Interchangeable Scheduling Policies in Real-Time Middleware for Distribution

Juan López Campos (lopezju@unican.es)
J. Javier Gutiérrez (gutierjj@unican.es)
Michael González Harbour (mgh@unican.es)

Computers and Real-Time Group, University of Cantabria

Funded by the Spanish Government (TIC2002-04123-C03-02,TRECOM)
and by the European Commission (IST-2001-34820, FIRST)

11th International Conference on Reliable Software Technologies
Ada-Europe 2006
Porto, Portugal 5 - 9 June

Introduction

Current middleware for real-time distribution is based on fixed-priority scheduling

- RT-CORBA
- Ada DSA

Complexity in embedded systems introduces new requirements for scheduling

- reservations
- QoS
- flexible scheduling

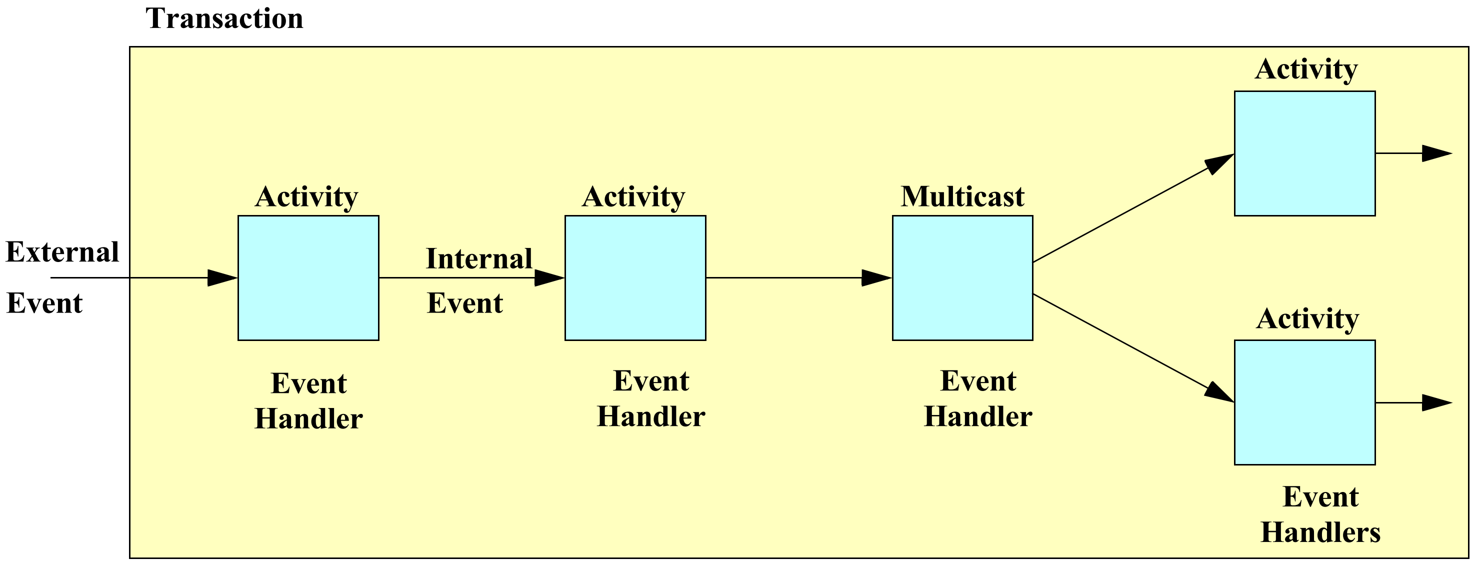
Complex scheduling policies require complex parameters

- a single priority or deadline is not enough

Objectives

The objective is to support in distribution middleware:

- different scheduling policies
- complex scheduling parameters
- a mechanism to express the scheduling parameters in the transactional model



Starting Point and Background

MaRTE OS

- POSIX minimal RTOS for embedded applications

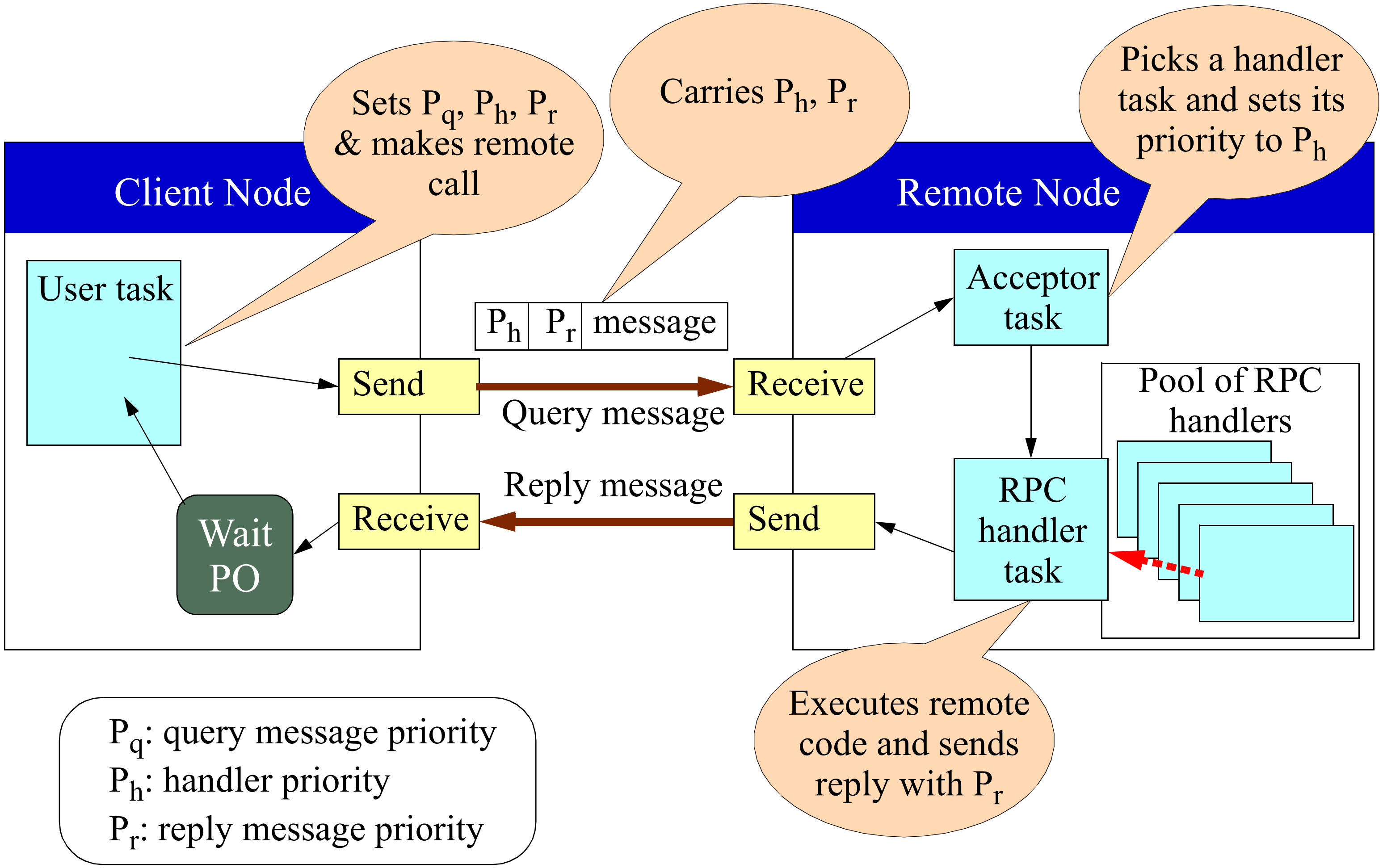
RT-EP

- Real Time Ethernet Protocol, fixed-priority scheduling based on token passing on a logical ring

RT-GLADE on top of MaRTE and RT-EP

- Modification of GLADE: implementation of the Distributed Systems Annex of Ada (DSA) by GNAT
  - enhances the real-time capabilities
  - allows a free assignment of priorities to RPCs

Priorities Mechanism in RT-GLADE



P_q: query message priority

P_h: handler priority

P_r: reply message priority

Drawbacks of the RT-GLADE Implementation

Only one scheduling policy: Fixed Priorities

The middleware manages the priorities only at the first level of the RPC:

- unable to express priorities in nested calls

There are some points of inefficiency:

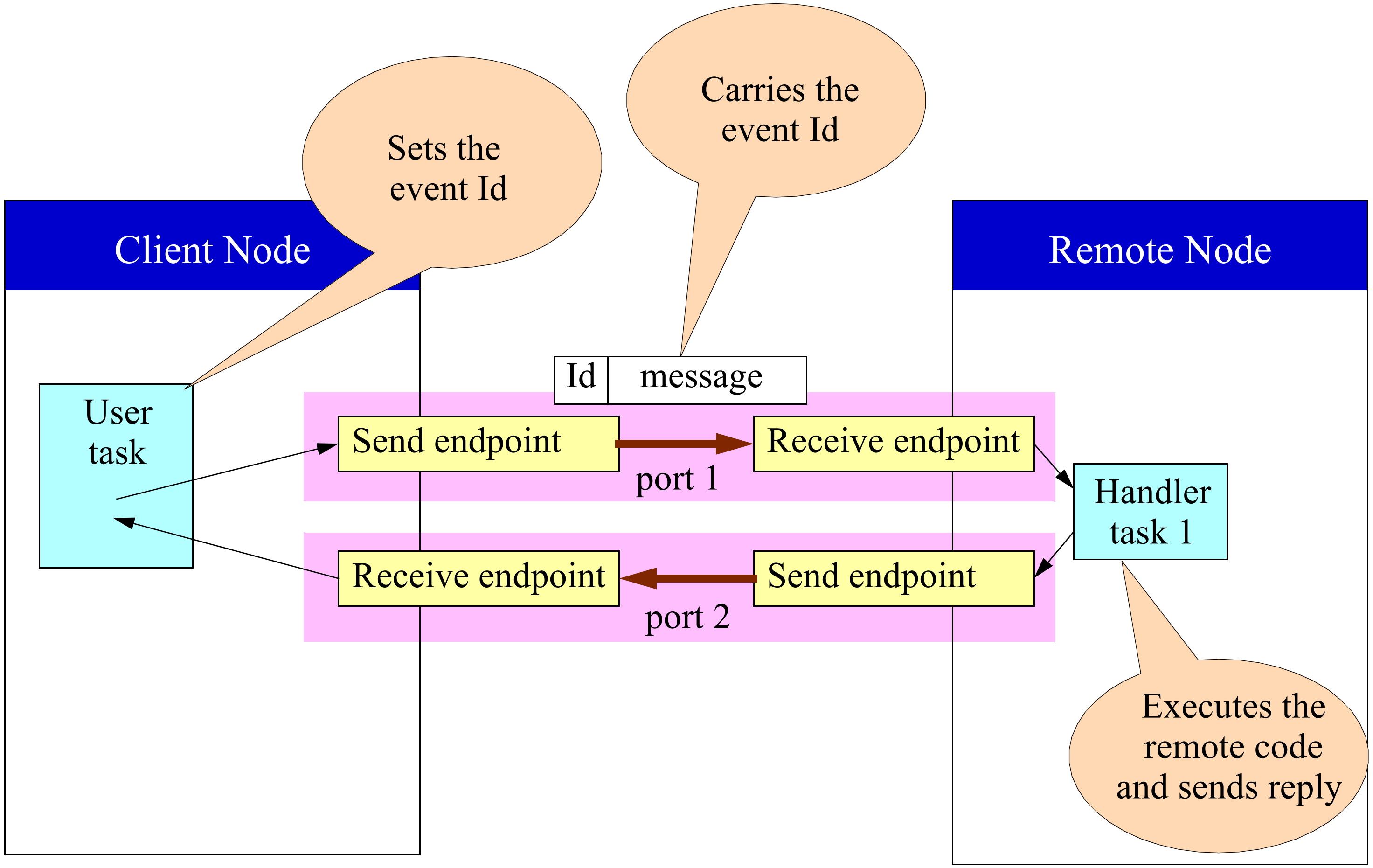
- transmission of scheduling parameters that are larger than a single number
- dynamic changes of scheduling parameters for RPC handlers could have a high cost
  - i.e., negotiation process

Proposed Solution

When instantiating a SW component:

- **Create the scheduling entities necessary to execute the RPCs**
- **Processors:**
  - **RPC Handlers at the server side**
  - **explicit creation of the necessary tasks with their appropriate scheduling parameters**
- **Networks:**
  - **communication ports to be used**
  - **creation of *endpoints* to send and receive messages**
  - **the scheduling parameters are linked to the send endpoints**

Proposed Solution: Event ID



Communication Layer of RT-GLADE

UNIVERSIDAD DE CANTABRIA

Main changes to previous version:

- Acceptor tasks removed:
  - RPC Handler tasks wait directly on the network
- RPC handler pool removed:
  - they are created statically when required
- The waiting mechanism for the RPC reply has been removed:
  - the client task waits directly on the network

Simpler architecture that supports complex scheduling policies

Adding Support for a New Scheduling Policy

Extend the abstract tagged types

- Task_Scheduling_Parameters
  - scheduling parameters of an RPC handler task
- Message_Scheduling_Parameters
  - scheduling parameters used for the messages sent through an endpoint

And implement the private primitives

- procedure Create_Task
- procedure Create_Send_Endpoint

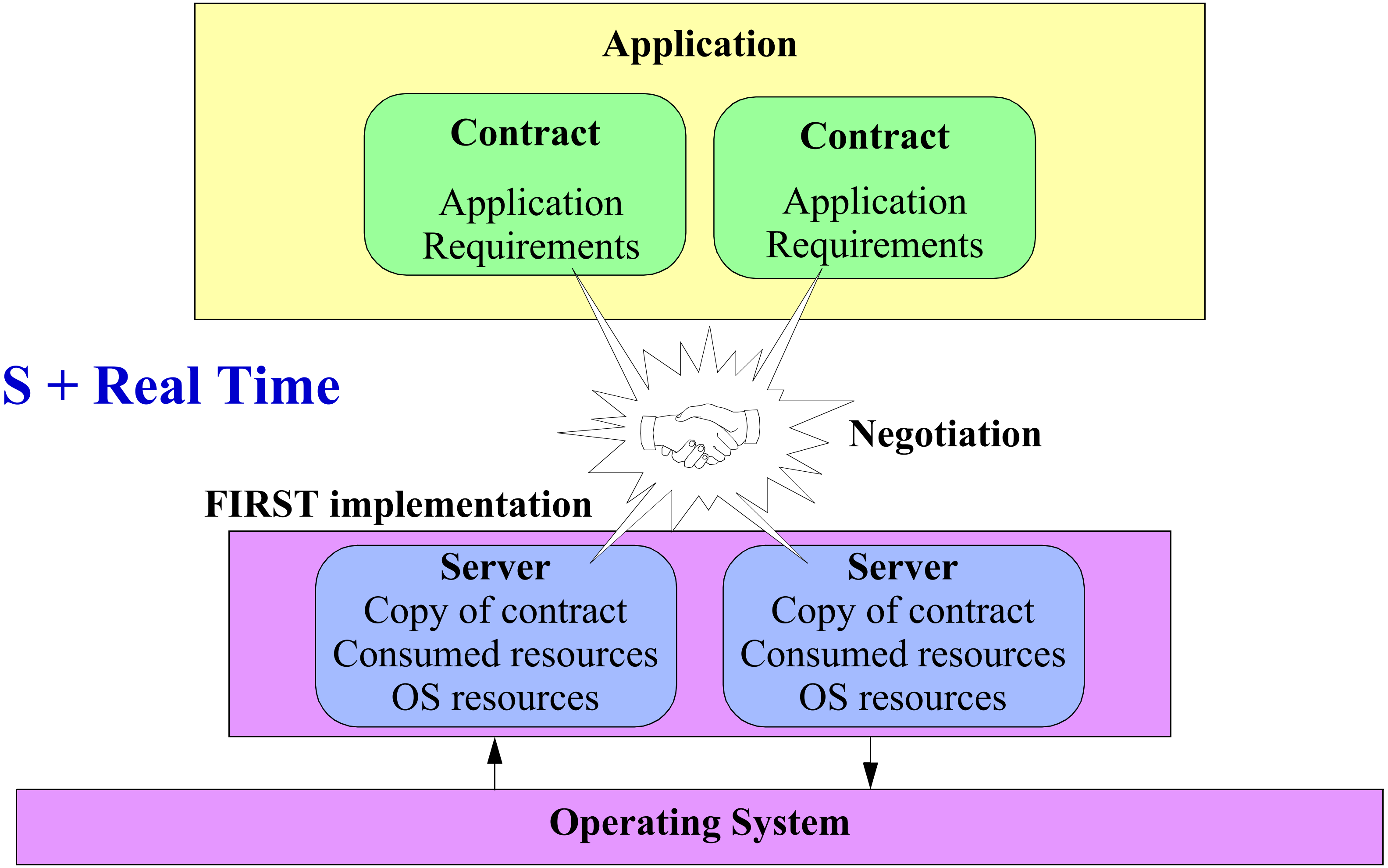
Implementation of Specific Policies

We have implemented two policies:

- Fixed priorities
  - the only parameter is a single task or message priority
- FIRST contracts
  - complex contract-based scheduling policy requiring negotiation

FIRST Scheduling Framework

QoS + Real Time


Application

Contract

Application Requirements

Contract

Application Requirements

Negotiation

FIRST implementation

Server

Copy of contract

Consumed resources

OS resources

Server

Copy of contract

Consumed resources

OS resources

Operating System

Actions to Initialize Software Components Involved in an RPC

Choose an unused

- event id
- remote port
- client node port

In the client’s node

- `Create_Query_Send_Endpoint`
- `Create_Reply_Receive_Endpoint`

In the remote node

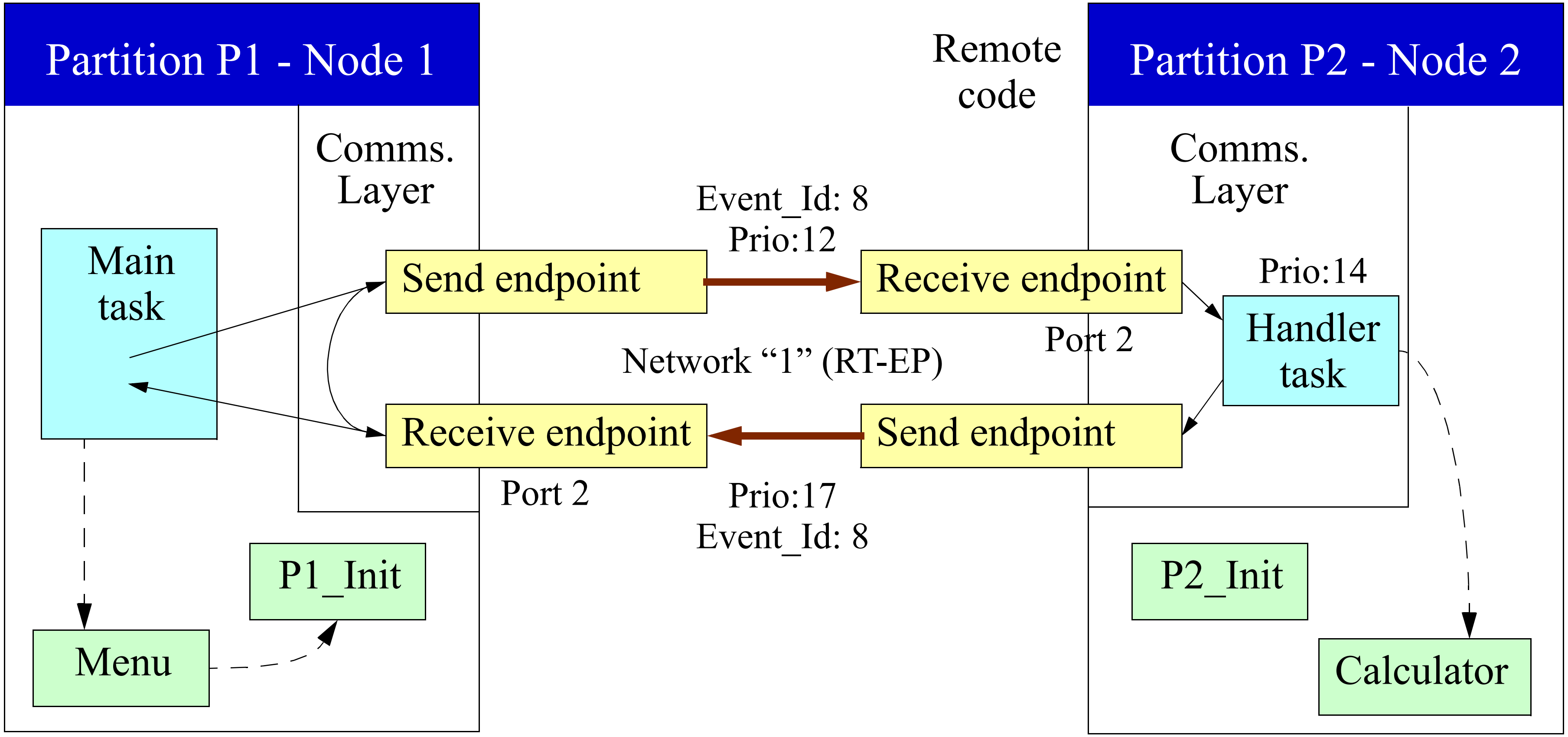
- `Create_RPC_Handler` (with its attached receive endpoint)
- `Create_Reply_Send_Endpoint`

Actions to make an RPC

Set the *event id* for the transaction

- `Set_Event_Id`

Usage Example (fixed priorities)



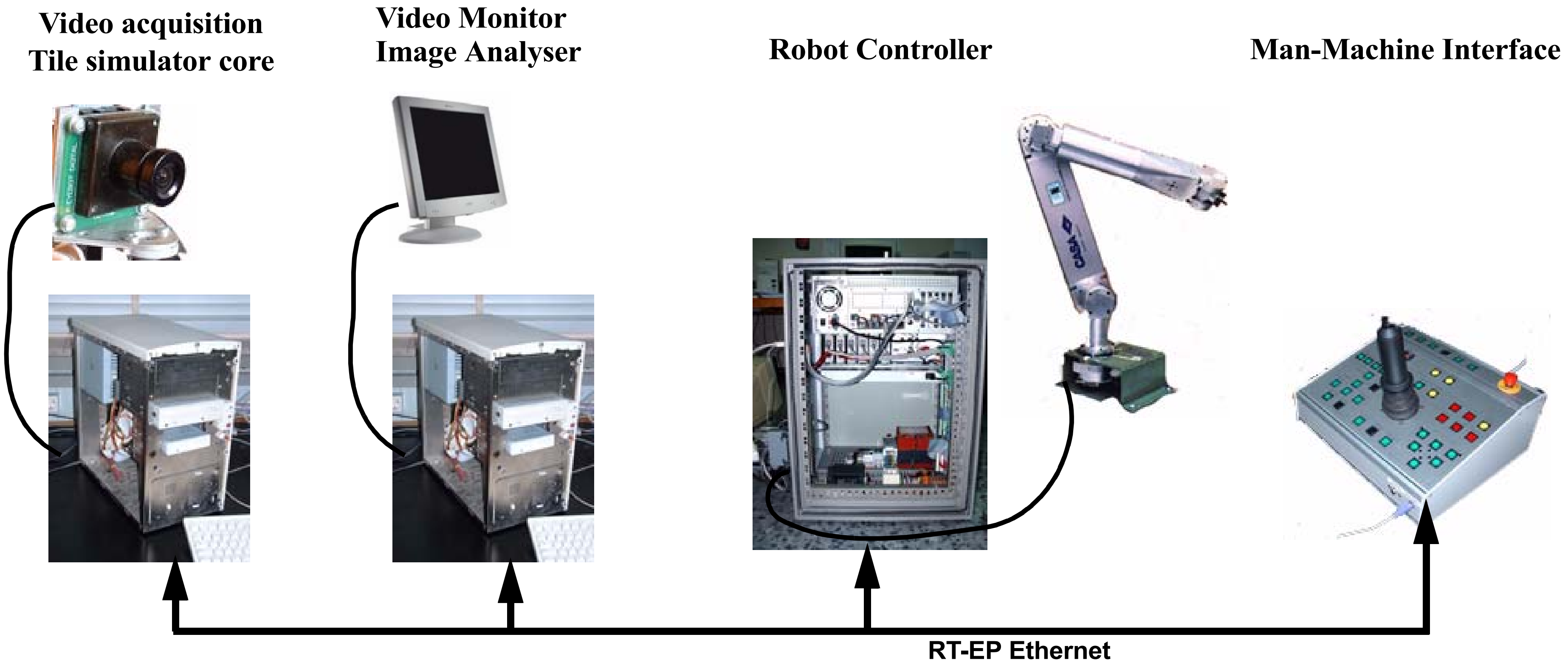
Usage Example (cont’d)

```
package Calculator is
   pragma Remote_Call_Interface;
   function Add (A : in Integer; B : in Integer) return Integer;
end Calculator;

with Calculator, RT_Glade_Event_Id_Handling, P1_Init;
procedure Menu is
   A,B,Sum:Integer;
begin
   -- Initialize the communications endpoints
   P1_Init;
   -- Set the event id for subsequent RPCs
   Rt_Glade_Event_Id_Handling.Set_Event_Id(8);
    loop
      ...
      Sum := Calculator.Add (A, B); -- remote call
      ...
    end loop;
end Menu;
```

Usage Example (cont’d)

```
with Ada.Real_Time, Rt_Glade_Scheduling;
with Rt_Glade_Scheduling.Priorities;
procedure P1_Init is
   R_Message_Params :
      Rt_Glade_Scheduling.Message_Scheduling_Parameters_Ref;
begin
    -- Create the send endpoint for the query messages
   R_Message_Params:= new
      Rt_Glade_Scheduling.Priorities.Message_Priority’
          (Rt_Glade_Scheduling.Message_Scheduling_Parameters
           with Endpoint_Priority => 12);
   Rt_Glade_Scheduling.Create_Query_Send_Endpoint
      (Params => R_Message_Params.all, Node => 2,
       Net => 1, Port => 2, Event => 8);

   -- Create the receive endpoint for the reply messages
   Rt_Glade_Scheduling.Create_Reply_Receive_Endpoint
      (Net => 1, Port => 2, Event => 8);
end P1_Init;
```

Implementation and Evaluation



Video acquisition
Tile simulator core

Video Monitor
Image Analyser

Robot Controller

Man-Machine Interface

RT-EP Ethernet

+181 lines of code in an application of 13000 lines

Discussion

Explicit creation of RPC handlers and message endpoints

- should be a minor problem in real-time systems, in which a "transparent architecture" is unwise

Explicit setting of event id's

- no problem; we already had explicit setting of priorities
- added benefit for transactions with nested RPCs

Pool of RPC handlers is replaced by a possibly larger set of explicitly created handlers

- although RPC handlers may be shared by several remote operations

Conclusions

Proposed a new mechanism for distribution middleware

- to support different scheduling policies
  - including complex contract-based policies
  - implementation of middleware is independent of the scheduling policy
- to support real-time transactions with nested remote calls

Implementation for fixed priorities and FIRST contracts in Ada

- simpler implementation architecture
- at the expense of more tasks and network access points

More information in http://www.ctr.unican.es/