Sistemas Distribuidos de Tiempo Real

Apuntes: TEMA 7

Por: J. Javier Gutiérrez gutierjj@unican.es
http://www.ctr.unican.es/

Grupo de Computadores y Tiempo Real, Universidad de Cantabria

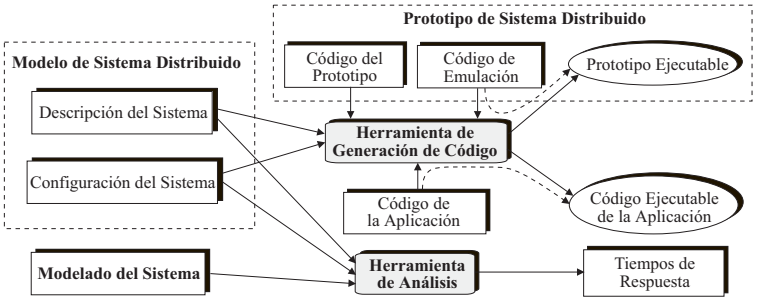
Sistemas distribuidos de tiempo real

PARTE III: Middlewares de distribución

- TEMA 5. Middleware de distribución para el modelo Ada
- TEMA 6. Middleware de distribución esquizofrénico para lenguaje Ada: CORBA y DSA
- TEMA 7. Middlewares de distribución de tiempo real

Introducción

Esquema de desarrollo de sistemas distribuidos



- integrar diseño, implementación y análisis

Introducción (cont.)

Se necesita una *plataforma distribuida* de tiempo real:

- ***Sistema operativo*** **de tiempo real: MaRTE OS**
- ***Redes*** **de tiempo real: RT-EP o bus CAN**
- ***Middleware*** **de tiempo real:**
  - **RT-GLADE: modificación de GLADE con optimización de las características de tiempo real**
  - **PolyORB: mejorado en el tratamiento de las llamadas remotas**

Aplicaciones que se pueden abordar principalmente:

- **sistemas empotrados con un pequeño número de procesadores y redes (heterogéneos)**

Introducción (cont.)

Herramientas de modelado y análisis de planificabilidad:

- **MAST (Modeling and Analysis Suite for Real Time Applications):**
  - **modelo del sistema, técnicas de análisis (prioridades fijas y EDF) y asignación de prioridades**
- **Modelo de tiempo real para UML analizable por MAST**
- **Emulador que opera sobre el modelo de MAST (tiempos promedio y contraste de resultados del análisis)**

Planificación flexible:

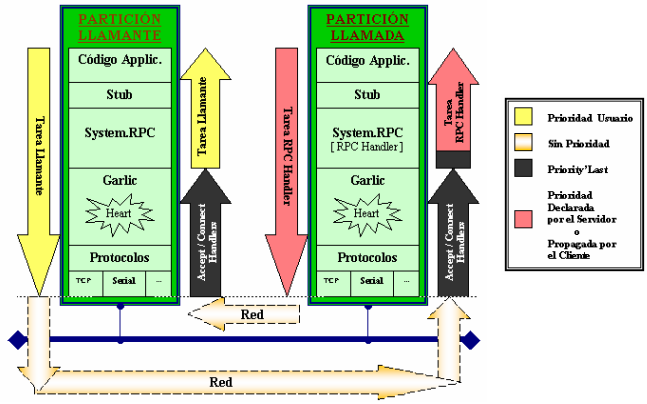
- **QoS y tiempo real tratados simultáneamente en un sistema basado en contratos y con planificación jerárquica**

Recordatorio de GLADE

Detalles de la implementación desde el punto de vista del tiempo real:

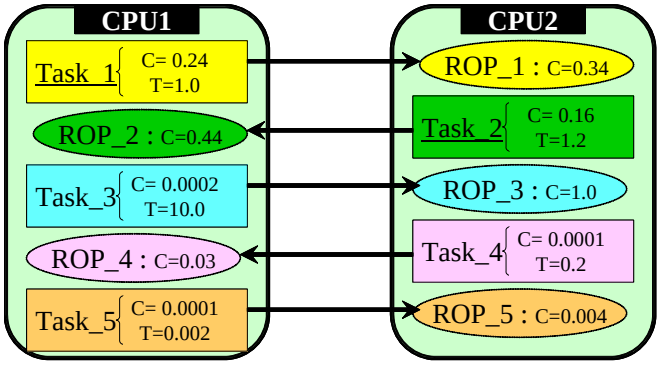
- **Linux sobre TCP/IP**
- ***pool*** **de tareas configurable por cada partición para la ejecución de peticiones remotas**
- **creación dinámica de tareas conectoras de las peticiones en la partición llamada**
  - ***Acceptor Task --> Connector Task --> Tarea del pool***
- **políticas de prioridades para las tareas del pool:**
  - ***Client Propagated*** **(ejecución remota a la prioridad del cliente)**
  - ***Server Declared*** **(todas las tareas del pool a la misma prioridad)**
- **elección arbitraria de una red de entre las disponibles**

Recordatorio del esquema de RPC en GLADE



Importancia de las asignaciones de prioridades en tiempo real

Ejemplo con Task_1 y Task_2 con requisitos temporales D=T



Importancia de las asignaciones de prioridades en tiempo real (cont.)

Evaluación en GLADE/Linux:

Priority Policy	Task_1 Prio.	ROP_1 Prio.	Task_2 Prio.	ROP_2 Prio.	Task_1 WCRT (s) D = 1.0	Task_2 WCRT (s) D = 1.2
Client Propagated	High	High	Medium	Medium	1.389	1.496
	Medium	Medium	High	High	1.293	1.416
Server Declared	High	Media	High	Medium	1.526	1.336
	High	High	Medium	Medium	1.523	1.337
	Medium	Medium	High	High	1.440	1.394

Importancia de las asignaciones de prioridades en tiempo real (cont.)

Evaluación en RT-GLADE/MaRTE OS

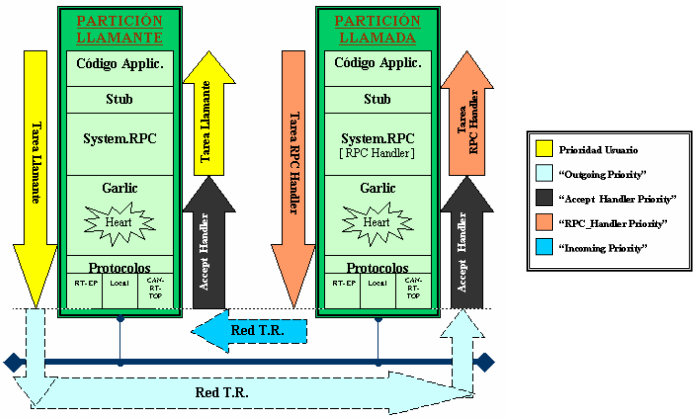
Priority Policy	Task_1 Prio.	ROP_1 Prio.	Task_2 Prio.	ROP_2 Prio.	Task_1 WCRT (s) D = 1.0	Task_2 WCRT (s) D = 1.2
MAST (network no priorit.)	High	Medium	High	Medium	0.806	0.898
Client Propagated	High	High	Medium	Medium	0.638	1.527
	Medium	Medium	High	High	No Acot.	0.653
Server Declared	High	Medium	High	Medium	1.538	1.149
	High	High	Medium	Medium	2.148	2.887
	Medium	Medium	High	High	No Acot.	0.655

Introducción a RT-GLADE

Modificación de GLADE que sigue conforme al estándar Ada:

- priorización de las tareas manejadoras de RPCs y de los mensajes enviados por la red
- incorporación de las redes RT con particionamiento del mensaje en paquetes
- eliminación de la incertidumbre en la selección de una red de entre las varias posibles
- eliminación de la creación dinámica de tareas en la recepción de una RPC
- adaptación de la configuración del sistema incluyendo aspectos de las redes de tiempo real

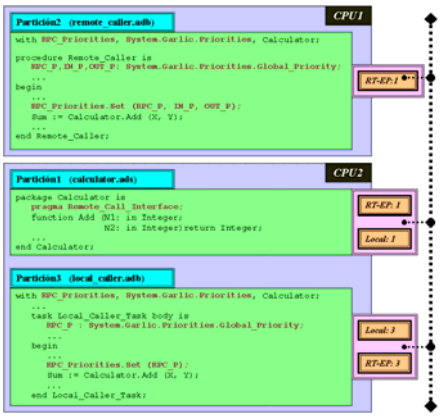
Esquema de RPC en RT-GLADE



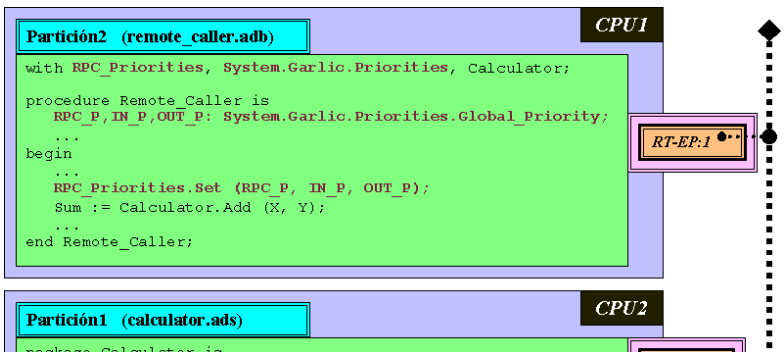
Asignación de prioridades en RT-GLADE

```
with System.Garlic.Priorities;
use System.Garlic.Priorities;

package RPC_Priorities is
   procedure Set (RPC_Handler : in Global_Priority);
   procedure Set (RPC_Handler,
                  Outgoing_Message : in Global_Priority);
   procedure Set (RPC_Handler,
                  Outgoing_Message,
                  Incoming_Message : in Global_Priority);
   procedure Get (RPC_Handler,
                  Outgoing_Message,
                  Incoming_Message : out Global_Priority);
end RPC_Priorities;
```

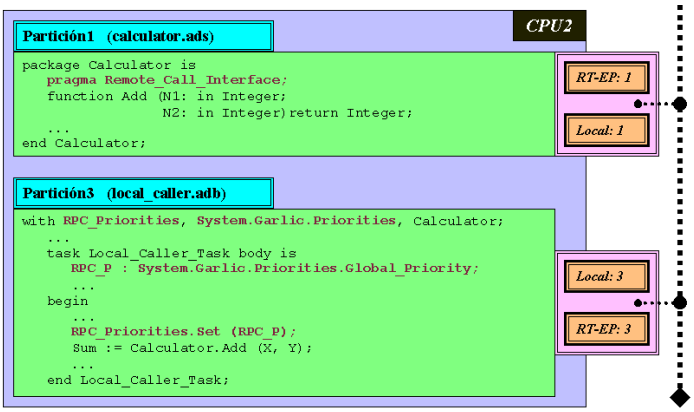

Ejemplo de uso de la asignación de prioridades



Ejemplo de uso de la asignación de prioridades (cont.)



Ejemplo de uso de la asignación de prioridades (cont.)



Configuración en RT-GLADE

La carga de las particiones se realiza manualmente

Con el atributo `Self_Location` se configuran los enlaces entre particiones (*Partition Links*):

- alternativa a los canales que establece una conexión unidireccional entre particiones y sirve para fijar la red que se utiliza y no los filtros (que no se usan)

El atributo `Task_Pool` se usa siempre con una configuración estática (los tres valores iguales) para evitar la creación y destrucción dinámica de tareas

Configuración en RT-GLADE (cont.)

El atributo `Priority` establece la prioridad inicial del *Accept Handler*:

- esto permite asignarle una prioridad inferior a la de las tareas de comunicaciones y superior a cualquiera de la aplicación
- en GLADE este atributo se añadió para fijar la prioridad de todas las tareas manejadoras de RPC dentro de una misma partición

El nuevo atributo `Max_RPC_Replies` establece el número máximo de llamadas RPCs simultáneas desde una partición:

- fija el número de *entries* del objeto protegido en el que las tareas que han realizado una RPC esperan su respuesta

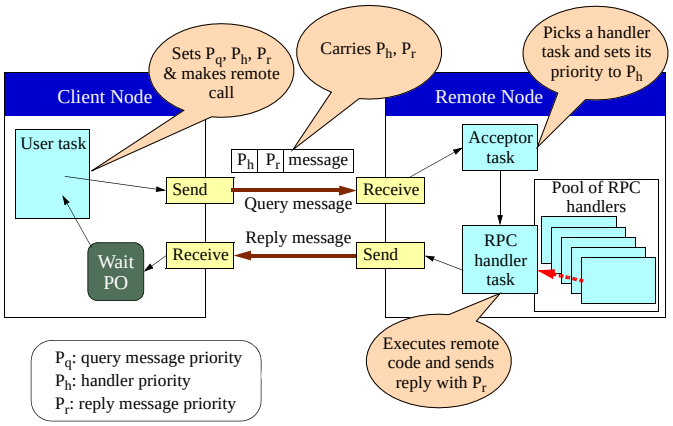
Ejemplo de configuración de RT-GLADE

```
configuration Configuration_File is

   pragma Starter (None);

   Partition1: partition := (Calculator);
   Partition2: partition  := (Remote_Caller);
   Partition3: partition;

   procedure Remote_Caller is in Partition2;

   procedure Local_Caller;
   for Partition3'Main use Local_Caller;

   pragma Boot_Location (("RT_EP", "CPU1:1"));
```


Ejemplo de configuración de RT-GLADE (cont.)

```
   for Partition1'Self_Location use
        ((("rt_ep"," CPU2:1"),("Local", "CPU2:1:Partition3")));
   for Partition2'Self_Location use ("rt_ep", "CPU1:1");
   for Partition3'Self_Location use
        ((("rt_ep"," CPU2:3"),("Local", "CPU2:3:Partition1")));

   for Partition1'Priority use 27;

   for Partition1'Task_Pool use (8,8,8);

   for Partition1'Max_RPC_Replies use 4;

end Configuration_File;
```


Transacciones y prioridades en RT-GLADE



Inconvenientes de la implementación RT-GLADE

Sólo maneja prioridades

Sólo maneja prioridades en el primer nivel de llamada a RPC:

- **no maneja llamadas anidadas**

Puntos de ineficiencia (pensando en los contratos de los proyectos FIRST y FRESCOR):

- **transmisión de otros parámetros de planificación más voluminosos que las prioridades**
- **cambio dinámico de los parámetros de planificación de un *RPC Handler* muy costoso**

Planificación basada en contratos

Planificación flexible basada en contratos:

- **los contratos representan una reserva de los recursos del sistema**

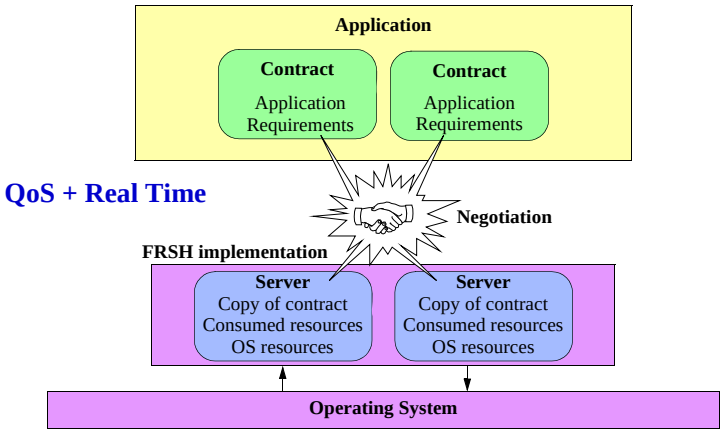
Los contratos se negocian de forma:

- **local: test de admisión de los recursos disponibles**
- **distribuida: *Data Transaction Manager***

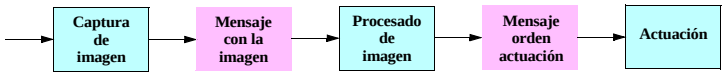
Las comunicaciones se realizan mediante paso de mensajes:

- **uso directo de la red (sin middleware)**

Planificación basada en contratos (cont.)



Ejemplo de llamadas anidadas en transacción distribuida




```
task Captura_Imagen is
   pragma Priority (10);
end Captura_Imagen;

task body Captura_Imagen is
begin
  ...
  Prioridad del mensaje y de la
  ejecución de Procesa_Imagen;
  Procesa_Imagen();
  ...
end Captura_Imagen;
```

```
procedure Procesa_Imagen() is
begin
  ...
  Prioridad del mensaje y de la
  ejecución de Actua;
  Actua ();
  ...
end Procesa_Imagen;

procedure Actua () is
begin
   ...
end Actua;
```


Ejemplo de llamadas anidadas en transacción distribuida (cont.)

```
task Captura_Imagen is
   pragma Priority (10);
end Captura_Imagen;
task body Captura_Imagen is
begin
  ...
  Prioridad del mensaje y de la
  ejecución de Procesa_Imagen;
  Procesa_Imagen();
  ...
end Captura_Imagen;
task Captura_Imagen_Bis is
   pragma Priority (15);
end Captura_Imagen_Bis;
task body Captura_Imagen_Bis is
begin
   Similar a Captura_Imagen
end Captura_Imagen_Bis;
```

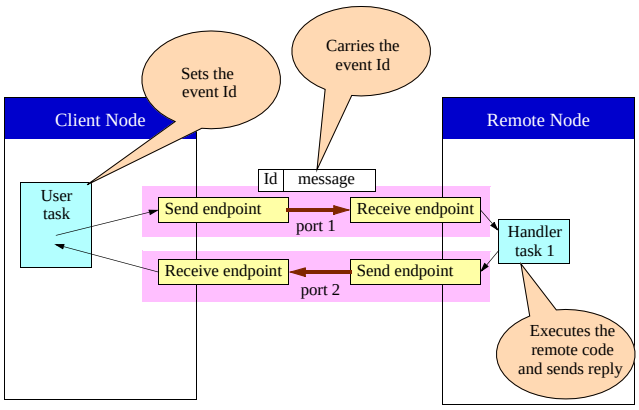
```
procedure Procesa_Imagen() is
begin
  ...
  ¿Establece prioridad del
   mensaje y de la ejecución de
   Actua?;
  Actua ();
  ...
end Procesa_Imagen;

procedure Actua () is
begin
   ...
end Actua;
```


Nuevo mecanismo de gestión de transacciones en RT-GLADE

Crear las entidades de planificación necesarias para ejecutar las RPCs al instanciar el componente:

- **Procesadores:**
  - ***RPC Handlers*** **en el lado del servidor**
  - **creación explícita de las tareas necesarias con sus parámetros de planificación**
- **Redes:**
  - **puertos de comunicaciones usados**
  - **creación de *endpoints* para envío y recepción de mensajes**
  - **parámetros de planificación asociados a los *endpoints* de envío**

Nuevo mecanismo de gestión de transacciones en RT-GLADE (cont.)



Nivel de comunicaciones de RT-GLADE

Cambios principales sobre la versión anterior:

- la tarea ***Acceptor*** ha sido eliminada:
  - la tarea RPC Handler espera directamente en la red
- el pool de tareas se ha eliminado
  - las tareas se crean estáticamente cuando se requiera
- el mecanismo de espera a la respuesta también se ha eliminado:
  - la tarea cliente espera directamente en la red

Es una arquitrectura más simple que soporta políticas mucho más complejas

Para añadir una nueva política de planificación

Hay que extender los tipos abstract tagged:

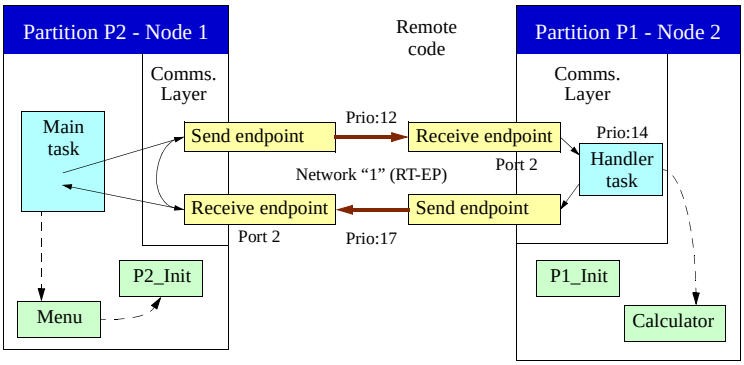
- `Task_Scheduling_Parameters`
  - parámetros de planificación de una tarea manejadora de RPC
- `Message_Scheduling_Parameters`
  - parámetros de planificación usados para los mensajes que se envían por la red

Hay que implementar las operaciones privadas:

- procedimiento `Create_Task`
- procedimiento `Create_Send_Endpoint`

RT-GLADE soporta prioridades fijas y contratos FIRST

Ejemplo de uso



Ejemplo de uso: Calculadora

```
package Calculadora is

   pragma Remote_Call_Interface;

   function Sumar (Operador1 : in Integer;
                   Operador2 : in Integer)
                   return Integer;

   function Restar (Operador1 : in Integer;
                    Operador2 : in Integer)
                    return Integer;

end Calculadora;
```


Ejemplo de uso: Menu

```
with Calculadora;
with Event_Id_Attributes_Set_Get;
with P2_Init;
```

Ejemplo de uso: Menu (cont.)

```
procedure Menu is
   Resultado_Suma : Integer;  Opp1, Opp2 : Integer := 0;
begin
   -- Initialize the communications endpoints
   P2_Init;
   -- Set the event id for subsequent RPCs
   Event_Id_Attributes_Set_Get.Set_Event_Id (12);
   loop
      begin
         ...
         Resultado_Suma := Calculadora.Sumar (Opp1, Opp2);
         ...
   end loop;
end Menu;
```


Ejemplo de uso: P2_Init

```
with System.Garlic.Priorities;
with Ada.Real_Time;
with Rt_Glade_Scheduling;
with Rt_Glade_Scheduling.Priorities;

Procedure P2_Init is

   R_Message_Params :
      Rt_Glade_Scheduling.Message_Scheduling_Parameters_Ref;
   RTEP_Message_Params : Rt_Glade_Scheduling.Priorities
      .Priorities_Message_Scheduling_Parameters;
```


Ejemplo de uso: P2_Init (cont.)

```
begin
   -- Initial Values to Priorities Params variables

   RTEP_Message_Params.Outgoing_Priority := 12;

   -- Assign the Priorities Params variables values
   -- to the Rt_Glade_Scheduling Ref variables

   Rt_Glade_Scheduling.To_Message_Params_Ref
     (RTEP_Message_Params, R_Message_Params);
```

Ejemplo de uso: P2_Init (cont.)

```
   -- Create_Send_Message_Handler
   Rt_Glade_Scheduling.Create_Send_Message_Handler
     (Params => R_Message_Params.all,
      Node   => 2,
      Net    => 1,
      Port   => 2,
      Event  => 12);

   -- Create the receive endpoint for the reply messages
   Rt_Glade_Scheduling.Create_Return_Message_Handler
     (Net   => 1,
      Port  => 2,
      Event => 12);
end P2_Init;
```


Ejemplo de uso: P1_Init

```
with Rt_Glade_Scheduling;
with Rt_Glade_Scheduling.Priorities;

procedure P1_Init is

   R_Task_Params :
      Rt_Glade_Scheduling.Task_Scheduling_Parameters_Ref;
   R_Message_Params :
      Rt_Glade_Scheduling.Message_Scheduling_Parameters_Ref;

   RTEP_Task_Params : Rt_Glade_Scheduling.Priorities
     .Priorities_Task_Scheduling_Parameters;
   RTEP_Message_Params : Rt_Glade_Scheduling.Priorities
     .Priorities_Message_Scheduling_Parameters;
```


Ejemplo de uso: P1_Init (cont.)

```
begin
   -- Initial Values to Priorities Params variables

   RTEP_Message_Params.Outgoing_Priority := 17;
   RTEP_Task_Params.RPC_Handler_Priority := 14;

   -- Assign the Priorities Params variables values
   -- to the Rt_Glade_Scheduling Ref variables

   Rt_Glade_Scheduling.To_Task_Params_Ref
     (RTEP_Task_Params, R_Task_Params);
   Rt_Glade_Scheduling.To_Message_Params_Ref
     (RTEP_Message_Params, R_Message_Params);
```

Ejemplo de uso: P1_Init (cont.)

```
   -- Create_Send_Message_Handler
   Rt_Glade_Scheduling.Create_Send_Message_Handler
     (Params => R_Message_Params.all,
      Node   => 1,
      Net    => 1,
      Port   => 2,
      Event  => 12);
   -- Create_RPC_Handler and its corresponding Receive Endpoint
   Rt_Glade_Scheduling.Create_RPC_Handler
     (Params => R_Task_Params.all,
      Net    => 1,
      Port   => 2,
      Event  => 12);
end P1_Init;
```


Ejemplo de uso: Configuración

```
configuration config is

   -- Manual launching
   pragma Starter (None);

   -- Partition 'p1' contains calculadora, and the main program
   -- of 'p1' is p1_init (to create the endpoints in p1)

   p1: partition := (calculadora);
   procedure p1_init;
   for p1'Main use p1_init;
```


Ejemplo de uso: Configuración

```
   -- Partition 'p2' is going to be the main partition
   -- in menu code: the application and the endpoints creation

   p2: partition;
   procedure menu is in p2;

   -- La particion p1 contactará para arrancar con la p2
   -- mediante RTEP en el canal 1 del PC cliente

   pragma Boot_Location ("Rt_Ep", "cliente:1:p1");
   for p1'Self_Location use (("Rt_Ep", "calcula:1:p2"));
   -- for p2'Self_Location use (("Rt_Ep", "cliente:1:p1"));

end config;
```

Ejemplo de uso: fichero de ctrpcs.txt

Este fichero contiene los nombres de los nodos y los protocolos que usa para comunicaciones:

```
calcula
>rt_ep = 00:02:44:3B:6A:C5
cliente
>rt_ep = 00:02:44:3B:6A:DE
```


Adaptación de PolyORB a RT

Se han añadido dos nuevas políticas de control:

- ReAdy To gO (RATO) para controlar las tareas del ORB: cada llamada remota es procesada por una sola tarea evitando cambios de contexto
- Thread Per Target (TPT) para controlar la creación y destrucción de tareas en el ORB
  - implementación de los endpoints
  - las tareas manejadoras de llamadas remotas se crean en la fase de configuración

Adaptación de PolyORB a RT (cont.)

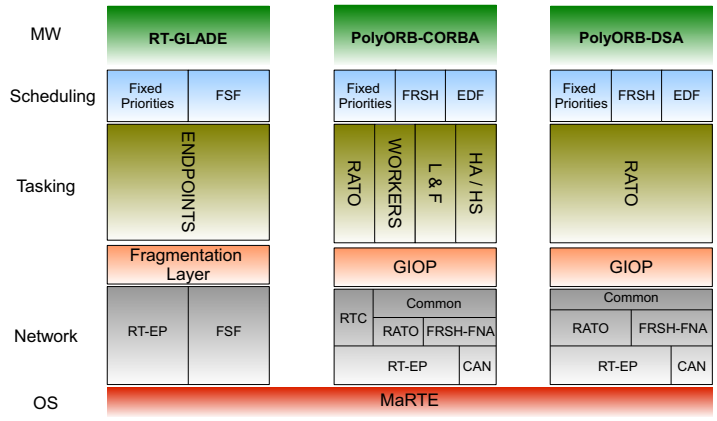
Se han añadido nuevas personalidades de protocolo:

- RATO sobre RT-EP
- FRSH-FNA sobre RT-EP y CAN

Ambas se conectan a GIOP con un nivel común y GIOP les proporciona el nivel de fragmentación de mensajes

Además para CORBA se dispone de la personalidad RTC en la que los parámetros de planificación se envían por la red

Arquitecturas de middleware de tiempo real



Evaluación de CORBA

10.000 invocaciones de una llamada remota simple con dos tipos de medidas:

- **1 cliente: evaluar la sobrecarga del middleware**
- **5 clientes: evaluar la efectividad del sistema de tiempo real**

Políticas de tarea y control usadas:

- **Single-thread para un cliente**
- **Multi-thread para 5 clientes**
  - **L&F: TAO and PolyORB-CORBA**
  - **Thread Pool: GLADE**
  - **Endpoints: RT-GLADE y adaptación a RT de PolyORB-DSA**

Planificación por prioridades fijas

Evaluación de CORBA en Linux

Linux, one client, time in µs

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
TAO	998	1380	914	75	0.06
PolyORB-CORBA	1424	4302	1189	373	0.01
GLADE	415	3081	340	261	0.02
Stand-alone network	129	678	118	40	0.12

Linux, five clients, time in µs

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
TAO	1371	6376	889	356	0.02
PolyORB-CORBA	5399	11554	1593	1050	0.02
GLADE	1700	5953	595	496	0.12

Evaluación de CORBA en MaRTE OS

MaRTE OS, one client, time in µs

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	2997	3012	2770	6	0.01
PolyORB-DSA	4117	4487	3835	300	42.50
RT-GLADE	1080	1151	955	23	0.03
Stand-alone network	959	964	707	3	0.01

MaRTE OS, five clients, time in µs

	Avg. Time	Max. Time	Min. Time	Std. Deviation	10% from Max. (%)
PolyORB-CORBA	3527	6566	2748	727	0.11
PolyORB-DSA	4516	5299	3531	320	0.02
RT-GLADE	1000	1462	896	27	0.06