Sistemas Distribuidos de Tiempo Real

Apuntes: TEMA 5

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 a GLADE

Implementación del Ada distribuido de Ada Core:

- **programa único dividido en particiones**
- **las particiones se distribuyen en los nudos procesadores**
- **distribución por RPCs y objetos: transparencia de uso**
- **configuración, arranque de particiones y otros aspectos definidos por la implementación**

Estructura de GLADE:

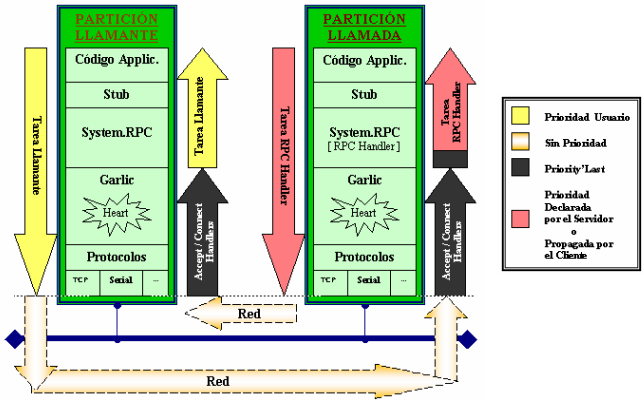
- **GARLIC: es el PCS (Partition Communication Subsystem)**
- **GNATDIST: herramienta de particionamiento, chequeo de consistencia, generación de stubs y enlace de las particiones con el PCS**

Introducción a GLADE (cont.)

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

Esquema de RPC en GLADE



El DSA en GLADE

Aspectos destacados de la implementación:

- Variables y access no remotos
- Fallos de RPCs
- Excepciones
- RPCs asíncronas
- Unidades genéricas
- Llamadas remotas concurrentes
- Aborto y terminación
- Configuración

Variables y access no remotos

En las declaraciones de paquetes Remote Types y Remote Call Interface:

- las declaraciones de variables están prohibidas
- los tipos access no remotos se permiten siempre que se proporcionen los subprogramas de marshalling y unmarshalling

Fallos de RPC

Las llamadas a procedimiento remoto se ejecutan al menos una vez:

- pueden realizar el trabajo remoto correctamente
- o pueden fallar retornando una excepción

Si ocurre un error de comunicaciones se eleva la excepción:

- `System.RPC.Communcantion_Error`

Excepciones

Cualquier excepción elevada en un método o llamada a procedimiento remoto se propaga al que ha hecho la llamada, manteniendo la semántica de Ada

Ejemplo:

- Partición 1: Internal, RemPkg1 y RemExcMain
- Partición 2: RemPkg2

Comentario:

- una excepción visible sólo en la partición 1 es propagada a la partición 2, tratada con `when others` y enviada de nuevo a la partición 1 en la que vuelve a ser visible

Excepciones (cont.)

```
package Internal is
   Exc : exception;
end Internal;

package RemPkg2 is
   pragma Remote_Call_Interface;
   procedure Subprogram;
end RemPkg2;

package RemPkg1 is
   pragma Remote_Call_Interface;
   procedure Subprogram;
end RemPkg1;
```


Excepciones (cont.)

```
with Ada.Text_IO, Ada.Exceptions;
use Ada.Text_IO, Ada.Exceptions;
with RemPkg2, Internal;

procedure RemExcMain is

begin
   RemPkg2.Subprogram;
   exception when E : Internal.Exc =>
      Put_Line (Exception_Message (E)); -- Output "Message"
end RemExcMain;
```


Excepciones (cont.)

```
with RemPkg1, Ada.Exceptions;
use Ada.Exceptions;

package body RemPkg2 is
   procedure Subprogram is
   begin
      RemPkg1.Subprogram;
   exception when E : others =>
      Raise_Exception (Exception_Identity (E),
                       Exception_Message (E));
   end Subprogram;
end RemPkg2;
```

Excepciones (cont.)

```
with Internal, Ada.Exceptions;
use Ada.Exceptions;

package body RemPkg1 is

   procedure Subprogram is
   begin
      Raise_Exception (Internal.Exc'Identity, "Message");
   end Subprogram;

end RemPkg1;
```


RPCs asíncronas: definición RCI

```
package AsynchronousRCI is
   pragma Remote_Call_Interface;

   procedure Asynchronous (X : Integer);
   pragma Asynchronous (Asynchronous);

   procedure Synchronous  (X : Integer);

   type AsynchronousRAS is access procedure (X : Integer);
   pragma Asynchronous (AsynchronousRAS);

end AsynchronousRCI;
```


RPCs asíncronas: definición RT

```
package AsynchronousRT is
   pragma Remote_Types;

   type Object is tagged limited private;
   type AsynchronousRACW is access all Object'Class;
   pragma Asynchronous (AsynchronousRACW);

   procedure Asynchronous (X : Object);
   procedure Synchronous  (X : in out Object);
   function Create return AsynchronousRACW;

private
   type Object is tagged limited null record;
end AsynchronousRT;
```

RPCs asíncronas: uso de RCI

Dependiendo del subprograma apuntado la llamada puede ser síncrona o asíncrona

```
with AsynchronousRCI;
use AsynchronousRCI;
procedure AsynchronousMain is
   RAS  : AsynchronousRAS;
begin
   --  Asynchronous Dynamically Bound Remote Call
   RAS := AsynchronousRCI.Asynchronous'Access;
   RAS (0);  --  Abbrev for RAS.all (0)
   --  Synchronous Dynamically Bound Remote Call
   RAS := AsynchronousRCI.Synchronous'Access;
   RAS (0);
end AsynchronousMain;
```


RPCs asíncronas: uso de RT

Si el método tiene sólo parámetros `in` la llamada es asíncrona y si no síncrona

```
with AsynchronousRT;
use AsynchronousRT;

procedure AsynchronousMain is
   RACW : AsynchronousRACW := Create;
begin
   --  Asynchronous Dynamically Bound Remote Call
   Asynchronous (RACW.all);
   --  Synchronous Dynamically Bound Remote Call
   Synchronous (RACW.all);
end AsynchronousMain;
```


Unidades genéricas

Las unidades genéricas se pueden categorizar pero no heredan automáticamente la categoría

```
generic
package GenericRCI is
   pragma Remote_Call_Interface;
   procedure P;
end GenericRCI;

with GenericRCI;
package RCIInstantiation is new GenericRCI;
pragma Remote_Call_Interface (RCIInstantiation);

with GenericRCI;
package NormalInstantiation is new GenericRCI;
```

Llamadas remotas concurrentes

En la especificación del PCS no se define cuándo uno o más threads de control deben estar disponibles para procesar los mensajes que llegan y esperar a la terminación del RPC

Sin embargo, el PCS debe ser reentrante permitiendo llamadas concurrentes que den servicio a peticiones remotas

En GLADE esto significa que:

- en la implementación del PCS se maneja un ***pool*** de tareas de manera transparente al usuario

Aborto y terminación

Aborto:

- si un constructor que contiene una llamada remota se aborta, la llamada al subprograma remoto se cancela
- la implementación decide si la ejecución se aborta inmediatamente o no como resultado de la cancelación

Terminación:

- una partición activa termina cuando termina su tarea de entorno
- una partición no puede terminar antes de que el programa Ada en sí mismo termine

Configuración

El DSA no describe la forma en la que se tiene que configurar una aplicación distribuida:

- es el usuario el que tiene que definir cómo son las particiones de su programa y en qué máquinas van

La herramienta `gnatdist` y su lenguaje de configuración permite al usuario partir su programa y especificar la máquina en la que ejecutará cada partición

`gnatdist` lee un fichero de configuración y construye un ejecutable por cada partición definida

Configuración (cont.)

La forma de configurar y distribuir una aplicación será:

- Escribir una aplicación Ada no distribuida y usar los pragmas de categorización para indicar los paquetes que se pueden llamar remotamente
- Cuando la aplicación funcione correctamente, escribir el fichero de configuración que mapea los paquetes en particiones (RCIs y RTs, e identificar el procedimiento principal)
- Llamar a: `gnatdist <fichero de configuración>`
- Ejecutar la aplicación distribuida de acuerdo con el modo de arranque elegido

Formato de llamada a *gnatdist*

```
gnatdist [switches] configuration-file [list-of-partitions]
```

- los switches de `gnatdist` son los mismos que los de `gnatmake`
- por defecto la salida es un informe de las acciones realizadas
- el switch `-n` permite saltar la primera etapa de recompilación de la aplicación no distribuida
- los nombres de los ficheros de configuración deben acabar con `.cfg`
- puede haber varios ficheros de configuración para la misma aplicación distribuida
- si se da una lista de particiones en la línea de comandos, sólo se construyen las particiones indicadas

Funcionamiento interno de *gnatdist*

- Cada unidad de compilación del programa se compila en un módulo objeto (como en las aplicaciones no distribuidas) llamando a `gnatmake` sobre los fuentes de las distintas particiones
- Los *stubs* y los *skeletons* se compilan en módulos objeto con varios chequeos de los tiempos de creación de los ficheros para evitar recompilaciones inútiles
- `gnatdist` realiza unos cuantos chequeos de consistencia; en particular:
  - que todos los paquetes RCI y Shared Passive estén mapeados en particiones,
  - y que estén mapeados sólo en una partición

Funcionamiento interno de *gnatdist* (cont.)

- Por último, se crean los ejecutables para cada partición:
  - el código de carga de las particiones se integra en la partición principal, excepto si se ha especificado otra opción
  - en este último caso se genera un ***shell script*** (o nada) para arrancar las particiones en la máquina apropiada

Lenguaje de configuración de GLADE

Se trata de un lenguaje diseñado como la norma Ada

Este lenguaje evoluciona junto con GLADE (ahora congelado) y se reutiliza en PolyORB (no completamente)

Los aspectos más destacados de este lenguaje son:

- Nuevas palabras reservadas
- Nuevos pragmas y cláusulas de representación
- Declaración de la configuración
- Declaración de partición

Lenguaje de configuración de GLADE (cont.)

- Declaración de localización
- Atributo *Main*
- Pragma *Starter*
- Pragma *Boot_Location*
- Atributo *Self_Location*
- Atributo *Passive*
- Atributo *Data_Location*
- Atributo *Allow_Light_PCS*
- Pragma *Priority*
- Atributo *Priority*

Lenguaje de configuración de GLADE (cont.)

- Atributo *Host*
- Pragma *Import*
- Atributo *Directory*
- Atributo *Command_Line*
- Atributo *Termination*
- Atributo *Reconnection*
- Declaración de canal
- Atributo *Filter* para partición y canal
- Pragma *Registration_Filter*
- Pragma *Version*
- Atributo *Task_Pool*

Nuevas palabras reservadas

- `Configuration`
  - encapsula una configuración
- `Partition`
  - tipo predefinido para declarar particiones
- `Channel`
  - tipo predefinido para declarar canales entre particiones

Para evitar conflictos entre Ada y GLADE estas palabras han sido reservadas incluso si no se está usando el lenguaje de configuración

Nuevos pragmas y cláusulas de representación

Es posible modificar el comportamiento por defecto de una configuración mediante la definición de un pragma:

```
PRAGMA ::=
   pragma PRAGMA_NAME [(PRAGMA_ARGUMENTS)];
```

También es posible modificar el comportamiento por defecto de particiones o canales mediante atributos aplicados a los tipos:

```
REPRESENTATION_CLAUSE ::=
   for Partition'ATTRIBUTE_NAME use ATTRIBUTE_ARGUMENTS;
  |for Channel'ATTRIBUTE_NAME use ATTRIBUTE_ARGUMENTS;
```

Nuevos pragmas y cláusulas de representación (cont.)

Además, se puede modificar el comportamiento por defecto de particiones o canales mediante atributos aplicados a las propias particiones o canales:

```
REPRESENTATION_CLAUSE ::=
   for PARTITION_IDENTIFIER'ATTRIBUTE_NAME use
                              ATTRIBUTE_ARGUMENTS;
```

Cuando se aplica una cláusula de definición de atributo a un objeto dado de un tipo predefinido, se sobrescribe la definición del tipo predefinido

Declaración de la configuración

La unidad de configuración tiene una parte de especificación y una parte opcional, y se declara como si fuera un procedimiento

Se utiliza la palabra reservada `configuration`

```
CONFIGURATION_UNIT ::=
   configuration IDENTIFIER is
      DECLARATIVE_PART
   [begin
      SEQUENCE_OF_STATEMENTS]
   end [IDENTIFIER];
```


Declaración de partición

En la parte declarativa de la configuración se pueden declarar particiones, cada una de las cuales se puede inicializar con una lista de unidades Ada:

- se utiliza la palabra reservada `partition`
- una vez declarada la partición es una lista vacía de unidades Ada
- el operador `:=` permite añadir unidades a la lista

```
PARTITION_DECLARATION ::=
   DEFINING_IDENTIFIER_LIST : Partition
      [:= ENUMERATION_OF_ADA_UNITS];
```

Declaración de localización

Las clases de localización que se definen en GLADE son:

- `Boot_Location`: define la localización de red a usar para comunicarse con el servidor de arranque (boot server) durante la fase de arranque (boot)
- `Self_Location`: define la localización de red a usar por otras particiones para comunicar con la actual
- `Data_Location`: define la localización de almacenamiento de datos usado por la partición actual para mapear sus unidades Shared Passive

La localización de red se compone de un protocolo como tcp y unos datos de protocolo como `máquina:puerto`

Atributo *Main*

En una aplicación distribuida se pueden tener varios ejecutables, pero pensando en una versión no distribuida de la aplicación se podría tener un ejecutable principal responsable de arrancar la aplicación completa

El lenguaje de configuración permite declarar un procedimiento principal como en una aplicación no distribuida:

```
MAIN_PROCEDURE_DECLARATION ::=
   procedure MAIN_PROCEDURE_IDENTIFER is in
                                        PARTITION_IDENTIFIER;
```

La partición en la que se mapea el procedimiento principal se llama partición principal

Atributo *Main* (cont.)

La partición principal además contiene el servidor de arranque

El procedimiento principal de las otras particiones tiene un cuerpo nulo, aunque el usuario puede proporcionar uno:

```
PROCEDURE_DECLARATION ::=
   procedure PROCEDURE_IDENTIFIER;

REPRESENTATION_CLAUSE :=
   for PARTITION_IDENTIFIER'Main use PROCEDURE_IDENTIFIER;
```

Pragma *Starter*

Por defecto el ejecutable principal carga el resto de particiones del programa

El pragma Starter permite elegir el modo de arranque:

```
CONVENTION_LITERAL ::= Ada   |
                       Shell |
                       None
PRAGMA ::=
   pragma Starter ([CONVENTION =>] CONVENTION_LITERAL);
```


Pragma *Starter* (cont.)

- **Ada**: por defecto las particiones se cargan desde la partición principal usando una shell remota
- **Shell**: el usuario puede pedir un script que arranque las diferentes particiones una a una en las máquinas remotas adecuadas usando una shell remota
- **None**: el usuario se encarga de arrancar manualmente las particiones, probablemente cada una desde la propia máquina en la que va a ejecutar

Atributo *Passive*

Una partición por defecto es activa

Este atributo permite definir una partición pasiva, en cuyo caso **gnatdist** comprueba que sólo se han mapeado unidades Shared Passive:

```
REPRESENTATION_CLAUSE :=
   for PARTITION_IDENTIFIER'Passive use BOOLEAN_LITERAL;
```

Pragma *Priority*

Controla la prioridad a la que se ejecuta la llamada remota:

- **por defecto se ejecuta a la prioridad del cliente `Client_Propagated`**
- **se puede especificar que se use una única prioridad en el servidor `Server_Declared`**

```
PRIORITY_POLICY_LITERAL ::= Server_Declared
                          | Client_Propagated

PRAGMA ::=
   pragma Priority ([Policy =>] PRIORITY_POLICY_LITERAL);
```


Atributo *Priority*

Establece la prioridad del servidor para la política de gestión de llamadas remotas `Server_Declared`:

- **por defecto es la prioridad de la tarea anónima que hace el servicio**

```
REPRESENTATION_CLAUSE ::=
   for PARTITION_IDENTIFIER'Priority use INTEGER_LITERAL;
```


Declaración de canal

El lenguaje de configuración permite describir las particiones y también los canales, que son enlaces bidireccionales entre dos particiones

```
CHANNEL_DECLARATION ::=
   CHANNEL_IDENTIFIER : Channel
      [:= PARTITION_PEER];
PARTITION_PEER ::= (PARTITION_IDENTIFIER,
                    PARTITION_IDENTIFIER);
```

Ejemplo:

```
   A_Channel : Channel := (Partition_1, Partition_2);
```

Entre dos particiones sólo puede haber un canal

Atributo *Filter* para partición y canal

Los filtros realizan de manera transparente transformaciones de los datos enviados y recibidos en las llamadas entre particiones:

- por ejemplo, compresiones y descompresiones

Por defecto no se aplican filtros

Los filtros se aplican sobre los canales, por lo que es necesario definirlos previamente

Se puede especificar un filtro para un canal:

- todos los datos del canal usan el filtro

```
A_Channel : Channel := (Partition_1, Partition_2);
for A_Channel'Filter use "ZIP";
```


Atributo *Filter* para partición y canal (cont.)

También se puede especificar un filtro para una partición:

- los datos que gestione la partición en las llamadas remotas usarán el filtro

```
for Partition_1'Filter use "ZIP";
```

O para todas las particiones:

```
for Partition'Filter use "ZIP";
```

Finalmente, se puede sobrescribir el uso de los filtros:

```
My_Channel : Channel := (Partition_1, Partition_2);
for My_Channel'Filter  use "ZIP";
for Partition_1'Filter use "Some_Other_Filter";
```

- `Partition_1` usa `Some_Other_Filter` excepto para la comunicación con `Partition_2` que usa ZIP

Pragma *Registration_Filter*

Se usa para el envío de algunos parámetros que requieren algunos filtros necesarios para su funcionamiento

```
PRAGMA ::=
   pragma Registration_Filter ([Filter =>] STRING_LITERAL);
```

Atributo *Task_Pool*

El pool de tareas se puede configurar a través de este atributo

```
REPRESENTATION_CLAUSE ::=
   for PARTITION_IDENTIFIER'Task_Pool use
                                      TASK_POOL_SIZE_ARRAY;

TASK_POOL_SIZE_ARRAY ::=
  (NATURAL_LITERAL,  --  Task Pool Minimum Size
   NATURAL_LITERAL,  --  Task Pool High Size
   NATURAL_LITERAL); --  Task Pool Maximum Size
```

Ejemplo:

```
   for Partition'Task_Pool use (3, 8, 15);
```


Atributo *Task_Pool* (cont.)

Los tres valores que se especifican son:

- Tamaño mínimo del pool:
  - número de tareas anónimas que se crean al principio y están siempre disponibles
  - si se necesitan más se crean dinámicamente.
- Techo del pool:
  - cuando una RPC se completa, la tarea anónima se destruye si el tamaño del pool es superior a este valor; si no lo es la tarea se encola
- Tamaño máximo del pool:
  - este valor es el máximo absoluto de tareas del pool, es decir, el número máximo de RPCs que se pueden hacer concurrentemente
  - si se agotan las tareas la petición se encola