# PolyORB High Integrity User's Guide

Ada 2005 Edition
Version 1.1w
Date: 17 May 2011

**Jérôme Hugues, Bechir Zalila**

# Table of Contents

# About This Guide

This document describes PolyORB High-Integrity Ada (PolyORB-HI-Ada), a reduced version of the PolyORB schizophrenic middleware (http://libre.adacore/com/polyorb) for High-Integrity systems.

There are two versions of PolyORB High Integrity. The first, written in Ada is called PolyORB-HI-Ada, and the other, written in C, is called PolyORB-HI-C. The following manual focuses on PolyORB-HI-Ada.

## What This Guide Contains

This guide contains the following chapters:

- Chapter 1 [Introduction to PolyORB-HI-Ada], page 3 provides a brief description of middleware and PolyORB-HI-Ada's architecture.
- Chapter 2 [Installation], page 5 details how to configure and install PolyORB-HI-Ada on your system.
- Chapter 3 [Building a system], page 7 details how to build a distributed system from its AADL description.
- Chapter 4 [Adding new transport mechanism], page 9 details how to add new transport mechanisms seamlessly to PolyORB-HI-Ada using AADLv2 features.
- Appendix A [Supported features], page 11 details the features that are available in PolyORB-HI-Ada,as well as the Ada restrictions for HI systems it follows.
- Appendix B [AADL to Ada transformations], page 15 details the mapping rules to map an AADL model onto a High-Integrity Distributed System.
- Appendix C [PolyORB-HI-Ada API], page 33 provides an overview of PolyORB-HI-Ada API.
- Appendix D [FAQ], page 35 provides answers to Frequently Asked Questions.
- Appendix E [References], page 37 provides a list of useful references to complete this documentation.
- Appendix F [GNU Free Documentation License], page 39 contains the text of the license under which this document is being distributed.

## Conventions

Following are examples of the typographical and graphic conventions used in this guide:

- `Functions`, `utility program names`, `standard names`, and `classes`.
- '`Option flags`'
- '`File Names`', '`button names`', and '`field names`'.
- *Variables*.
- *Emphasis*.
- [optional information or parameters]
- Examples are described by text

```
and then shown this way.
```

Commands that are entered by the user are preceded in this manual by the characters "`$ `" (dollar sign followed by space). If your system uses this sequence as a prompt, then the commands will appear exactly as you see them in the manual. If your system uses some other prompt, then the command will appear with the `$` replaced by whatever prompt character you are using.

Full file names are shown with the "`/`" character as the directory separator; e.g., '`parent-dir/subdir/myfile.adb`'. If you are working on a Windows platform, please note that the "\" character should be used instead.

# 1 Introduction to PolyORB-HI-Ada

PolyORB-HI-Ada is a middleware for High-Integrity Systems, it inherits most concepts of the schizophrenic middleware *PolyORB* while being based on a complete new source code base, compatible with the Ravenscar profile and the restrictions for High-Integrity systems.

In order to ease the construction of Distributed High-Integrity Systems, PolyORB-HI-Ada relies on the AADL language and the Ocarina toolsuite (*[VZ06]*) to allocate statically all required resources and generate stubs, skeletons, marshallers and concurrent structures. Hence, PolyORB-HI-Ada acts as an AADL runtime.

Ocarina supports both AADLv1 *[SAE04]* and AADLv2 *[SAE09a]* as input models, and the data modeling *[SAE09b]* and programming languages annexes *[SA09c]*.

This manual describes the elements specific to PolyORB-HI-Ada.

# 2 Installation

## 2.1 Supported Platforms

PolyORB-HI-Ada has been compiled and successfully tested on

- native platforms:
  - Linux
  - Mac OS X
  - Solaris
  - FreeBSD
  - Windows
- Real-Time Operating Systems (RTOS) for Embedded systems
  - MaRTE OS, by Universidad de Cantabria,
  - RTEMS, by OAR Corp.
- bare-board platforms
  - ERC32, using GNAT Pro for ERC32
  - LEON, using GNAT Pro for LEON 6.2.1 or above, or GNAT for LEON 2.1.0 by Universidad Politecnic de Madrid,

*Note: PolyORB-HI-Ada should compile and run on every target for which GNAT is available*

## 2.2 Tree structure

PolyORB-HI-Ada source distribution has the following tree structure:

- '`doc/`': this documentation,
- '`examples/`': set of examples to test PolyORB-HI-Ada
- '`share/`': common files
- '`src/`': core of PolyORB-HI-Ada
- '`tools/`': helper executables to run multi-nodes applications
- '`ChangeLog`': release information,
- '`COPYING`': GPLv2 license document,
- '`README`': short description of the distribution.
- '`INSTALL`': quick installation instruction

When installed with Ocarina, in '`$OCARINA_PATH`' directory

- docutmentation is in '`$OCARINA_PATH/share/doc/ocarina`';
- examples are in '`$OCARINA_PATH/examples/ocarina/polyorb-hi-ada/`': set of examples to test PolyORB-HI-Ada
- runtime files are in '`$OCARINA_PATH/include/ocarina/runtime/polyorb-hi-ada/`'.

## 2.3 Build requirements

To be compiled, PolyORB-HI-Ada requires the following tools:

- For native platforms: a recent version of the GNAT compiler, e.g. GNAT Pro 6.2.1 or above, GNAT {GAP, GPL} 2008 or later or GNAT GCC 4.3.0 or later
- For LEON targets: the GNAT for LEON 2.1 or above; or GNAT Pro 6.2.1 or above
- For ECR32: the GNAT for ERC32 2006 or later.

PolyORB-HI-Ada also relies on AADL-to-Ada code generation features provided by Ocarina. Therefore, it is important to select a version of Ocarina that is compatible with this version of PolyORB-HI-Ada. Whenever possible, pick a unified archive that contains both tools.

## 2.4 Installation instructions

To install PolyORB-HI-Ada, please observe the following steps:

- Install GNAT and Ocarina as specified by their respective documentations and make sure their 'bin/' installation directories are located at the top of your PATH environment variable.
- Issue `./configure`. The `configure` script can take several options: issue `./configure --help` to have a quick overview of them. For examples. `./configure --enable-debug` will configure the middleware to be built with all debug options.
- Issue `make && make install`

## 2.5 Build instructions

PolyORB-HI-Ada must be installed correctly in order to be able to build examples.

To compile all examples, simply issue `make examples` from the main source directory. To clean the examples, issue `make clean-examples` from the main source directory.

The examples may be built with the debug information and with the GNAT compiler checks to make their debugging possible. In this case the footprint of the generated binaries is about almost 1MB per executable. This is the default behavior of the `make examples` command. If the user wants to make the examples without any debug information and any GNAT check, he should use the `make examples 'BUILD=Release'` command instead. The footprint of the generated executable will be reduced considerably.

Each example uses the GNAT build infrastructure and a makefile.

For each example, a makefile is provided with the following rules:

- `build-all`: generate code from the example and compile it;
- `clean`: clean all generated files;

# 3 Building a system

In this chapter, we discuss the construction of an application, using PolyORB-HI-Ada and an AADL model of the application.

## 3.1 Building examples

Each example provides a makefile that does the following steps:

1. parse the AADL model;
2. generate Ada code from the AADL model;
3. compile each node, enforcing coding restrictions detailed in the several `.adc` files.

PolyORB-HI-Ada comes with different examples and configurations, please refer to '`examples/README`' and subsequent documentation files for more details.

## 3.2 Building a new system

To build your own system, you have two choices: using a scenario file or the command line.

- To use a scenario file, please follow these instructions
  1. build a scenario file, a scenario file is an AADL file containing a system describing your applications (AADL files, code generator, needed Ocarina non-standard property sets, ...). Here is an example from the *Ping* example:

```
--  This is a scenario file describing the AADL
--  application ping

--  $Id$

system ping
properties
  Ocarina_Config::Timeout_Property     => 4000ms;
  Ocarina_Config::Referencial_Files    =>
    ("node_a", "node_a_leon.ref",
    "node_b", "node_b_leon.ref");
  Ocarina_Config::AADL_Files           =>
    ("ping-local.aadl", "software.aadl");
  Ocarina_Config::Generator            => polyorb_hi_ada;
  Ocarina_Config::Needed_Property_Sets =>
    (value (Ocarina_Config::Data_Model),
     value (Ocarina_Config::Deployment),
     value (Ocarina_Config::Cheddar_Properties));
  Ocarina_Config::AADL_Version         => AADLv2;
  Ocarina_Config::Root_System_Name     => "ping.leon_gnat";
end ping;

system implementation ping.Impl
end  ping.Impl;
```

  2. issue the command `ocarina -b -x <scenario-file>`
- To use command line, please follow these instructions
  1. issue the command `ocarina -g polyorb_hi_ada <list-of-aadl-files>`

For a list of supported flags, please refer to the *Ocarina User's Guide*.

# 4 Adding new transport mechanism

In this section, we detail how to add new transport mechanisms seamlessly to PolyORB-HI-Ada using AADLv2 features.

We base the following on the example found in the examples directory, named '`aadlv2/device_drivers`'.

We use AADLv2 features to add integrate device drivers to PolyORB-HI/Ada. To do so, you need to perform the following steps

1. First, you need to define a device and point to an abstract AADLv2 component that supports its full implementation using the `Implemented_As` property. This abstract component will define all internals of the device driver.

```
package TCP_IP_Protocol
public
  device implementation TCP_IP_Device.impl
  properties
    Implemented_As => classifier (TCP_IP_Protocol::Driver_TCP_IP_Protocol.impl);
    Initialize_Entrypoint => classifier (TCP_IP_Protocol::Initialize);
  end TCP_IP_Device.impl;
end TCP_IP_Protocol;
```

2. The abstract implementation shall define or reference required thread, subprograms.
   1. AADL entities shall define threads and subprograms to handle incoming requests, unmarshall them, and do the upcall to PolyORB-HI-Ada internals. See code provided for the AADL subprogram `TCP_IP.Send`.
   2. User provided code shall define a subprogram to marshall and send data. This subprogram shall be exported using Ada `pragma Export`. The name of the corresponding symbol shall be made as follows <*name_of_device*>_*send*.

      In the `device_drivers` example, this name is `tcp_ip_device.impl_send`.

      **Note:** it is the user responsibility to ensure that marshalling/unmarshalling is performed correctly.
   3. In addition, you may add an initialisation procedure for this driver using the `Initialize_Entrypoint`.

      This function takes as input parameter one naming table as defined in the deployment table. This naming table shall be used to
      - Open incoming communication channels
      - Open channels to remote nodes

      See Section B.2 [Distributed application nodes], page 16.

# Appendix A Supported features

PolyORB-HI-Ada is a middleware dedicated to High-Integrity systems. Therefore, it relies on a limited subset of AADL and Ada features, and allow the user to enforce a large set of restrictions.

Here is the list of restrictions allowed, and enforced by Ocarina and the Ada compiler.

## A.1 Ada constructs

PolyORB-HI-Ada strictly follows Ada restrictions for High-Integrity systems. They are enforced through different compilation files:

- The files '`gnat.adc`' lists common restrictions applied to all compiled files

```
--  Ada restrictions to be supported by the PolyORB HI, common restrictions

--  For each restriction, we list its definition place in the Ada 2005
--  reference manual.

pragma Restrictions (No_Allocators);                -- H.4 (7)
pragma Restrictions (No_Floating_Point);            -- H.4 (14)
--  pragma Restrictions (No_Access_Subprograms);        -- H.4 (17)
pragma Restrictions (No_Unchecked_Access);          -- H.4 (18)
pragma Restrictions (No_Dispatch);                  -- H.4 (19)
pragma Restrictions (No_IO);                        -- H.4 (20)
pragma Restrictions (No_Recursion);                 -- H.4 (22)

--  pragma Partition_Elaboration_Policy (Sequential); -- H.6 (3)
--  This policy eases task initial synchronization and the bootstrap
--  of the VM, see Ada 2005 Rationale for more details). (Not yet
--  supported by GNAT)

--  pragma Restriction_Warnings (No_Implementation_Attributes);   -- 13.12 (2)
--  Style-check to enforce compliance with pure Ada, but also to avoid
--  the 'Unrestricted_Access attribute, which is GNAT specific.

--  pragma Restrictions (No_Obsolescent_Features);       -- 13.12 (4)
-- Deactivated as of 20090702: GPL 2009 deprecated
-- GNAT.Sockets.Initialize, but it might still be required for older
-- compilers supported.

pragma Restrictions (No_Unchecked_Deallocation);     -- J.13 (4)
--  Sibling of the No_Allocators restriction.
--  XXX to be replaced with a No_Dependence restriction

--  GNAT Specific restrictions

--  pragma Restrictions (No_Streams);                    -- GNAT specific

--  pragma Restrictions (No_Direct_Boolean_Operators);   -- GNAT specific
--  For the use of "and then" and "or else" in boolean operations

--  pragma Restrictions (No_Enumeration_Maps);           -- GNAT specific
-- Forbids <type>'Image for enumeration types. Cannot be supported
-- because some examples use it
```

```
--  pragma Restrictions (No_Implicit_Conditionals);     -- GNAT specific
--  pragma Restrictions (No_Implicit_Loops);            -- GNAT specific
--  pragma Restrictions (No_Enumeration_Maps);          -- GNAT specific
```

- '`native.adc`' lists restrictions applied to compiled files for native targets

```
--  Ada restrictions to be supported by PolyORB HI, for native targets

--  pragma Profile_Warnings (Ravenscar);              -- D.13.1
```

- '`hi-e.adc`' lists restrictions applied to compiled files for High-Integrity targets such as bare board LEON and ERC32 compilers.

```
--  Ada restrictions to be supported by the PolyORB HI, for HI targets

pragma Profile (Ravenscar);                            -- D.13.1

pragma Restrictions (No_Dependence => Ada.Finalization); -- 13.12.1 (12)
--  No controlled types, cannot be enforced for native compilers
--  because of the s-tpoben file.

--  pragma Restrictions (No_Dependence => Ada.Streams);      -- 13.12.1 (12)
--  No implicit dependency on Ada.Tags and other object oriented
--  compiler units. Cannot be enforced for native platforms because of
--  the g-socket file

--  pragma Restrictions (No_Exception_Handlers);             -- GNAT specific
--  Implied by GNAT Pro 5.03a1 for High-Integrity, for the Ravenscar
--  run-time,  cannot be enforced for native compilers because of the
--  s-tpoben file.

--  pragma Restrictions (No_Secondary_Stack);                -- GNAT specific
--  No unconstrained objects, including arrays, but also forbids
--  string concatenation. (GNAT specific)
```

- The file '`polyorb-hi.gpr`' lists the compilation options applied to all units, the different supported targets (``native'', ``leon'' and ``erc 32'').

The `check` rule of the makefile allows one to check for the remaining restrictions, using the '`gnatcheck`' tool.

## A.2 AADL features

PolyORB-HI-Ada acts as an AADLv1 or AADLv2 runtime. AADL is a complete description language. Some features cannot be implemented or supported by restricted HI runtimes.

This section lists AADL features supported by PolyORB-HI-Ada:

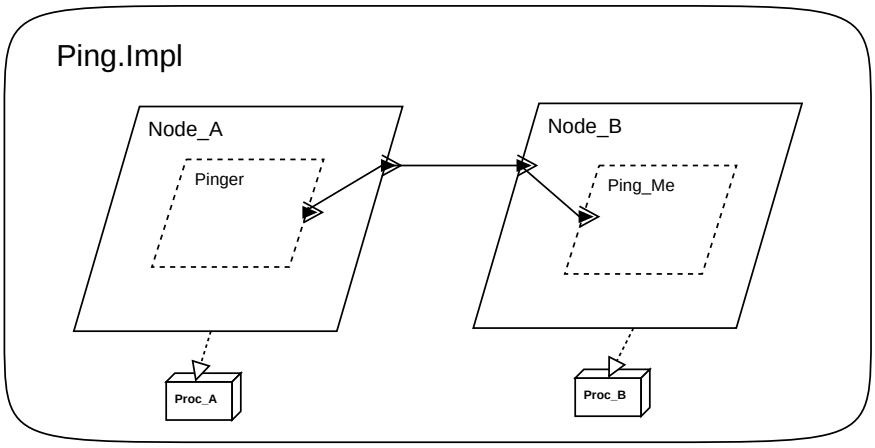
- asynchronous, oneway calls;
- data component types of statically bounded size;
- all compile-time and run-time restrictions enforced as part of the compilation process;
- User code must conform with the data modeling annex (*[SAE09b]*), and the programing language annex (*[SA09c]*).

Please refer to these documents for more details.

- PolyORB-HI-Ada can use different transport infrastructures:
  - on native platform, distribution can be tested using the native socket library provided by `GNAT.Sockets`.
  - user-provided transport layer can be used, provided they follow guidelines discussed in section See Chapter 4 [Adding new transport mechanism], page 9.

# Appendix B AADL to Ada transformations

In this chapter, we introduce some of the code patterns that are generated for each AADL entity. These mapping rules will be illustrated using the following example of a distributed application:




The figure above shows the architecture of the *Ping* example: a client, which is a process containing one single *periodic* thread, sends a message to the server which is a process containing one *aperiodic cyclic* thread that handles incoming ping messages from the client. Each node of the *Ping* application runs on a different machine.

This graphical example has been adapted as textual AADLv1 and AADLv2 models, and is available in the '`examples/aadlv1/ping`' or '`examples/aadlv2/ping`'.

## B.1 Whole distributed application

A distributed application is an application which is composed by interacting nodes. In this section, we give the AADL entities used to model a distributed application. Then, we give the rules applied to map these AADL entities onto instances of PolyORB-HI-Ada components, expressed as Ada code.

In the following, we detail only the rules that are directly related to the distributed application as a whole system. The rules that are specific to the components of the distributed application are explained later.

### B.1.1 AADL entities

To model a distributed application in AADL we use the `system` component. The system implementation shown on the following example models such system.

```
package PING_Package
public
  system implementation PING.Native
  subcomponents
    Node_A : process A.Impl;
    Node_B : process B.Impl
```

```
      {Deployment::port_number => 12002;};
    CPU : processor the_processor
      {Priority_Range => 0 .. 255;};
    the_bus : bus Ethernet_Bus;
  connections
    bus access the_bus -> CPU.ETH;
    port Node_A.Out_Port -> Node_B.In_Port
      {Actual_Connection_Binding => (reference ( the_bus ));};
  properties
    actual_processor_binding => (reference ( CPU ))
      applies to Node_A;
    actual_processor_binding => (reference ( CPU ))
      applies to Node_B;
  end PING.Native;
end PING_Package;
```

For each node (process) of the distributed application, we instantiate a subcomponent in the system implementation.

We use the `properties` section of the AADL `system` (see Section B.3 [Nodes], page 28 for more details) to map the logical nodes (processes) onto the physical nodes (processors). The `connections` section of the system implementation models the connections between the different nodes of the application.

### B.1.2 Ada mapping rules

A distributed application is mapped into a hierarchy of directories:

- the root directory of the distributed application has the same name as the system implementation that models the application, in lower case, all dots being converted into underscores. This directory is the root of the directory hierarchy of the generated Ada distributed application.
- for each node of the distributed application, a child directory having the same name as the corresponding process subcomponent (in lower case) is created inside the root directory. This child directory will contain all the code generated for the particular node it was created for (see Section B.2 [Distributed application nodes], page 16 for more details).

## B.2 Distributed application nodes (processes)

In this section, we give the AADL entities used to model a node of distributed application. Then, we give the rules applied to map Ada code from these AADL entities. Only rules that are related directly to a node as a whole subsystem are listed here. The rules that are specific to the sub-components of a node are explained in the sections that deal with these respective sub-components.

### B.2.1 AADL entities

To model a distributed application node in AADL we use the `process` component. The process implementation shown in the listing below shows such system. For each node of the distributed application, we add a process instantiation as subcomponent in the system implementation that models the distributed application.

```
package PING_Package
public
  process implementation A.Impl
  subcomponents
    Pinger : thread Software::P.Impl;
  connections
    port Pinger.Data_Source -> Out_Port;
  end A.Impl;
end PING_Package;
```

For each thread that belongs to a node of the distributed application, we instantiate a subcomponent in the process implementation. For each connection between a node and another, a `port` feature has to be added to both nodes with the direction `out` for the source and `in` for the destination (see Section B.5 [Connections], page 31 for more details on connections mapping).

### B.2.2 Ada mapping rules

All the Ada entities mapped from a distributed application node, are created in a child directory of the directory mapped from the distributed application. This directory has the same name as the process *subcomponent* instance relative to the handled node in the system implementation that model the distributed application, in lower case.

For example, all the entities relative to `Node_A`, an instance of the process `A` of the `Ping` example are generated in the directory `ping_impl/node_a`.

The following paragraphs list the Ada compilation units that are created for each node of the distributed application.

#### B.2.2.1 Data marshallers

The data marshallers are generated in a package called `PolyORB_HI_Generated.Marshallers`. This package provides routines to convert the data types used in the node and other useful generated type to a representation suitable for transmission over network.

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI_Generated.Activity;
with PolyORB_HI.Messages;
with PolyORB_HI_Generated.Types;

package PolyORB_HI_Generated.Marshallers is

  --  Marshallers for interface type of thread p.impl

  procedure Marshall
   (Data : PolyORB_HI_Generated.Activity.Software_P_Impl_Interface;
    Message : in out PolyORB_HI.Messages.Message_Type);

  procedure Unmarshall
   (Port : PolyORB_HI_Generated.Activity.Software_P_Impl_Port_Type;
```

```
      Data : out PolyORB_HI_Generated.Activity.Software_P_Impl_Interface;
      Message : in out PolyORB_HI.Messages.Message_Type);

   --  Marshallers for DATA type simple_type

   procedure Marshall
    (Data : PolyORB_HI_Generated.Types.Simple_Type;
     Message : in out PolyORB_HI.Messages.Message_Type);

   procedure Unmarshall
    (Data : out PolyORB_HI_Generated.Types.Simple_Type;
     Message : in out PolyORB_HI.Messages.Message_Type);

end PolyORB_HI_Generated.Marshallers;
```

The body of `PolyORB_HI_Generated.Marshallers` instantiates of the generic package `PolyORB_HI.Marshallers_G` for the corresponding types. The generic package `PolyORB_HI.Marshallers_G` is part of PolyORB-HI-Ada components. The example above shows the specification of the generated `PolyORB_HI_Generated.Marshallers` mapped from the `Node_A` node of the `Ping`.

### B.2.2.2 Node activity

We denote "activity" the set of the actions performed by one particular node. This denotes particularly the periodic, sporadic and hybrid threads that belongs to this node.

The code related to the node activity is generated in an Ada package called `PolyORB_HI_Generated.Activity` as shown in the following example:

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI_Generated.Types;
with PolyORB_HI.Errors;
with PolyORB_HI_Generated.Deployment;
with Ada.Real_Time;
with System;
with PolyORB_HI.Periodic_Task;

package PolyORB_HI_Generated.Activity is

   --  BEGIN: Entities used by all instances of component P.Impl

   type Software_P_Impl_Port_Type is
    (Data_Source);

   type Software_P_Impl_Interface
    (Port : Software_P_Impl_Port_Type := Software_P_Impl_Port_Type'First)
   is
      record
         case Port is
            when Data_Source =>
               Data_Source_DATA : PolyORB_HI_Generated.Types.Simple_Type;
```

```
      pragma Warnings (Off);
      when others =>
        null;
      pragma Warnings (On);
    end case;
  end record;

function Send_Output
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type)
 return PolyORB_HI.Errors.Error_Kind;

procedure Put_Value
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Thread_Interface : Software_P_Impl_Interface);

procedure Receive_Input
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type);

function Get_Value
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type)
 return Software_P_Impl_Interface;

function Get_Sender
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type)
 return PolyORB_HI_Generated.Deployment.Entity_Type;

function Get_Count
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type)
 return Standard.Integer;

function Get_Time_Stamp
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type)
 return Ada.Real_Time.Time;

procedure Next_Value
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : Software_P_Impl_Port_Type);

procedure Store_Received_Message
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Thread_Interface : Software_P_Impl_Interface;
  From : PolyORB_HI_Generated.Deployment.Entity_Type;
  Time_Stamp : Ada.Real_Time.Time :=
    Ada.Real_Time.Clock);

procedure Wait_For_Incoming_Events
 (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
  Port : out Software_P_Impl_Port_Type);

--  END: Entities used by all instances of component P.Impl

--  Periodic task : Pinger
```

```
   function PING_Package_Pinger_Job return PolyORB_HI.Errors.Error_Kind;

   procedure PING_Package_Pinger_Init;

   procedure PING_Package_Pinger_Recover;

   package PING_Package_Pinger_Task is
    new PolyORB_HI.Periodic_Task
      (Entity => PolyORB_HI_Generated.Deployment.Node_A_Pinger_K,
       Task_Period => Ada.Real_Time.Milliseconds
        (1000),
       Task_Deadline => Ada.Real_Time.Milliseconds
        (1000),
       Task_Priority => (System.Priority'First
         + (((2
           - 0)
           * (System.Priority'Last
             - System.Priority'First))
           / 255)),
       Task_Stack_Size => 100000,
       Job => PING_Package_Pinger_Job,
       Initialize_Entrypoint => PING_Package_Pinger_Init,
       Recover_Entrypoint => PING_Package_Pinger_Recover);

end PolyORB_HI_Generated.Activity;
```

All the naming rules explained in Section B.1 [Whole distributed application], page 15 are also applied to map the package name. The `PolyORB_HI_Generated.Activity` package contains all the routines mapped from the threads that belong to the handled node. This package contains also the instances of shared objects used in this node. See Section B.4 [Threads], page 29.

### B.2.2.3 Data types

All the data types mapped from AADL data components and used by a particular node of a distributed application are gathered in a separate Ada package called `PolyORB_HI_Generated.Types`. The name of this package is fixed for the sake of simplicity because it may be used by the user implementation code. The user does not have to change his implementation code if he renamed its distributed application.

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with User_Types;

package PolyORB_HI_Generated.Types is

  type Simple_Type is
   new Standard.Integer;

  --  simple_type'Object_Size ~= 32 bits
```

```
   Simple_Type_Default_Value : Simple_Type :=
    0;

   type Opaque_Type is
    new User_Types.The_Opaque_Type;

   --  opaque_type'Object_Size ~= 0 bits

   Opaque_Type_Default_Value : Opaque_Type;

end PolyORB_HI_Generated.Types;
```

### B.2.2.4 Subprograms

The mapping of all AADL subprogram components used by a particular node is generated in a separate Ada package. The name of this package is `PolyORB_HI_Generated.Subprograms` as shown in the following example:

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI_Generated.Types;

package PolyORB_HI_Generated.Subprograms is

   procedure Software_P_Wrapper_Impl
    (Data_Source : out PolyORB_HI_Generated.Types.Simple_Type);

   procedure Software_Do_Ping_Spg_Impl
    (Data_Source : out PolyORB_HI_Generated.Types.Opaque_Type);

   procedure Software_Do_Convert_Spg_Impl
    (Data_Sink : PolyORB_HI_Generated.Types.Opaque_Type;
     Data_Source : out PolyORB_HI_Generated.Types.Simple_Type);

end PolyORB_HI_Generated.Subprograms;
```

### B.2.2.5 Deployment information

The deployment information is the information each node has on the other nodes of the distributed applications. This information is used, in conjunction with the naming table (see the next paragraph) to allow a node to send a request to another node or to receive a request from another node. The deployment information is generated for each node in the package `PolyORB_HI_Generated.Deployment`.

The `PolyORB_HI_Generated.Deployment` package contains several Ada enumeration types:

- `Node_Type`. For each node in the application accessible from the current node, we create an enumeration literal whose name is mapped from the node "instance" declared in the

system implementation to which we concatenate the string "_K". All the naming rules listed in Section B.1 [Whole distributed application], page 15 have to be respected.

In addition, the constant `My_Node` of type `Node_Type` is defined and store the value of the current node.

- `Entity_Type`. For each thread in the application that belongs to the current node or that is connected (via its owner process) to one of the threads of the current node, we declare an enumeration literal. The defining identifier of the enumerator is mapped from the process subcomponent name and the thread subcomponent name as follows: `<Node_Name>_<Thread_Name>_K`. All the naming rules listed in Section B.1 [Whole distributed application], page 15 must be respected.
- `Port_Type`. For each thread port in the distributed applications accessible from the current node, we declare an enumeration literal. Only threads that belongs to the `Entity_Type` enumeration are mapped.

For each table, Ada enumeration clauses are generated to guarantee that each enumeration literal has a unique value across partitions.

The table called `Entity_Table` is generated to map threads to nodes, and `Port_Table` to maps port to threads.

Finally, we generate a constant (`Max_Payload_Size`) that indicates the maximal size (in bits) of a message handled in the current node. This size is computed automatically from the set of data types handled (sent or received) in the node. It will be used to configure the `PolyORB_HI.Messages` package.

The following example shows the `PolyORB_HI_Generated.Deployment` package relative to the node `Node_A` of the `Ping` example:

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");

package PolyORB_HI_Generated.Deployment is

  pragma Preelaborate;

  --  For each node in the distributed application add an enumerator

  type Node_Type is
   (Node_A_K,
    Node_B_K);

  --  Representation clause to have consistent positions for enumerators

  for Node_Type use
   (Node_A_K =>
     1,
    Node_B_K =>
     2);
```

```
--  Size of Node_Type fixed to 8 bits

for Node_Type'Size use 8;

Max_Node_Image_Size : constant Standard.Integer :=
 8;

--  Maximal Node_Image size for this node

--  Node Image

subtype UT_Deployment_Node_A_Node_Type_Range is
 Node_Type range Node_Type'First .. Node_Type'Last;

subtype UT_Deployment_Node_A_1_Max_Node_Image_Size is
 Integer range 1 .. Max_Node_Image_Size;

subtype UT_Deployment_Node_A_Node_Image_Component is
 Standard.String
   (UT_Deployment_Node_A_1_Max_Node_Image_Size);

type UT_Deployment_Node_A_Node_Image_Array is
 array (UT_Deployment_Node_A_Node_Type_Range)
   of UT_Deployment_Node_A_Node_Image_Component;

Node_Image : constant UT_Deployment_Node_A_Node_Image_Array :=
 UT_Deployment_Node_A_Node_Image_Array'
   (Node_A_K =>
     "Node_A_K",
    Node_B_K =>
     "Node_B_K");

My_Node : constant Node_Type :=
 Node_A_K;

--  For each thread in the distributed application nodes, add an enumerator

type Entity_Type is
 (Node_A_Pinger_K,
  Node_B_Ping_Me_K);

--  Representation clause to have consistent positions for enumerators

for Entity_Type use
 (Node_A_Pinger_K =>
   1,
  Node_B_Ping_Me_K =>
   2);

--  Size of Entity_Type fixed to 8 bits

for Entity_Type'Size use 8;

--  Entity Table

subtype UT_Deployment_Node_A_Entity_Type_Range is
 Entity_Type range Entity_Type'First .. Entity_Type'Last;
```

```
type UT_Deployment_Node_A_Entity_Table_Array is
 array (UT_Deployment_Node_A_Entity_Type_Range)
   of Node_Type;

Entity_Table : constant UT_Deployment_Node_A_Entity_Table_Array :=
 UT_Deployment_Node_A_Entity_Table_Array'
   (Node_A_Pinger_K =>
     Node_A_K,
    Node_B_Ping_Me_K =>
     Node_B_K);

Max_Entity_Image_Size : constant Standard.Integer :=
 16;

--  Maximal Entity_Image size for this node

--  Entity Image

subtype UT_Deployment_Node_A_1_Max_Entity_Image_Size is
 Integer range 1 .. Max_Entity_Image_Size;

subtype UT_Deployment_Node_A_Entity_Image_Component is
 Standard.String
   (UT_Deployment_Node_A_1_Max_Entity_Image_Size);

type UT_Deployment_Node_A_Entity_Image_Array is
 array (UT_Deployment_Node_A_Entity_Type_Range)
   of UT_Deployment_Node_A_Entity_Image_Component;

Entity_Image : constant UT_Deployment_Node_A_Entity_Image_Array :=
 UT_Deployment_Node_A_Entity_Image_Array'
   (Node_A_Pinger_K =>
     "Node_A_Pinger_K ",
    Node_B_Ping_Me_K =>
     "Node_B_Ping_Me_K");

--  For each thread port in the distributed application nodes, add an
--  enumerator

type Port_Type is
 (Node_A_Pinger_Data_Source_K,
  Node_B_Ping_Me_Data_Sink_K);

--  Representation clause to have consistent positions for enumerators

for Port_Type use
 (Node_A_Pinger_Data_Source_K =>
   1,
  Node_B_Ping_Me_Data_Sink_K =>
   2);

--  Size of Port_Type fixed to 16 bits

for Port_Type'Size use 16;

--  Port Table

subtype UT_Deployment_Node_A_Port_Type_Range is
```

```
   Port_Type range Port_Type'First .. Port_Type'Last;

  type UT_Deployment_Node_A_Port_Table_Array is
   array (UT_Deployment_Node_A_Port_Type_Range)
     of Entity_Type;

  Port_Table : constant UT_Deployment_Node_A_Port_Table_Array :=
   UT_Deployment_Node_A_Port_Table_Array'
     (Node_A_Pinger_Data_Source_K =>
       Node_A_Pinger_K,
      Node_B_Ping_Me_Data_Sink_K =>
       Node_B_Ping_Me_K);

  Max_Port_Image_Size : constant Standard.Integer :=
   27;

  --  Maximal Port_Image size for this node

  subtype Port_Sized_String is
   Standard.String
     (1 .. PolyORB_HI_Generated.Deployment.Max_Port_Image_Size);

  --  Port Image

  type UT_Deployment_Node_A_Port_Image_Array is
   array (UT_Deployment_Node_A_Port_Type_Range)
     of Port_Sized_String;

  Port_Image : constant UT_Deployment_Node_A_Port_Image_Array :=
   UT_Deployment_Node_A_Port_Image_Array'
     (Node_A_Pinger_Data_Source_K =>
       "Node_A_Pinger_Data_Source_K",
      Node_B_Ping_Me_Data_Sink_K =>
       "Node_B_Ping_Me_Data_Sink_K ");

  --  Maximal message payload size for this node (in bits)

  Max_Payload_Size : constant Standard.Integer :=
   112;

  --  Biggest type: simple_type

end PolyORB_HI_Generated.Deployment;
```

### B.2.2.6 Naming information

The naming information for a particular node allows this node to send requests to another node in the distributed application and to receive a request from another node. It contains for each node, the information necessary to establish a connection with a remote node. These information are deduced statically from the AADL model.

The naming information is generated in a package called `PolyORB_HI_Generated.Naming`.

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
```

```
--  changes will be lost when you re-run Ocarina      --
-------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI.Utils;
with PolyORB_HI_Generated.Deployment;

package PolyORB_HI_Generated.Naming is

  --  Naming Table for bus the_bus

  Naming_Table : constant PolyORB_HI.Utils.Naming_Table_Type :=
   (PolyORB_HI_Generated.Deployment.Node_A_K =>
     (PolyORB_HI.Utils.To_Hi_String
       ("127.0.0.1"),
      0),
    PolyORB_HI_Generated.Deployment.Node_B_K =>
     (PolyORB_HI.Utils.To_Hi_String
       ("127.0.0.1"),
      12002));

end PolyORB_HI_Generated.Naming;
```

As shown in the example above, for the node `Node_A` of the `Ping` example, the `PolyORB_HI_Generated.Naming` package contains:

- A static table called `Naming_Table` indexed by means of the `Node_Type` declared in the `PolyORB_HI_Generated.Deployment` package. This table contains, for each node of the distributed application, all the information necessary to establish a connection with this node.

### B.2.2.7 Transport Layer

The transport high-level layer is generated automatically for each node of the application in a package called `PolyORB_HI_Generated.Transport`. This package contains the routines to send and receive messages between the threads of the application, either locally or remotely, depending on the source and the destination of the message. In case of remote communication, the transport high-level layer will invoke the routines of the transport low level layer depending on the kind of the connection between the source and the destination.

The example below shows the `PolyORB_HI_Generated.Transport` generated for the node `Node_A` of the `Ping` example.

```
-------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
-------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI_Generated.Deployment;
with PolyORB_HI.Streams;
with PolyORB_HI.Messages;
with PolyORB_HI.Errors;

package PolyORB_HI_Generated.Transport is
```

```
   procedure Deliver
    (Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
     Message : PolyORB_HI.Streams.Stream_Element_Array);

   function Send
    (From : PolyORB_HI_Generated.Deployment.Entity_Type;
     Entity : PolyORB_HI_Generated.Deployment.Entity_Type;
     Message : PolyORB_HI.Messages.Message_Type)
    return PolyORB_HI.Errors.Error_Kind;

end PolyORB_HI_Generated.Transport;
```

### B.2.2.8 Main subprogram

The main subprogram is an Ada subprogram that does all the necessary initialization before the effective run of the node. The defining identifier of the main subprogram is mapped from the process subcomponent "instance" declared in the system implementation. All the naming rules listed in Section B.1 [Whole distributed application], page 15 have to be respected.

The body of the main subprogram contains the initialization of the communication transport layer, then we no longer need the main task, so we suspend it for ever. The following example shows the main subprogram generated for the node `Node_A` of the `Ping` example.

```
------------------------------------------------------
--  This file was automatically generated by Ocarina  --
--  Do NOT hand-modify this file, as your             --
--  changes will be lost when you re-run Ocarina      --
------------------------------------------------------
pragma Style_Checks
 ("NM32766");
with PolyORB_HI_Generated.Activity;
pragma Warnings (Off, PolyORB_HI_Generated.Activity);
pragma Elaborate_All (PolyORB_HI_Generated.Activity);
with System;
with PolyORB_HI.Transport_Low_Level;
with PolyORB_HI.Suspenders;

------------
-- Node_A --
------------

procedure Node_A is
  pragma Priority
   (System.Priority'Last);
begin
  --  Initialize default communication subsystem
  PolyORB_HI.Transport_Low_Level.Initialize;
  --  Unblock all user tasks
  PolyORB_HI.Suspenders.Unblock_All_Tasks;
  --  Suspend forever instead of putting an endless loop. This saves the CPU
  --  resources.
  PolyORB_HI.Suspenders.Suspend_Forever;
end Node_A;
```

In order to elaborate correctly the tasks, a special elaboration clause has to be added for the `PolyORB_HI_Generated.Activity` package in the main subprogram when the node contains periodic threads (see the example above).

## B.3 Nodes

A node is the set formed by a processor and an operating system (or real-time kernel).

In this section we present the AADL entities used to model a node. Then, we give the mapping rules used to generate Ada code expressing that a node runs on a particular node.

### B.3.1 AADL entities

To model both the processor and the OS, we use the `processor` AADL component. The characteristics of the processor are defined using the AADL properties. For example, if our distributed application uses an IP based network to make its node communicate, then each node must have an IP address. Each node must also precise its platform (native, LEON...). The listing following example shows how to express this using a custom property set.

```
package PING_Package
public
  processor the_processor
  features
    ETH : requires bus access Ethernet_Bus;
  properties
    Deployment::location => "127.0.0.1";
    Deployment::Execution_Platform => Native;
  end the_processor;
end PING_Package;
```

To map an application node (processor) to a particular node, we use the `Actual_Processor_Binding` property. The following example shows how the node `Node_A` is mapped to the processor `Proc_A` in the `Ping` example.

```
package PING_Package
public
  system implementation PING.Native
  subcomponents
    Node_A : process A.Impl;
    Node_B : process B.Impl
      {Deployment::port_number => 12002;};
    CPU : processor the_processor
      {Priority_Range => 0 .. 255;};
    the_bus : bus Ethernet_Bus;
  connections
    bus access the_bus -> CPU.ETH;
    port Node_A.Out_Port -> Node_B.In_Port
      {Actual_Connection_Binding => (reference ( the_bus ));};
  properties
    actual_processor_binding => (reference ( CPU ))
      applies to Node_A;
    actual_processor_binding => (reference ( CPU ))
      applies to Node_B;
  end PING.Native;
```

```
end PING_Package;
```

### B.3.2 Ada mapping rules

The Ada generated code concerning the code generation to model node mapping is located in the `PolyORB_HI_Generated.Naming` package. More precisely, the `Naming_Table` contains, for each node, the information related to its node. These information are dependant on the transport mechanism used in the distributed application.

## B.4 Threads

The threads are the active part of the distributed application. A node must contain at least one thread and may contain more than one thread. In this section, we give the AADL entities used to model threads. Then, we give the mapping rule to generate Ada code corresponding to the periodic and aperiodic threads.

The rules are listed relatively to the packages generated for the nodes and for the distributed application (see Section B.2 [Distributed application nodes], page 16 and Section B.1 [Whole distributed application], page 15). Only rules that are related directly to a thread as a whole subsystem are listed here.

### B.4.1 AADL entities

The `thread` AADL components are used to model threads in the distributed application. The `features` section of the thread component declaration describes the thread's interface (the ports that may be connected to the ports of other threads). The `properties` section of the thread implementation lists the properties of the thread such as its priority, its nature (periodic, sporadic or hybrid) and many other properties are expressed using AADL properties. The `calls` section of the thread implementation contains the sequences of subprograms the thread may call during its job. If the thread job consist of calling more than one subprogram, it is **mandatory** to encapsulate these calls inside a single subprogram which will consist the thread job. The thread job may also be specified by means of the standard AADL property `Compute_Entrypoint` applied to the thread or else to all its "IN EVENT [DATA] ports".The `connections` section of a thread implementation connects the parameters of the subprograms called by the thread to the ports of the thread or to the parameters of other called subprograms in the same thread.

```
package Software
public
  thread implementation P.Impl
  calls
    Mycall :
    {P_Spg : subprogram P_Wrapper.Impl;}
;
  connections
    parameter P_Spg.Data_Source -> Data_Source;
  properties
    Initialize_Entrypoint_Source_Text => "Msgs.Welcome_Pinger";
    Recover_Entrypoint_Source_Text => "Msgs.Recover";
    Dispatch_Protocol => Periodic;
    Period => 1000 Ms;
    Compute_Execution_time => 0 ms .. 3 ms;
```

```
    Deadline => 1000 ms;
    Priority => 2;
  end P.Impl;
end Software;
```

The listing above shows the thread `P` which belongs to the process `A` in the `Ping` example. We can see that `P` is a periodic thread with a period of 1000ms, that this thread has a unique `out event data port` and that at each period, the thread performs a call to the `Do_Ping_Spg` subprogram whose `out parameter` is connected to the thread port.

### B.4.2 Ada mapping rules for AADL threads

Periodic threads are *cyclic* threads that are triggered by and only by a periodic time event. between two time events the periodic threads do a non blocking job and then they sleep waiting for the next time event.

Sporadic threads are *cyclic* threads that are triggered by a sporadic event. The minimum inter-arrival time between two sporadic event is called the period of the sporadic thread.

Hybrid threads are *cyclic* threads that have a periodic behavior triggered by a periodic time event. They have also a sporadic behavior that is triggered by the reception of an external event between two occurrences of their period.

In the following, "threads" will denote periodic, sporadic and hybrid threads unless the contrary is explicitly precised.

#### B.4.2.1 Node activity

The majority of the code generated for threads is put in the `PolyORB_HI_Generated.Activity` package generated for the application node containing the handled thread. The generated entities are:

- An enumeration type listing all the ports of the thread. The name of the type is mapped from the thread instance name suffixed with "`_Port_Type`". The name of each enumerator is mapped form the corresponding port name. All the naming rules listed in Section B.1 [Whole distributed application], page 15 hold.

  For "hybrid" AADL threads, an extra in event port is added to the enumeration. This port is called `"Period_Event_"`. It will receive the period occurrences from the hybrid tasks driver.

- A discriminated record type (with the port type as discriminant) to describe the thread interface. the name of each field (in case of a data port) is mapped from the port name to which we add "`_DATA`" and the type is the Ada type mapped from the data component corresponding to the port. All the naming rules listed in Section B.1 [Whole distributed application], page 15 hold.

- a set of subprograms allowing the user code to manipulate the thread interface (see the thread part of the AADL standard for more details).

- a subprogram that represents the thread job. The defining identifier of the subprogram is mapped from the thread instance name in the process that models the node, to which we append the string "`_Job`". All the naming rules listed in Section B.1 [Whole distributed application], page 15 have to be respected. The body of this subprogram calls the subprograms mapped from the subprogram calls the thread performs. Then,

it sends the request to the remote threads it may be connected to. For periodic threads, the job subprogram is parameter-less. For sporadic threads, the job subprogram takes one parameter, the port the enumerator corresponding to the port that triggered the thread. For hybrid threads, if the thread has only a compute entrypoint then the job subprogram must have one parameter, the port the enumerator corresponding to the port that triggered the thread. If the hybrid thread has been assigned a compute entry point for each one of its ports, the job subprogram must be parameter-less.

- an Ada task. To simplify code generation, we just instantiate a generic package supplied with the PolyORB-HI-Ada runtime: `PolyORB_HI.Cyclic_Task`, `PolyORB_HI.Sporadic_Task` or `PolyORB_HI.Hybrid_Task`. This creates a task with the wanted properties at the elaboration time of the node. The package instantiation name is mapped from the thread instance name in the process that model the node, to which we append the string "`_Task`". All the naming rules listed in Section B.1 [Whole distributed application], page 15 hold. The package instantiation takes the following parameters:
  - the enumerator corresponding to the thread
  - the task period or inter-arrival time,
  - the task priority. If the user did not specify a priority, then `System.Default_Priority` is used,
  - the task stack size. If the user did not specify a stack size, then the 64Kb default size is used,
  - for sporadic and hybrid threads only, the subprogram that is called to wait for incoming events.
  - the task job which corresponds to the subprogram `<Thread_Name>_Job`.
  - an optional subprogram to initialize the thread (given by standard property `"Initialize_Entrypoint"`.

**IMPORTANT NOTE:** If a node contains at least one hybrid thread, an extra "driver" task is added to this node to awaken hybrid tasks at their period occurrences by sending an event to their extra in event port. The priority of this task is equal to the maximal system task priority.

## B.5 Connections

The connections are entities that support communication between the application nodes. In this section, we present the AADL entities used to model connection between nodes. There is no implicit mapping rules for AADL connections, they just help to know the data flow (in case of data connections) and some aspects of the control flow (event connections) in the distributed application.

### B.5.1 AADL entities

As said in Section B.2 [Distributed application nodes], page 16 and Section B.1 [Whole distributed application], page 15 a connection between two nodes of the distributed application is modeled by:

- The `ports` features that exist on each one of the nodes. Ports can be declared inside processes or threads. The direction of the port (`in`, `out` or `in out`) indicates the direction of the information flow.
- The `connections` section in the system implementation relative to the distributed application and in the process and thread implementations.

```
package PING_Package
public
  system implementation PING.Native
  subcomponents
    Node_A : process A.Impl;
    Node_B : process B.Impl
      {Deployment::port_number => 12002;};
    CPU : processor the_processor
      {Priority_Range => 0 .. 255;};
    the_bus : bus Ethernet_Bus;
  connections
    bus access the_bus -> CPU.ETH;
    port Node_A.Out_Port -> Node_B.In_Port
      {Actual_Connection_Binding => (reference ( the_bus ));};
  properties
    actual_processor_binding => (reference ( CPU ))
      applies to Node_A;
    actual_processor_binding => (reference ( CPU ))
      applies to Node_B;
  end PING.Native;
end PING_Package;
```

The listing above shows the connection between the node `A` and `B` in the system implementation.

The nature of the `port` (*event port*, *data port* or *event data port*) depends on the nature of the connection between the two nodes:

- if the message sent from one node to another node is only a triggering event and contains no data, we create an *event* port.
- if the message sent from one node to another node is a data message but it does not trigger the receiver thread, we create a *data* port.
- if the message sent from one node to another node is a data message that triggers the receiver thread, we create an *event data* port.

# Appendix C PolyORB-HI-Ada API

This section lists the API of PolyORB-HI-Ada, used to support the basics of distribution features and concurrent interactions.

# Appendix D Frequently Asked Questions

In this section, we list frequent questions related to code generation patterns.

1. *How are allocated buffers that store messages ?*

   Ocarina has a complete view over the system models. In particular, based on the knowledge of user types, Ocarina computes the maximum size of the payload to be sent, and stores it in the `PolyORB_HI_Generated.Deployment.Max_Payload_Size` constant. See Section B.2 [Distributed application nodes], page 16.

2. *How many Ada task are generated ?*

   - For each process, Ocarina generates one task per AADL thread, plus one additional task per process that holds an AADL `Hybrid` thread. See Section B.4 [Threads], page 29.
   - If this process relies on pre-defined transport protocol (using the `Deployment_Transport_API` property with a value different from `None` or `User`), then an additional task is added.

3. *How many Ada protected objects are generated ?*

   - For each process, Ocarina instantiates one protected object per AADL thread with "in" ports, as part of the instantiation building node activity. See Section B.2 [Distributed application nodes], page 16.
   - In addition, for each AADL data component whose `Concurrency_Protocol_Policy` is `Priority_Ceiling`, Ocarina builds one protected object.

4. *How to enable debug information ?*

   To enable debug, you must configure PolyORB-HI-Ada with the `--enable-debug` flag. See Section 2.4 [Installation instructions], page 6.

5. *How to display log information ?*

   To display log information at runtime, you need to change the value of `PolyORB_HI.Output.Current_Mode`.

# Appendix E References


1. [Ada05] ISO SC22/WG9. *Ada Reference Manual. Language and Standard Libraries. Consolidated Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1 (Draft 15)*, 2005. Available on urlhttp://www.adaic.com/standards/rm-amend/html/RM-TTL.html.
2. [ISO05] ISO/IEC. *TR 24718:2005 — Guide for the use of the Ada Ravenscar Profile in high integrity systems*, 2005. Based on the University of York Technical Report YCS-2003-348 (2003).
3. [SAE04] SAE. *Architecture Analysis & Design Language (AS5506)*. SAE, sep 2004. available at urlhttp://www.sae.org.
4. [SAE09a] SAE. *Architecture Analysis & Design Language v2 (AS5506A)*. SAE, jan 2009. available at urlhttp://www.sae.org.
5. [SAE09b] SAE. *Data Modeling Annex for the Architecture Analysis & Design Language v2 (AS5506A)*. SAE, nov 2009. available at urlhttp://www.sae.org.
6. [SAE09c] SAE. *Programming Language Annex for the Architecture Analysis & Design Language v2 (AS5506A)*. SAE, nov 2009. available at urlhttp://www.sae.org.
7. [VZH06] T. Vergnaud, B. Zalila, and J. Hugues. Ocarina: a Compiler for the AADL. Technical report, Telecom Paris, 2006.

# Appendix F GNU Free Documentation License

Version 1.1, March 2000



## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the

Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Heading 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute

the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# The Index