User's guide

PolyORB 2007 + RTC/RATO RTEP

Authors: *Hector Perez Tijero, J. Javier Gutierrez*
Universidad de Cantabria, May 2008

Index

Diagrams index

Introduction

PolyORB is a distribution middleware written in Ada language. It is often called schizophrenic middleware since it allows to communicate different distribution standards such as Ada DSA or CORBA. Those distribution models has been integrated in hard real-time systems through some extensions that allow to make the software predictable.

This middleware has been compiled and tested in different platforms like Windows, Linux or Solaris. However, there is not a current full real-time operating system support to take advantage of the RTCORBA or DSA facilities in such systems. Furthermore, available protocols are just based on IP networks which are fast but not predictable.

Having those concepts in mind, it looks that there is a gap to fill in order to provide support to systems with timing requirements.

If you require further information about original middleware, please refer to native PolyORB user's guide.

Real-Time middleware

The main objective is providing a complete real-time platform to distributed applications written in Ada. PolyORB has been modified and adapted to hard real-time systems by adding new control policies and network protocols personalities. Main issues included are the following:

- New operating system support: PolyORB has been ported to MaRTE OS, a real-time kernel for embedded applications that follows the Minimal Real-Time POSIX.13 subset.
- New protocol personality: A new real-time network protocol is supported: RTEP, a software-based token-passing Ethernet protocol for multipoint communications in real-time applications that does not require any modification to existing Ethernet hardware.
- New control over tasks: Two complementary tasking policies (RATO and TPT) has been added to provide predictability to system.
- Complete real-time platform: PolyORB now is currently working on a really real-time platform. This includes an operating system (MaRTE OS), a network protocol (RTEP) and a middleware behaviour that permits a complete control over tasks in our system (RATO)

Application personalities

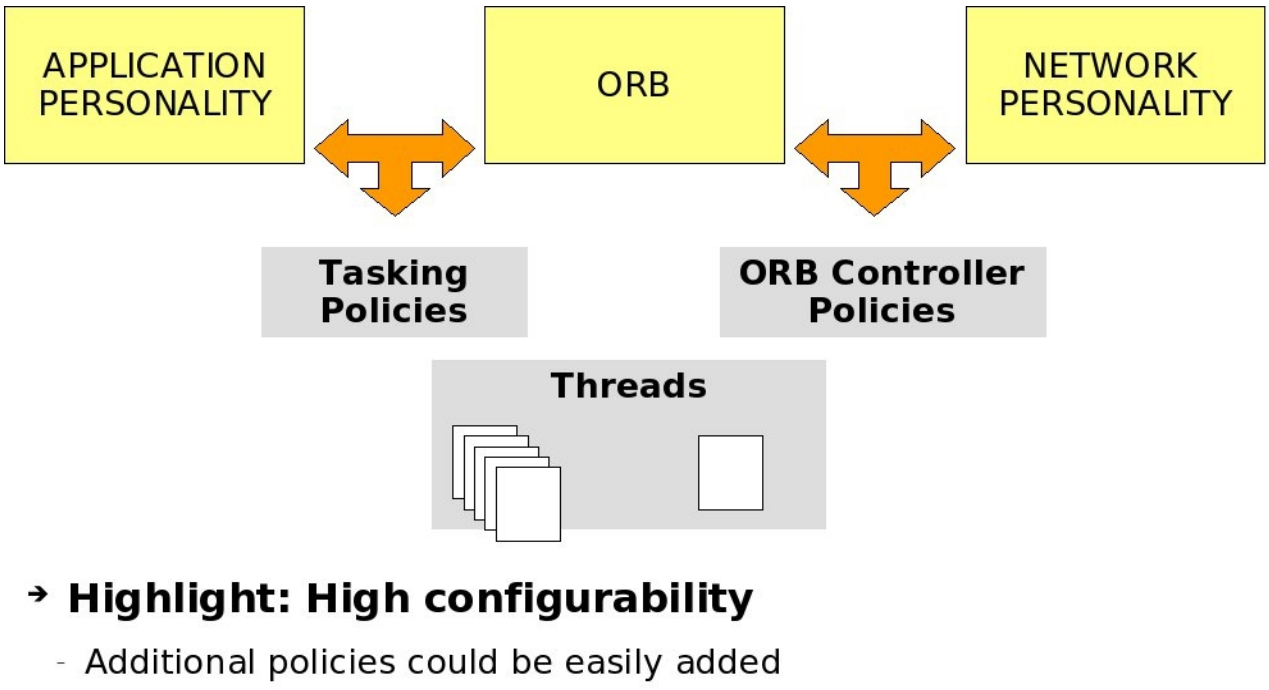
Current PolyORB version supports CORBA and DSA personalities. In the first case, real-time features are addressed via RTCORBA extension. In the second case, DSA is currently designed to make distributed applications with no real-time support.

In this work, we have just adapted CORBA personality to our real-time platform. Nowadays, we are working in adapting DSA and implementing mechanisms to support real-time features.

Control policies

The internal behaviour of the middleware depends on the tasking policies choice. There are several options but here we will address those intended to real-time systems. For further information on different policies, please refer to original users' guide.

There are two levels of control: orb tasking and orb controller. The first one is more related to the semantics of the message exchanged between local and remote nodes. Therefore, it will manage the task behaviour depending on the kind of message received. Respecting to the orb controller, it is more related to ORB internal operations. They are not orthogonal between them so take care of your selection



Control policies overview

RATO

ReAdy To Go (RATO) is a policy to control ORB tasks. Generally speaking, it manages the ORB main loop, I/O events and requests processing. In shorter words, it determines which will be next job to a given task.

RATO add a complete control over tasks in system. Its main characteristic is processing a complete request without context switching by making task-endpoint associations.

TPT

Under Thread per Target policy new threads are not created after configuration time. In fact, it is a static thread pool that allows associate some information at the beginning to make task-endpoint associations (therefore not anonymous tasks).

Both policies, TPT and RATO, are intended to be used together since both are looking for the same objective (neither context switching nor anonymous tasks).

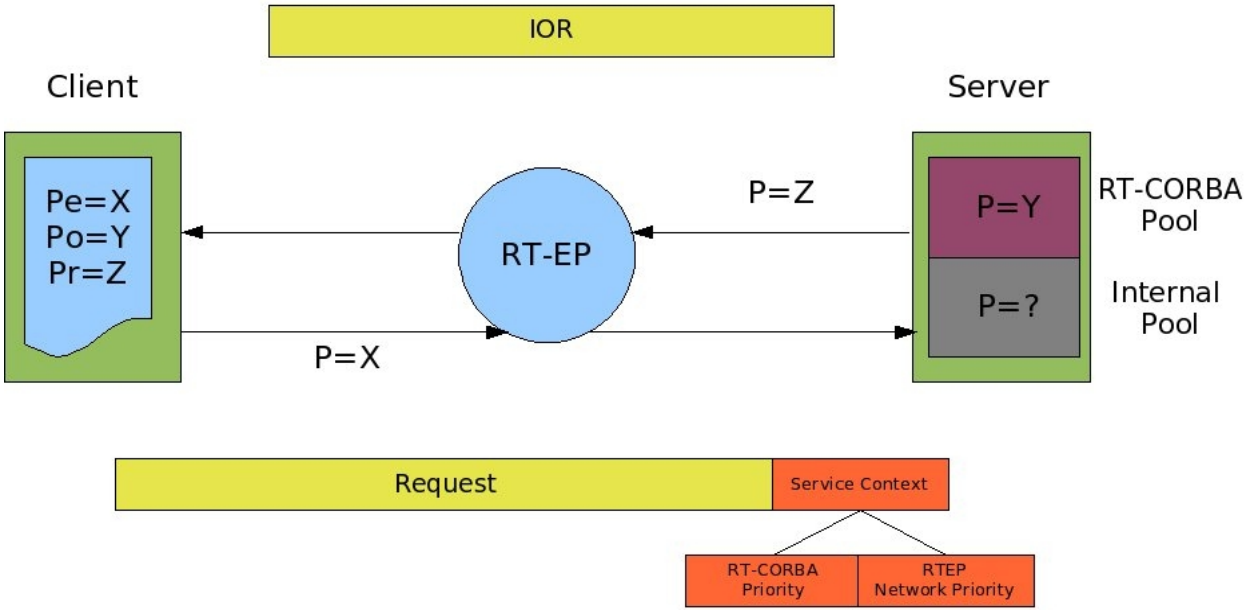
Network protocol personalities

Protocol personalities adapts network events to middleware messages. In our case, we have been working with CORBA generic protocol (GIOP) in the transport layer and a Real-Time Ethernet Protocol based on token passing (RTEP) in the protocol level.

RTC-RTEP

This personality has been developed just for test purposes. It can be used with all the original PolyORB environment (orb tasking and controller policies, RTCORBA personality to address real-time features, etc). This protocol layer adds support for network priorities in the whole process (request and reply). Those priorities are configured through an API detailed in later chapters. Furthermore, reply priority is sent as service context data in the network message.

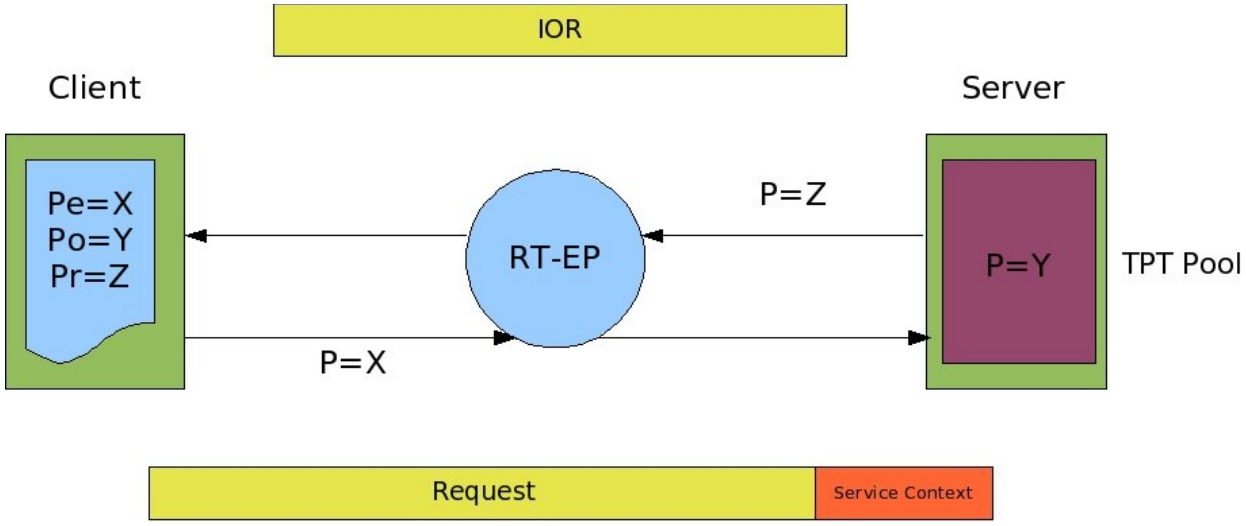
Since original PolyORB does not take into account priorities at ORB level, there is a gap in the requests management which is prone to priority inversion.



RATO-RTEP

This personality takes advantage of RATO and TPT facilities to manage network messages and apply scheduling parameters more efficiently. In contrast to RTC-RTEP, scheduling parameters are defined for the whole request and are not sent through the network. In this case all the scheduling parameters are set at configuration time through an API detailed later.

This configuration does not need RTCORBA environment since real-time features are addressed establishing scheduling parameters previously. This philosophy looks more suitable to distributed hard real-time systems with few dozens of nodes or when scheduling parameters are complex to send them through the network or even when the cost of context switching related to scheduling parameters (i.e. contracts) associated is too high.



Installation

The currents installation is based on several patches that add several files and modify compiling archives mainly. Source code of original software remains almost untouched. In this version, just CORBA personality with no services has been ported to MaRTE OS platform. Please refer to *Future Work* section if you require more information about other utilities.

Pre-requisites:

- GNAT 2007
- MaRTE OS 1.72 (December 2007)
- PolyORB 2007
- Autoconf 2.57 or newer, Automake 1.6.3 or newer, and Libtool 1.5.8 or newer

1. Copy *MaRTE_PolyORB.patch* file and the installation script (*apply_patch_marte*) to MARTE_DIR (no previous installation is required).

   Example: *If MaRTE OS is in /home/usr1/marte, copy both files to /home/usr1/marte*

2. Run *apply_patch_marte* script
3. Install MaRTE OS as usual (*minstall*) and add the *utils* subdirectory to your PATH environment variable
4. Go to MARTE_DIR/polyorb_config and run *make* in order to compile some extras files related to PolyORB and miscelaneous utilities
5. Copy *PolyORB.patch* file and the installation script (*apply_patch_polyorb*) to POLYORB_DIR (no previous installation is required).
6. Run *apply_patch_polyorb* script
7. Re-configure PolyORB running in POLYORB_DIR:

   1. *./support/reconfig*
   2. *./configure –prefix=/wherever_you_want_to_install –with-appli-perso=”corba”*

   You can add *–enable-debug* flag to configure script in order to work with debugging symbols. DSA personality is not available yet for MaRTE OS platform .

8. Run *make* in POLYORB_DIR
9. Run *make install* in POLYORB_DIR
10. Compile examples after configuring POLYORB_INSTALL_DIR in Makefile (See details inside)

Future Work

Our current work is focusing on:

- Porting to MaRTE OS the following features:
  - DSA personality
  - Naming service
- Enable distributed transaction support
- Optimize source code and control policies to reduce unnecessary overhead

Appendix A: Other features

Debug facilities

Since MaRTE OS does not have a file system support yet, a basic debug package has been developed in order to enable debug messages compiling independent.

Package PolyORB.API.RTEP_MAC contains the API to manage it. It is just one procedure called *Enable_Debug* which enable/disable different parts of the software developed:

- ***procedure Enable_Debug (VARIABLES);***

Variables to configure:

Transport : Network level information: Destination, Channel, Priority

Asynch : Registering event of sources in ORB

Lanes : Thread behaviour in RTPOA threadpool

Leader_Followers : Thread behaviour in L&F policy

RATO : Thread behaviour in RATO policy

QoS : QoS RTEP Parameters info (Service Context environment)

TPT : Thread behaviour in Thread Per Target policy

Init : Info about initialization process

References : Info about binding and reusable objects (not used by now)

Buffers : Buffer store debug

Example of use: *To enable transport debug facilities please call*

Enable_Debug (Transport => True);

Naming server

This CORBA service has been ported to MaRTE since it is completely necessary to DSA

personality. However, it has not been released yet. Please contact us if you require more information.

Configuration File

Configuration_file.ads is in polyorb-config MaRTE's subdirectory. Since MaRTE OS does not provide full file system support to store polyorb configuration file (polyorb.conf), some options has to be read from this ads file. This will be changed once FAT driver for MaRTE OS will be finished in order to follow the original programming philosophy. Parameters to set up:

- *Num_Threads_In_Pool:* Maximun number of threads in pool created by Thread Per Target policy. Each thread is created throguh API.Scheduling, not at init phase. This configuration paramenter is just to take some control in *TPT,* but in Thread Pool policy manage the number of threads in the system.
- *Num_Monitors*: Maximun number of monitors in system. It must be given the same value as Num_Threads + 1 for Thread per Target Policy (one task per monitor and background task) but this value is critic in performance due to its presence in many cycles so it's a good habit to tune both to our requirements.
- *Naming_Service_IOR:* Interoperable Object Reference of the name server in use. It must be selected by server in order to register remote object. Furthermore, it must be selected by client in order to get remote object address. Specify it in configuration files to pre-elaborate them.

Time measure facilities

In order to perform time measures of the software a special package has been developed. It is divided into two parts depending on the output: console and ethernet. The API is not included in PolyORB but in MaRTE OS subdirectory called polyorb-config. A time measure is set up with an ID so you can divide your code into small subparts to measure.

Console output

This is the most simple way to get some measures. The API includes:

- ***procedure Set_And_Reset_Time_Measures (Max_Number : in Natural);***

  Set the total number of measures you are expecting and reset other time variables

- ***procedure Take_Measure_Start (Id : Natural);***

  Set Id to the start time.

- ***procedure Take_Measure_Stop (Id : Natural);***

  Set Id to the stop time.

- ***procedure Store_Measure (Id : Natural;***

  Max_Time_Allowed: Time_Span;

  Top_Measures_Reached : out Boolean);

  Store measure marked as ID. You can specify in addition two parameters. *Max_Time_Allowed* allows you to put a top limit to avoid measures problems like network errors. *Top_Measures_Reached* is set to true when Max_Number has been reached (that is, measures taken above time limit are not computed).

- ***procedure Show_Average (Id : Natural);***

  Print time figures within ID measure and previously stored through Store_Measure: average, maximun and minimun times.

- ***procedure Show_Std_Deviation (Id : Natural);***

  Function to print standard deviation and number of measures included in an interval of 10% maximum.

Ethernet output

This is a more convenient way if you require to store your measures for future use or if you need to use another software to process the data. This API is just an Ada wrapper to posix time facilities included in MaRTE OS. Unlike the console mode, this measures are identified through a name which is sent through ethernet as well.

- ***function Reset_And_Init_Eth_Time_Measures (Name : String) return Integer;***

  Name will identify the measure in remote node. The return value will be the identification to take measures.

- ***procedure Take_Eth_Measure_Start (Id : Integer);***

  Set Id to the start time.

- ***procedure Take_Eth_Measure_Stop (Id : Integer);***

  Set Id to the stop time.

- ***procedure Send_Data;***

  Send data stored to host in broadcast mode.

Additional requirements are needed to run correctly this mode:

- Receiver software in host: Simple program to listen and store ethernet messages

  Ej: linux_eth_receive_log included in MaRTE examples/logger directory
- Include additional MaRTE OS files with posix time facilities to compile:
  - $(MARTE)/polyorb_config/measure_times_c_std.o
  - $(MARTE)/misc/time_measurement_posix.o
  - $(MARTE)/misc/logger.o

Appendix B: User's API

RTC-RTEP

There is just one procedure in the package to get it works. It is used in client side (local node) to configure remote host location and priorities support

- ***procedure Configure_RTEP_Parameters (IOR_Ref : in String;***
  P_in : in RTEP_MAC.Priority;
  O_out : in RTEP_MAC.Priority;
  P_out : in RTEP_MAC.Priority;
  Event_ID : in Natural;
  Node : in PolyORB.QoS.RTEP_Parameters.Node_Role);

Configure_RTEP_Parameters: Establish remote node info and new QoS parameters to be sent through the network.

IOR_Ref *:* It is the information published by remote objects to start the communication between nodes. The string has the format: “IOR:XXX”

P_in: Priority in the network to send the request

O_out: Priority of the object invocation in remote node (Deprecated since we use RTCorba standard to establish remote object priority)

P_out: Priority in the network to send the reply

Event_ID: Event identification

Node: Node role. It can take CLIENT or SERVER value (deprecated)

RATO-RTEP

Both nodes must be configured at configuration time. We have to specify not only scheduling parameters but also event and channel to complete the request. It is mandatory to use with TPT + RATO control policies.

- ***procedure Create_Receive_Endpoint***
  (Params : Message_Scheduling_Parameters_Ref;
  Dest_Node : RTEP_MAC.Station_ID;
  Event_ID : Natural;
  Channel : RTEP_MAC.Channel;
  Endpoint : out RTEP_MAC.Endpoint_Id);

Create_Receive_Endpoint: Create one task and associate it to a new endpoint together with the scheduling parameters defined.

Params: Generic scheduling params. They must be formatted via Set_Scheduling_Params
Dest_Node: Station to send replies. Each endpoint is associated to one station
Event_ID: Event identification
Channel: Waiting channel for incoming request
Endpoint: Endpoint identification

- ***procedure Create_Send_Endpoint***
  (Param : Message_Scheduling_Parameters_Ref;
  Dest_Node : RTEP_MAC.Station_ID;
  Event_ID : Natural;
  Channel : RTEP_MAC.Channel;
  Endpoint : out RTEP_MAC.Endpoint_Id);

Create_Send_Endpoint: Associate the current task executing the procedure to a new endpoint together with the scheduling parameters defined.

Params: Generic scheduling parameters. They must be formatted via Set_Scheduling_Params
Dest_Node: Station to send request. Each endpoint is associated to one station
Event_ID: Event identification
Channel: Waiting channel for the reply
Endpoint: Endpoint identification

- ***procedure Set_Scheduling_Params***
  (From_Params : in Message_Scheduling_Parameters;
  To_Params : in out Message_Scheduling_Parameters_Ref);

Set_Scheduling_Params: Format the type of data to allow generic use of it

From_Params: Scheduling parameters to make the conversion
To_Params: Scheduling parameters converted

References

Further information available:

PolyORB: https://libre.adacore.com/

MaRTE OS and RTEP: http://marte.unican.es/index.htm