 5.0E-6);
Operation (
          Type                      => Simple,
          Name                      => Actualize_Graphic,
          Worst_Case_Execution_Time => 10.0E-3,
          Avg_Case_Execution_Time   => 6.5E-3,
          Best_Case_Execution_Time  => 5.0E-3);
Operation (
          Type                      => Simple,
          Name                      => Plan_Trajectory,
          Worst_Case_Execution_Time => 200.0E-3,
          Avg_Case_Execution_Time   => 80.0E-3,
          Best_Case_Execution_Time  => 20.0E-3);
Operation (
          Type                      => Simple,
          Name                      => Transfer_Command,
          Worst_Case_Execution_Time => 32.0E-5,
          Avg_Case_Execution_Time   => 32.0E-5,
          Best_Case_Execution_Time  => 32.0E-5);
Operation (
          Type                      => Simple,
          Name                      => Transfer_Status,
          Worst_Case_Execution_Time => 64.0E-5,
          Avg_Case_Execution_Time   => 64.0E-5,
          Best_Case_Execution_Time  => 64.0E-5);
Operation (
          Type                      => Simple,
          Name                      =>
Station_Communication.Send_Command,
          Worst_Case_Execution_Time => 1.5E-5,
          Avg_Case_Execution_Time   => 1.5E-5,
          Best_Case_Execution_Time  => 1.5E-5);
Operation (
          Type                      => Simple,
          Name                      =>
Station_Communication.Wait_Status,
          Worst_Case_Execution_Time => 2.2E-5,
```

```
          Avg_Case_Execution_Time   => 2.2E-5,
          Best_Case_Execution_Time  => 2.2E-5);
Operation (
          Type                      => Simple,
          Name                      =>
Controller_Communication.Send_Status,
          Worst_Case_Execution_Time => 3.1E-5,
          Avg_Case_Execution_Time   => 3.1E-5,
          Best_Case_Execution_Time  => 3.1E-5);
Operation (
          Type                      => Simple,
          Name                      =>
Controller_Communication.Wait_Command,
          Worst_Case_Execution_Time => 0.20E-5,
          Avg_Case_Execution_Time   => 0.2E-5,
          Best_Case_Execution_Time  => 0.2E-5);
Operation (
          Type                      => Simple,
          Name                      => Control_Algorithm,
          Worst_Case_Execution_Time => 2.5E-4,
          Avg_Case_Execution_Time   => 2.5E-4,
          Best_Case_Execution_Time  => 2.5E-4);
Operation (
          Type                      => Simple,
          Name                      => Do_Control,
          Worst_Case_Execution_Time => 1.5E-4,
          Avg_Case_Execution_Time   => 1.5E-4,
          Best_Case_Execution_Time  => 1.5E-4);
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000003,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Display_Data_Lock));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000004,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Unlock=> (Display_Data_Lock));
Operation (
          Type                      => Composite,
          Name                      => Protected_Oper000002,
          Composite_Operation_List  => (Protected_Oper000003,
                                        Display_Data.Read,
                                        Protected_Oper000004));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000006,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Display_Data_Lock));
Operation (
          Type                      => Simple,
```

```
          Name                       => Protected_Oper000007,
          Worst_Case_Execution_Time  => 0.0,
          Avg_Case_Execution_Time    => 0.0,
          Best_Case_Execution_Time   => 0.0,
          Shared_Resources_To_Unlock=> (Display_Data_Lock));
Operation (
          Type                       => Composite,
          Name                       => Protected_Oper000005,
          Composite_Operation_List   => (Protected_Oper000006,
                                         Display_Data.Write,
                                         Protected_Oper000007));
Operation (
          Type                       => Composite,
          Name                       => Actualize_Display,
          Composite_Operation_List   =>
(Station_Communication.Wait_Status,
                                         Protected_Oper000002,
                                         Protected_Oper000005,
                                         Actualize_Graphic));
Operation (
          Type                       => Simple,
          Name                       => Protected_Oper000009,
          Worst_Case_Execution_Time  => 0.0,
          Avg_Case_Execution_Time    => 0.0,
          Best_Case_Execution_Time   => 0.0,
          Shared_Resources_To_Lock   => (Display_Data_Lock));
Operation (
          Type                       => Simple,
          Name                       => Protected_Oper000010,
          Worst_Case_Execution_Time  => 0.0,
          Avg_Case_Execution_Time    => 0.0,
          Best_Case_Execution_Time   => 0.0,
          Shared_Resources_To_Unlock=> (Display_Data_Lock));
Operation (
          Type                       => Composite,
          Name                       => Protected_Oper000008,
          Composite_Operation_List   => (Protected_Oper000009,
                                         Display_Data.Read,
                                         Protected_Oper000010));
Operation (
          Type                       => Simple,
          Name                       => Protected_Oper000012,
          Worst_Case_Execution_Time  => 0.0,
          Avg_Case_Execution_Time    => 0.0,
          Best_Case_Execution_Time   => 0.0,
          Shared_Resources_To_Lock   => (Display_Data_Lock));
Operation (
          Type                       => Simple,
          Name                       => Protected_Oper000013,
          Worst_Case_Execution_Time  => 0.0,
          Avg_Case_Execution_Time    => 0.0,
          Best_Case_Execution_Time   => 0.0,
          Shared_Resources_To_Unlock=> (Display_Data_Lock));
Operation (
          Type                       => Composite,
          Name                       => Protected_Oper000011,
          Composite_Operation_List   => (Protected_Oper000012,
```

```
                                        Display_Data.Write,
                                        Protected_Oper000013));
Operation (
          Type                      => Composite,
          Name                      => Attend_Event,
          Composite_Operation_List  => (Protected_Oper000008,
                                        Plan_Trajectory,
                                        Protected_Oper000011,

Station_Communication.Send_Command));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000015,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Servos_Data_Lock));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000016,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Unlock=> (Servos_Data_Lock));
Operation (
          Type                      => Composite,
          Name                      => Protected_Oper000014,
          Composite_Operation_List  => (Protected_Oper000015,
                                        Servos_Data.Get,
                                        Protected_Oper000016));
Operation (
          Type                      => Enclosing,
          Name                      => Report,
          Worst_Case_Execution_Time => 1.22E-3,
          Avg_Case_Execution_Time   => 1.15E-3,
          Best_Case_Execution_Time  => 1.1E-3,
          Composite_Operation_List  => (Protected_Oper000014));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000018,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Servos_Data_Lock));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000019,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Unlock=> (Servos_Data_Lock));
Operation (
          Type                      => Composite,
          Name                      => Protected_Oper000017,
          Composite_Operation_List  => (Protected_Oper000018,
                                        Servos_Data.Put,
                                        Protected_Oper000019));
```

```
Operation (
          Type                      => Enclosing,
          Name                      => Manage,
          Worst_Case_Execution_Time => 11.5E-3,
          Avg_Case_Execution_Time   => 5.2E-3,
          Best_Case_Execution_Time  => 5.1E-3,
          Composite_Operation_List  => (Protected_Oper000017));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000021,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Servos_Data_Lock));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000022,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Unlock=> (Servos_Data_Lock));
Operation (
          Type                      => Composite,
          Name                      => Protected_Oper000020,
          Composite_Operation_List  => (Protected_Oper000021,
                                        Servos_Data.Get,
                                        Protected_Oper000022));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000024,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Lock  => (Servos_Data_Lock));
Operation (
          Type                      => Simple,
          Name                      => Protected_Oper000025,
          Worst_Case_Execution_Time => 0.0,
          Avg_Case_Execution_Time   => 0.0,
          Best_Case_Execution_Time  => 0.0,
          Shared_Resources_To_Unlock=> (Servos_Data_Lock));
Operation (
          Type                      => Composite,
          Name                      => Protected_Oper000023,
          Composite_Operation_List  => (Protected_Oper000024,
                                        Servos_Data.Put,
                                        Protected_Oper000025));
Operation (
          Type                      => Composite,
          Name                      => Control_Servos,
          Composite_Operation_List  => (Protected_Oper000020,
                                        Control_Algorithm,
                                        Do_Control,
                                        Protected_Oper000023));

--Inicio del escenario propiamente dicho :
```

```
Transaction (
 Type            => Regular,
 Name            => Control_Servos_Trans,
 External_Events => (
                     ( Type       => Periodic,
                       Name       => Init_Control,
                       Period     => 5.0E-3
                     )
                    ),
 Internal_Events => (
                     ( Type                 => Regular,
                       Name                 => End_Control_Servos,
                       Timing_Requirements  =>
                       ( Type     => Hard_Global_Deadline,
                         Deadline => 5.0E-3,
                         Referenced_Event => Init_Control)
                     )
                    ),
 Event_Handlers  => (
                     ( Type               => System_Timed_Activity,
                       Input_Event        => Init_Control,
                       Output_Event       => End_Control_Servos,
                       Activity_Operation => Control_Servos,
                       Activity_Server    => Servos_Controller_Task
                     )
                    )
);

-- End of Transaction. --

Transaction (
 Type            => Regular,
 Name            => Report_Process,
 External_Events => (
                     ( Type       => Periodic,
                       Name       => Init_Report,
                       Period     => 100.0E-3
                     )
                    ),
 Internal_Events => (
                     ( Type                 => Regular,
                       Name                 => Display_Refreshed,
                       Timing_Requirements  =>
                       ( Type     => Hard_Global_Deadline,
                         Deadline => 100.0E-3,
                         Referenced_Event => Init_Report)
                     ),
                     ( Type                 => Regular,
                       Name                 => T000026
                     ),
                     ( Type                 => Regular,
                       Name                 => T000027
                     )
                    ),
 Event_Handlers  => (
                     ( Type               => System_Timed_Activity,
                       Input_Event        => Init_Report,
```

```
                      Output_Event       => T000026,
                      Activity_Operation => Report,
                      Activity_Server    => Reporter_Task
                     ),
                     ( Type               => Activity,
                      Input_Event        => T000026,
                      Output_Event       => T000027,
                      Activity_Operation => Transfer_Status,
                      Activity_Server    => Msj_Status_Server
                     ),
                     ( Type               => Activity,
                      Input_Event        => T000027,
                      Output_Event       => Display_Refreshed,
                      Activity_Operation => Actualize_Display,
                      Activity_Server    => Display_Refresher_Task
                     )
                    )
);

-- End of Transaction. --

Transaction (
 Type            => Regular,
 Name            => Execute_Command,
 External_Events => (
                     ( Type       => Sporadic,
                      Name       => Tick_Command,
                      Min_Interarrival => 1.0
                     )
                    ),
 Internal_Events => (
                     ( Type                 => Regular,
                      Name                 => Command_Processed,
                      Timing_Requirements  =>
                      ( Type     => Hard_Global_Deadline,
                        Deadline => 1.0,
                        Referenced_Event => Tick_Command)
                     ),
                     ( Type                 => Regular,
                      Name                 => T000028
                     ),
                     ( Type                 => Regular,
                      Name                 => T000029
                     )
                    ),
 Event_Handlers  => (
                     ( Type               => Activity,
                      Input_Event        => Tick_Command,
                      Output_Event       => T000028,
                      Activity_Operation => Attend_Event,
                      Activity_Server    => Command_Interpreter_Task
                     ),
                     ( Type               => Activity,
                      Input_Event        => T000028,
                      Output_Event       => T000029,
                      Activity_Operation => Transfer_Command,
                      Activity_Server    => Msj_Command_Server
```

```
                      ),
                      ( Type               => Activity,
                        Input_Event        => T000029,
                        Output_Event       => Command_Processed,
                        Activity_Operation => Manage,
                        Activity_Server    => Command_Manager_Task
                      )
                     )
);

-- End of Transaction. --
=====================================================================
```

APENDICE II: FICHERO RESULTANTE DEL ANÁLISIS DE TIEMPO REAL.

FILE: `..\Teleoperated_Robot_Mast_Files\Control_Teleoperado.out`

```
MAST Version: 1.00

Parsing input file
Calculating Ceilings
Invoking the analysis tool...
Transaction (
   Name     => control_servos_trans,
   Results  =>
      ( (Type                          => Timing_Result,
         Event_Name                    => end_control_servos,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => init_control,
              Time_Value       => 0.003191)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => init_control,
              Time_Value       => 0.001646)),
         Jitters                       =>
            ((Referenced_Event => init_control,
              Time_Value       => 0.001545)))));

Transaction (
   Name     => report_process,
   Results  =>
      ( (Type                          => Timing_Result,
         Event_Name                    => display_refreshed,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => init_report,
              Time_Value       => 0.032053)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => init_report,
              Time_Value       => 0.011097)),
         Jitters                       =>
            ((Referenced_Event => init_report,
              Time_Value       => 0.020956))),
        (Type                          => Timing_Result,
         Event_Name                    => t000026,
         Num_Of_Suspensions            => 0,
         Blocking_Time                 => 1.388E-04,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => init_report,
              Time_Value       => 0.012945)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => init_report,
              Time_Value       => 0.004410)),
         Jitters                       =>
            ((Referenced_Event => init_report,
```

```
             Time_Value       => 0.008535))),
          (Type                          => Timing_Result,
           Event_Name                    => t000027,
           Num_Of_Suspensions            => 0,
           Blocking_Time                 => 0.002500,
           Worst_Global_Response_Times   =>
              ((Referenced_Event => init_report,
                Time_Value       => 0.014471)),
           Best_Global_Response_Times    =>
              ((Referenced_Event => init_report,
                Time_Value       => 0.005797)),
           Jitters                       =>
              ((Referenced_Event => init_report,
                Time_Value       => 0.008674)))));

Transaction (
   Name     => execute_command,
   Results  =>
      ( (Type                          => Timing_Result,
         Event_Name                    => command_processed,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.370071)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.041397)),
         Jitters                       =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.328674))),
        (Type                          => Timing_Result,
         Event_Name                    => t000028,
         Num_Of_Suspensions            => 0,
         Blocking_Time                 => 1.388E-04,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.256126)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.020293)),
         Jitters                       =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.235833))),
        (Type                          => Timing_Result,
         Event_Name                    => t000029,
         Num_Of_Suspensions            => 0,
         Blocking_Time                 => 0.00,
         Worst_Global_Response_Times   =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.258346)),
         Best_Global_Response_Times    =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.020987)),
         Jitters                       =>
            ((Referenced_Event => tick_command,
              Time_Value       => 0.237359)))));

Calculating Slacks...
```

```
Calculating Transaction Slack for control_servos_trans
Calculating Transaction Slack for report_process
Calculating Transaction Slack for execute_command
Transaction-Specific Slacks =>
  (control_servos_trans => 88.28 %,
   report_process => 261.72 %,
   execute_command => 180.47 %)
System slack : 71.09%
```