PolyORB High Integrity User's Guide

C Edition
Version 1.0w
Date: 16 May 2011

Julien Delange, Jérôme Hugues

Table of Contents

Table of Contents

About This Guide

This document describes PolyORB High-Integrity C (PolyORB-HI-C), 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-C.

What This Guide Contains

This guide contains the following chapters:

- Chapter 1 [Introduction to PolyORB-HI-C], page 3 provides a brief description of middleware and PolyORB-HI-C's architecture.
- Chapter 2 [Configuration], page 5 details how to configure PolyORB-HI-C.
- Chapter 3 [Building a system], page 9 details how to build a distributed system from its AADL description.
- Appendix A [Supported features], page 11 details the features that are available in PolyORB-HI-C,as well as the restrictions on the language it follows.
- Appendix B [AADL to C transformations], page 13 details the mapping rules to map an AADL model onto a High-Integrity Distributed System.
- Appendix C [PolyORB-HI-C API], page 41 provides an overview of PolyORB-HI-C API.
- Appendix E [References], page 61 provides a list of useful references to complete this documentation.
- Appendix F [GNU Free Documentation License], page 63 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.c`’. If you are working on a Windows platform, please note that the “\” character should be used instead.

1 Introduction to PolyORB-HI-C

PolyORB-HI-C 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-C relies on the AADL language and the Ocarina toolsuite (*[VZH06]*) to allocate all required ressources and generate stubs, skeletons, marshallers and concurrent structures.

Ocarina/PolyORB-HI-C supports both AADLv1 and AADLv2 as input models.

This manual describes the different elements parts of PolyORB-HI-C.

2 Configuration

2.1 Supported Platforms

PolyORB-HI-C has been compiled and sucessfully tested on

- native platforms
  - Linux
  - Mac OS X
  - Solaris
  - FreeBSD
  - Windows
- embedded platforms
  - RTLinx, using Elinos
  - Nintendo DS (tm) (Linux) - see http://www.dslinux.org
  - Nokia 770 (Linux) - see http://www.maemo.org
  - LEON (SPARC-like CPU) (RTEMS)
  - Spif (PowerPC CPU) (RTEMS) - see http://www.enst.fr/~spif/

Note: PolyORB-HI-C should compile and run on every POSIX-compliant system. Its network stack uses the socket API, and is compatible with many operating systems.

2.2 Tree structure

PolyORB-HI-C has the following tree structure:

- '`doc/`': documentation,
- '`examples/`': set of examples to test PolyORB-HI-C
- '`share/`': common files (aadl files used by Ocarina, makefiles, ...)
- '`src/`': core of PolyORB-HI
- '`tools/`': some script to handle the packaging and a verification tool to check if the binaries are compliant with the POSIX restrictions.
- '`ChangeLog`': release information,
- '`COPYING`': GPLv2 licence document,
- '`README`': short description of the distribution.

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-c/`': set of examples to test PolyORB-HI-C
- runtime files are in '`$OCARINA_PATH/include/ocarina/runtime/polyorb-hi-c/`'.

2.3 Build requirements

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

- a C compiler that produces binaries for the target architecture.
- a standard C-library, for common functions like `socket()` or `pthread_createt()`.

Note: For each tested bare board, the toolchains provides Makefiles to configure additional environment variables

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

2.4 Configuration instructions

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

- Install your C compiler 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. At the end of the configuration process, a file with all parameters is created ('`include/po_hi_config.h`'. If this file is not created, the compilation is not possible.
- Issue `make && make install`

2.4.1 Examples of configuration

2.4.1.1 Configure the framework for the LEON platform (case of cross-compilation)

In this example, we want configure the framework for the LEON platform and the RTEMS operating system. The LEON architecture is similar to the SPARC. In other words, we have to use a compiler that is different from the one used to compile native binaries. In our case, the compiler is called `sparc-rtems-gcc`. Consequently, the host name will be `sparc-rtems`. This name will be used in the configuration step, with the `--host` option.

If we want to configure the target with the GIOP protocol, we just have to invoke the following command :

`./configure --enable-giop --host=sparc-rtems`

The `configure` script will detect that you are cross-compiling and will detect the endianness of the processor and the size of each types. If no error was detected, the framework is ready to be used. The following example reproduce the output when we use this type of configuration. Note that the `build` switch is optionnal. If you don't provide it, the script will automatically detect your system type.

```
$ ./configure --host=sparc-rtems --build=i386-linux --enable-giop
```

Note: If you use RTEMS, you have to define the `RTEMS_MAKEFILE_PATH` *as RTEMS documentation describe it. Generally, you have to follow all instructions that are described with the system you will use.*

2.4.1.2 Configure the framework for the native platform

Compile for the native platform means that you will use the same machine to compile the code and run the binaries. Most of the time, the compiler will be `gcc`. In the present case, we will configure the framework for the RAW protocol. We just invoke the following command :

```
./configure
```

If no error was detected, the framework is ready to use. If you encounter errors, please check that you have a compiler that works fine.

2.5 Build instructions

PolyORB-HI-C 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. 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 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-C and an AADL model of the application.

3.1 Building examples

Each example provide a makefile that does the following steps:

1. parse the AADL model;
2. generate C code from the AADL model;
3. compile each node

PolyORB-HI-C 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 that has to be used, needed Ocarina non-standard property sets:

```
system mysystem
properties
  Ocarina_Config::AADL_Files           =>
    ("mymodel1.aadl", "mymodel2.aadl");
  Ocarina_Config::Generator            => polyorb_hi_c;
  Ocarina_Config::Needed_Property_Sets =>
    (ARAO, Cheddar_Properties);
end mysystem;

system implementation mysystem.Impl
end  mysystem.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_c <list-of-aadl-files>`

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

Appendix A Supported features

A.1 C constructions and restrictions

PolyORB-HI-C introduces Ravenscar-like restrictions on C concurrent features.

Moreover, the code is compliant with the Application Environment Profile (AEP) defined by the OMG. If these constructions are warrant on the underlying middleware, it does not apply on the user code. In other words, the code provided by the user and used by the generated code must be written carefully.

If you want to check that your application if compliant with the AEP profile, use the `check-symbols` tool, available in the '`tools`' directory.

A.2 AADL features

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

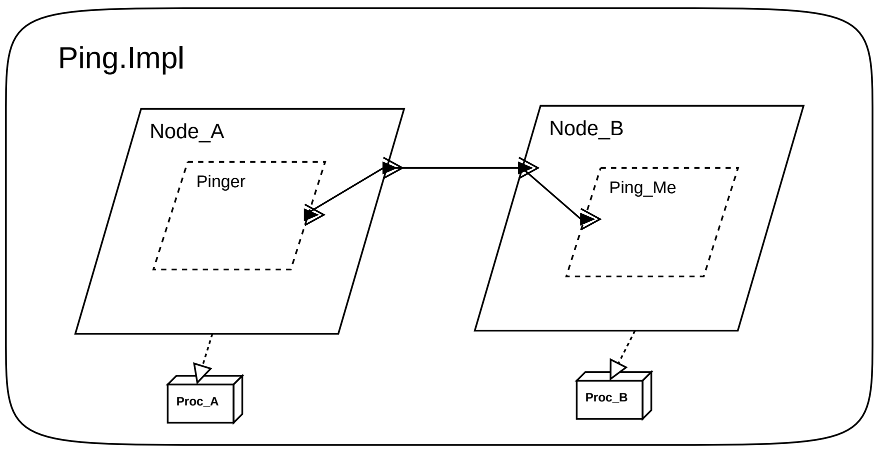
This section lists AADL features supported by PolyORB-HI-C:

- asynchronous, oneway calls;
- data component types of statically bounded size;
- all compile-time and run-time restrictions enforced as part of the compilation process;
- PolyORB-HI-Ada can use different transport infrastructures:
  - on native platform, distribution can be tested using the native socket library.
  - user-provided transport layer can be used, provided they follow guidelines discussed in section **XXX**.

Appendix B AADL to C transformations

In the following, and for each component of the distributed application, we give the AADL entities that are used to model this component, and then the transformation rules used by the code generator to generate C code from these entities.

The mapping rules will be illustrated using the following simple 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.

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 AADL entities onto instances VM-level container, expressed as C 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 in the sections that deals with these respective components.

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.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
  -- Nodes
```

```
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  --  Processors
  --  ...
connections
  --  ...
properties
  --  ...
end PING.Impl;
```

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 [Hosts], page 20 for more details) to map the different nodes on the different platforms of the distributed application. The `connections` section of the system implementation models the connections between the different nodes of the application.

B.1.2 C mapping rules

A distributed application is mapped into a hierarchy of directories:

- the root directory of the distributed application which has the same name as the system implementation that model the application, in lower case, all dot being converted into underscores. This directory is the root of the directory hierarchy of the generated C 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 14 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 C 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.

```
process A
features
  Out_Port : out event data port Simple_Type;
end A;
```

```
process implementation A.Impl
subcomponents
  Pinger : thread P.Impl;
connections
  event data port Pinger.Data_Source -> Out_Port;
end A.Impl;
```

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 29 for more details on connections mapping).

B.2.2 C mapping rules

All the C 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 the process `A` of the `Ping` example are generated in the directory `ping_impl/node_a`.

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

B.2.2.1 Marshallers functions

The marshallers functions are used to put all request and types values in a message in order to send them through a network connections. All marshalling functions are declared in the file `marshallers.c`

B.2.2.2 Node activity

We denote "activity" the set of the actions performed by one particular node which are not triggered by other nodes. All the periodic threads of a node are part of the node activity.

The code related to the node activity is generated in an C file with the name '`activity.c`'. An example is shown below :

```
#include <po_hi_types.h>
#include <po_hi_gqueue.h>
#include <request.h>
#include <deployment.h>
#include <types.h>
#include <subprograms.h>
#include <po_hi_task.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_int8_t __po_hi_data_source_local_destinations[1] = {ping_me_global_data_sink};
__po_hi_uint8_t __po_hi_pinger_woffsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_offsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_used_size[__po_hi_pinger_nb_ports];
```

```
__po_hi_uint8_t __po_hi_pinger_empties[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_first[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_recent[__po_hi_pinger_nb_ports * sizeof(__po_hi_request_t)];
__po_hi_uint8_t __po_hi_pinger_queue[0 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_pinger_total_fifo_size = 0;
__po_hi_port_t __po_hi_pinger_history[0];
__po_hi_uint8_t __po_hi_pinger_n_dest[__po_hi_pinger_nb_ports] = {1};
__po_hi_int8_t __po_hi_pinger_fifo_size[__po_hi_pinger_nb_ports] = {__PO_HI_GQUEUE_FIFO_OUT};
__po_hi_uint8_t* __po_hi_pinger_destinations[__po_hi_pinger_nb_ports] = {__po_hi_data_source_local_destin
/*  Periodic task : Pinger*/

/****************/
/* pinger_job   */
/****************/

void* pinger_job ()
{
   simple_type data_source_request_var;
   __po_hi_request_t data_source_request;

   __po_hi_gqueue_init(node_a_pinger_k,__po_hi_pinger_nb_ports,__po_hi_pinger_queue,__po_hi_pinger_fifo_s
   __po_hi_wait_initialization();
   while (1)
   {
      /*  Call implementation*/
      do_ping_spg(&(data_source_request_var));
      /*  Set the OUT port values*/
      data_source_request.vars.pinger_global_data_source.pinger_global_data_source = data_source_request_
      data_source_request.port = data_source_request_var;
      __po_hi_gqueue_store_out(node_a_pinger_k,pinger_local_data_source,&(data_source_request));
      /*  Send the OUT ports*/
      __po_hi_gqueue_send_output(node_a_pinger_k,pinger_global_data_source);
      __po_hi_wait_for_next_period(node_a_pinger_k);
   }
}


/*************************/
/* __po_hi_main_deliver   */
/*************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      default:
      {
         break;
      }
   }
}
```

All the naming rules explained in Section B.1 [Whole distributed application], page 13 are also applied to map the package name. This file contains all the routines mapped from the periodic threads that belong to the handled node (see Section B.4 [Threads], page 21 for more details on thread mapping). This package contains also the instances of shared objects used in this node (see Section B.7 [Data], page 34 for more details). If the node does not contain any *periodic* thread nor shared objects, there is no '`activity.c`' file generated for this node. Thus, the node `B` in the `Ping` example does not have a '`activity.c`' package.

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 C file called '`types.h`'.

For more detail on the mapping of data components, see Section B.7 [Data], page 34.

B.2.2.4 Subprograms

The mapping of all AADL subprogram components used by a particular node is generated in a separate file called '`subprograms.c`'. The content of the file is shown in the following example:

For more detail on the mapping of subprogram components, see Section B.6 [Subprograms], page 33.

B.2.2.5 Deployment information

The deployment information is the information each node has on the other nodes in 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 two C files : '`deployment.h`' and '`deployment.c`'.

The file '`deployment.h`' contains the following types

- a first type called `__po_hi_node_t`. For each node in the application we create an enum 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 13 have to be respected.
- a second type called `__po_hi_entity_t`. For each thread in the the application, we declare an enum.
- a third type called `__po_hi_task_id`. For each thread that run on the current node.
- a fourth type called `__po_hi_entity_server_t`. For each node that may communicate with the current node, we add a value in this enum. It will be used by the transport layer. Please note that at least one server is declared : the value `invalid_server`.
- a fifth type called `__po_hi_port_t` that contains all global port identifier.

More, this file contains the following maccros :

- `__PO_HI_NB_ENTITIES` is the number of entities in the whole distributed system.
- `__PO_HI_NB_TASKS` is the number of the tasks that will be started on the current node
- `__PO_HI_NB_NODES` is the number of nodes in the distributed system.

- `__PO_HI_PROTECTED` is the number of protected objects use on the current node.
- `__PO_HI_NB_PORTS` that represent the total number of ports in the whole distributed system.

The file 'deployment.c' contains three variables :

- `mynode` variable which has the value of the handled node.
- `__po_hi_entity_table` variable is used to know on which node an entity runs.
- `__po_hi_port_global_to_local` variable is used to convert a global port identifier to a local port identifier
- `__po_hi_port_global_to_entity` variable is used to know on which entity a given port is. This table is used convert a global port identifier to an entity identifier.

The following example shows the `Deployment` package relative to the node `A` of the `Ping` example:

```
#ifndef __DEPLOYMENT_H_
#define __DEPLOYMENT_H_
#include <po_hi_protected.h>
typedef enum
{
   pinger_local_data_source = 0
} __po_hi_pinger_t;

#define __po_hi_pinger_nb_ports 1

typedef enum
{
   ping_me_local_data_sink = 0
} __po_hi_ping_me_t;

#define __po_hi_ping_me_nb_ports 1

/*  For each node in the distributed application add an enumerator*/

typedef enum
{
   node_a_k = 0,
   node_b_k = 1
} __po_hi_node_t;

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

typedef enum
{
   node_a_pinger_k_entity = 0,
   node_b_ping_me_k_entity = 1
} __po_hi_entity_t;

typedef enum
{
   node_a_pinger_k = 0
} __po_hi_task_id;

#define __PO_HI_NB_TASKS 1
```

```
/*  For each thread in the distributed application nodes THAT MAY COMMUNICATE*/
/*   with the current node, add an enumerator*/

typedef enum
{
   invalid_server = -1
} __po_hi_entity_server_t;

#define __PO_HI_NB_SERVERS 0

#define __PO_HI_NB_PROTECTED 0

#define __PO_HI_NB_NODES 2

#define __PO_HI_NB_ENTITIES 2

#define __PO_HI_NB_PORTS 2

typedef enum
{
   pinger_global_data_source = 0,
   ping_me_global_data_sink = 1
} __po_hi_port_t;

#endif
```

B.2.2.6 Naming information

The naming information for a particular node `A` allow 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 file called '`naming.c`'.

```
#include <po_hi_protocols.h>
#include <deployment.h>
/*  Naming Table*/
__po_hi_inetport_t node_port[__PO_HI_NB_NODES] = {__PO_HI_NOPORT,12002};
__po_hi_inetaddr_t node_addr[__PO_HI_NB_NODES] = {__PO_HI_NOADDR,"127.0.0.1"};
```

As shown in the example above, for the node `A` of the `Ping` example, the '`naming.c`' file contains:

- An array called `node_port` indexed by the values of `__po_hi_node_t`. It tells the port to connect on for each node in the distributed system.
- An array called `node_addr` indexes by the values of `__po_hi_node_t`. It tells the address to connect on for each node.

B.2.2.7 Main function

The main function is a function that does all the necessary initialization before the effective run of the node. This function is stored in a file called '`main.c`'. This function initializes the components of the node (protected types, network layer, ...), creates the tasks and wait that all components are initialized. The following example shows the main subprogram generated for the node `A` of the `Ping` example.

```
#include <activity.h>
#include <po_hi_common.h>
#include <po_hi_main.h>
#include <po_hi_time.h>
#include <po_hi_task.h>
/***********************/
/* __PO_HI_MAIN_NAME */
/***********************/

__PO_HI_MAIN_TYPE __PO_HI_MAIN_NAME ()
{

   __po_hi_initialize();
   __po_hi_create_periodic_task(node_a_pinger_k,__po_hi_milliseconds(5000),2,pinger_job);
   __po_hi_wait_initialization();
   __po_hi_wait_for_tasks();
   return (__PO_HI_MAIN_RETURN);
}
```

B.3 Hosts

A host 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 host. Then, we give the mapping rules used to generate C code expressing that a node runs on a particular host.

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 host must have an IP address. Each host must also precise its platform (native, LEON...). The listing following example shows how to express this using a custom property set.

```
processor the_processor
properties
  ARAO::location           => "127.0.0.1";
  ARAO::Execution_Platform => Native;
end the_processor;
```

To map an application node (processor) to a particular host, 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.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
```

```
  --  Nodes
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  --  Processors
  CPU_A  : processor the_processor;
  CPU_B  : processor the_processor;
connections
  --  ...
properties
  --  Processor bindings
  actual_processor_binding => reference CPU_A applies to Node_A;
  actual_processor_binding => reference CPU_B applies to Node_B;
end PING.Impl;
```

B.3.2 C mapping rules

The C generated code concerning the code generation to model host mapping is located in the '`naming.c`' file. More precisely, the `node_addr` and `node_port` contains, for each node, the information related to its host. 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 C 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 14 and Section B.1 [Whole distributed application], page 13). 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 describe the thread 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) and many other properties ares expressed using AADL properties. The `calls` section of the thread implementation contains the sequences of subprograms the thread may call during its job (see Section B.6 [Subprograms], page 33 for more details on the subprogram mapping). 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 `connections` section of a thread implementation connects the parameters of the subprograms called by the thread to the ports of the threads or to the parameters of other called subprograms in the same thread.

```
thread P
features
  Data_Source : event out data port Simple_Type;
```

```
end P;

thread implementation P.Impl
calls {
  -- ...
};
connections
  -- ...
properties
  Dispatch_Protocol => Periodic;
  Period            => 1000 Ms;
end P.Impl;
```

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 C mapping rules for periodic 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.

B.4.2.1 Node activity

The majority of the code generated for the periodic threads is put in the '`activity.c`' file generated for the application node containing the handled thread. Each periodic thread is created in the main function ('`main.c`' file) with the `__po_hi_create_periodic_task` function-call.

The generated code in the '`activity.c`' file is a parameterless function that represents the thread job. The defining identifier of the function 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 13 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. Finally, at the end of the function, we make a call to the `__po_hi_wait_next_period()` with the task identifier as parameter. This call ensure that we wait the next period before we start the function again.

The generated code in '`main.c`' file is a function-call that creates a periodic task. The task is created with the function `__po_hi_create_periodic_task`. This creates a periodic 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 "`_k`". All the naming rules listed in Section B.1 [Whole distributed application], page 13 have to be respected. The function-call takes the following parameters:

- the enumerator corresponding to the thread
- the task period,

- the task priority. If the user did not specify a priority, then `__PO_HI_DEFAULT_PRIORITY` is used,
- the task job which corresponds to the subprogram `<Thread_Name>_job`.

The following example shows the generated code for the periodic thread `Pinger` from the node `Node_A` of the `Ping` example:

```
#include <po_hi_types.h>
#include <po_hi_gqueue.h>
#include <request.h>
#include <deployment.h>
#include <types.h>
#include <subprograms.h>
#include <po_hi_task.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_int8_t __po_hi_data_source_local_destinations[1] = {ping_me_global_data_sink};
__po_hi_uint8_t __po_hi_pinger_woffsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_offsets[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_used_size[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_empties[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_first[__po_hi_pinger_nb_ports];
__po_hi_uint8_t __po_hi_pinger_recent[__po_hi_pinger_nb_ports * sizeof(__po_hi_request_t)];
__po_hi_uint8_t __po_hi_pinger_queue[0 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_pinger_total_fifo_size = 0;
__po_hi_port_t __po_hi_pinger_history[0];
__po_hi_uint8_t __po_hi_pinger_n_dest[__po_hi_pinger_nb_ports] = {1};
__po_hi_int8_t __po_hi_pinger_fifo_size[__po_hi_pinger_nb_ports] = {__PO_HI_GQUEUE_FIFO_OUT};
__po_hi_uint8_t* __po_hi_pinger_destinations[__po_hi_pinger_nb_ports] = {__po_hi_data_source_local_destin
/*  Periodic task : Pinger*/

/****************/
/* pinger_job   */
/****************/

void* pinger_job ()
{
   simple_type data_source_request_var;
   __po_hi_request_t data_source_request;

   __po_hi_gqueue_init(node_a_pinger_k,__po_hi_pinger_nb_ports,__po_hi_pinger_queue,__po_hi_pinger_fifo_s
   __po_hi_wait_initialization();
   while (1)
   {
      /*  Call implementation*/
      do_ping_spg(&(data_source_request_var));
      /*  Set the OUT port values*/
      data_source_request.vars.pinger_global_data_source.pinger_global_data_source = data_source_request_
      data_source_request.port = data_source_request_var;
      __po_hi_gqueue_store_out(node_a_pinger_k,pinger_local_data_source,&(data_source_request));
      /*  Send the OUT ports*/
      __po_hi_gqueue_send_output(node_a_pinger_k,pinger_global_data_source);
      __po_hi_wait_for_next_period(node_a_pinger_k);
   }
}
```

```
/*************************/
/* __po_hi_main_deliver   */
/*************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      default:
      {
         break;
      }
   }
}
```

B.4.3 C mapping rules for sporadic threads

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

B.4.3.1 Node activity

The majority of the code generated for the sporadic threads is put in the '`activity.c`' file generated for the application node containing the handled thread. Each periodic thread is created in the main function ('`main.c`' file) with the `__po_hi_create_sporadic_task` function-call.

The generated code in the '`activity.c`' file is a parameterless function that represents the thread job. The defining identifier of the function 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 13 have to be respected. In the body of the function, the thread will wait for an event (most of the time : a message from another entity).

The generated code in '`main.c`' file is a function-call that creates the sporadic task. The task is created with the function `__po_hi_create_sporadic_task`. This creates a sporadic 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 "`_k`". All the naming rules listed in Section B.1 [Whole distributed application], page 13 have to be respected. The function-call takes the following parameters:

- the enumerator corresponding to the thread
- the task priority. If the user did not specify a priority, then `__PO_HI_DEFAULT_PRIORITY` is used,

- the task job which corresponds to the subprogram `<Thread_Name>_job`.

The following example shows the generated code for the sporadic thread `Ping_Me` from the node `Node_B` of the `Ping` example.

```
#include <po_hi_gqueue.h>
#include <po_hi_types.h>
#include <request.h>
#include <deployment.h>
#include <po_hi_task.h>
#include <subprograms.h>
#include <po_hi_main.h>
#include <marshallers.h>
extern __po_hi_entity_t __po_hi_port_global_to_entity[__PO_HI_NB_PORTS];
extern __po_hi_port_t __po_hi_port_global_to_local[__PO_HI_NB_PORTS];
__po_hi_uint8_t __po_hi_ping_me_woffsets[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_offsets[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_used_size[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_empties[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_first[__po_hi_ping_me_nb_ports];
__po_hi_uint8_t __po_hi_ping_me_recent[__po_hi_ping_me_nb_ports * sizeof(__po_hi_request_t)];
__po_hi_uint8_t __po_hi_ping_me_queue[16 * sizeof(__po_hi_request_t)];
__po_hi_uint16_t __po_hi_ping_me_total_fifo_size = 16;
__po_hi_port_t __po_hi_ping_me_history[16];
__po_hi_uint8_t __po_hi_ping_me_n_dest[__po_hi_ping_me_nb_ports] = {0};
__po_hi_int8_t __po_hi_ping_me_fifo_size[__po_hi_ping_me_nb_ports] = {16};
__po_hi_uint8_t* __po_hi_ping_me_destinations[__po_hi_ping_me_nb_ports] = {NULL};

/*********************/
/* ping_me_deliver   */
/*********************/

void ping_me_deliver
      (__po_hi_request_t* request)
{

   switch (request->port)
   {
      case ping_me_global_data_sink:
      {
         __po_hi_gqueue_store_in(node_b_ping_me_k,ping_me_local_data_sink,request);

         break;
      }
      default:
      {
         break;
      }
   }
}

/*  Sporadic task : Ping_Me*/
/*  Get the IN ports values*/

/*****************/
/* ping_me_job   */
/*****************/
```

```
void* ping_me_job ()
{
   __po_hi_port_t port;
   __po_hi_request_t data_sink_request;

   __po_hi_gqueue_init(node_b_ping_me_k,__po_hi_ping_me_nb_ports,__po_hi_ping_me_queue,__po_hi_ping_me_fi
   __po_hi_wait_initialization();
   while (1)
   {
      __po_hi_gqueue_wait_for_incoming_event(node_b_ping_me_k,&(port));
      __po_hi_compute_next_period(node_b_ping_me_k);
      if (__po_hi_gqueue_get_count(node_b_ping_me_k,ping_me_local_data_sink))
      {
               __po_hi_gqueue_get_value(node_b_ping_me_k,ping_me_local_data_sink,&(data_sink_request));
         __po_hi_gqueue_next_value(node_b_ping_me_k,ping_me_local_data_sink);

      }
      /*  Call implementation*/
      ping_spg(data_sink_request.vars.ping_me_global_data_sink.ping_me_global_data_sink);
      __po_hi_wait_for_next_period(node_b_ping_me_k);
   }
}


/**************************/
/* __po_hi_main_deliver   */
/**************************/

void __po_hi_main_deliver
      (__po_hi_msg_t* message)
{
   __po_hi_request_t request;
   __po_hi_entity_t entity;

   __po_hi_unmarshall_request(&(request),message);
   entity = __po_hi_port_global_to_entity[request.port];
   switch (entity)
   {
      case node_b_ping_me_k_entity:
      {
         ping_me_deliver(&(request));

         break;
      }
      default:
      {
         break;
      }
   }
}
```

B.4.4 Deployment information

As said in Section B.2 [Distributed application nodes], page 14, the files '`deployment.h`' and '`deployment.c`' are generated for each node in the distributed application. For each thread port in the whole distributed application, we declare an enumerator in this type. 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`. For each that that may communicate, we generate the following elements

- A variable called `__po_hi_<thread_name>_local_to_global` (in `deployment.c`) that is used to convert a local port identifier of the thread to a global one.
- A type `__po_hi_<thread_name>_t` that will contain on local port identifier.
- A macro `__po_hi_<thread_name>_nb_ports` that will contain the number of ports for the thread.

For these elements, all the naming rules listed in Section B.1 [Whole distributed application], page 13 must be respected.

```
#ifndef __DEPLOYMENT_H_
#define __DEPLOYMENT_H_
#include <po_hi_protected.h>
typedef enum
{
   pinger_local_data_source = 0
} __po_hi_pinger_t;

#define __po_hi_pinger_nb_ports 1

typedef enum
{
   ping_me_local_data_sink = 0
} __po_hi_ping_me_t;

#define __po_hi_ping_me_nb_ports 1

/*  For each node in the distributed application add an enumerator*/

typedef enum
{
   node_a_k = 0,
   node_b_k = 1
} __po_hi_node_t;

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

typedef enum
{
   node_a_pinger_k_entity = 0,
   node_b_ping_me_k_entity = 1
} __po_hi_entity_t;

typedef enum
{
   node_a_pinger_k = 0
} __po_hi_task_id;

#define __PO_HI_NB_TASKS 1

/*  For each thread in the distributed application nodes THAT MAY COMMUNICATE*/
/*   with the current node, add an enumerator*/
```

```
typedef enum
{
   invalid_server = -1
} __po_hi_entity_server_t;

#define __PO_HI_NB_SERVERS 0

#define __PO_HI_NB_PROTECTED 0

#define __PO_HI_NB_NODES 2

#define __PO_HI_NB_ENTITIES 2

#define __PO_HI_NB_PORTS 2

typedef enum
{
   pinger_global_data_source = 0,
   ping_me_global_data_sink = 1
} __po_hi_port_t;

#endif


#include <deployment.h>
__po_hi_entity_server_t server_entity_table[__PO_HI_NB_ENTITIES] = {invalid_server,invalid_server};
__po_hi_node_t entity_table[__PO_HI_NB_ENTITIES] = {node_a_k,node_b_k};
__po_hi_node_t mynode = node_a_k;
```

The listing above shows the generated `__po_hi_entity_server_t` and `entity_table` for the nodes `B` from the `Ping` example.

B.4.5 Port mapping

Threads can contain one or several ports. To handle them, we declared several arrays in the `activity.c`

- `__po_hi_<port_name>_destinations` : array for each port of the thread which contains all destinations of the port.
- `__po_hi_<thread_name>_woffsets` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_offsets` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_used_size` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_empties` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_first` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_recent` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.

- `__po_hi_<thread_name>_queue` : array (size = size of the global queue for the thread) used by \pohic to handle the global queue.
- `__po_hi_<thread_name>_total_fifo_size` : variable that contains the size of the global queue. It is the sum of all port size for the thread.
- `__po_hi_<thread_name>_history` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_n_dest` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread. It contains the number of destinations for each port of the thread.
- `__po_hi_<thread_name>_fifo_size` : array (size = number of ports in the thread) used by \pohic for the global queue of the thread.
- `__po_hi_<thread_name>_destinations` : array (size = number of ports in the thread) that contains all destinations for each port.

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. In this section, we will talk about how data are sended (marshall functions) and which functions are used to send and receive data.

B.5.1 AADL entities

As said in Section B.2 [Distributed application nodes], page 14 and Section B.1 [Whole distributed application], page 13 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.

```
system PING
end PING;

system implementation PING.Impl
subcomponents
  --  Nodes
  Node_A : process A.Impl;
  Node_B : process B.Impl {ARAO::port_number => 12002;};

  --  Processors
  CPU_A  : processor the_processor;
  CPU_B  : processor the_processor;
connections
  --  Port connections
  event data port Node_A.Out_Port -> Node_B.In_Port;
properties
```

```
  --  Processor bindings
  actual_processor_binding => reference CPU_A applies to Node_A;
  actual_processor_binding => reference CPU_B applies to Node_B;
end PING.Impl;
```

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.

B.5.2 Marshallers

In a distributed system, when we send any data to a node, we need to put them in a stream. We call that the marshall operation. On the other hand, find data in a stream is called the unmarshall operation. In each distributed application, we generate marshallers for each types and request. These functions will marshall/unmarshall data in/from a message.

All marshallers functions are generated in a file called '`marshallers.c`'. The marshall (or unmarshall) functions for request are prefixed by the string `__po_hi_marshall_request_` (or `__po_hi_unmarshall_request_`). Marshall (or unmarshall) functions for types are prefixed by the string `__po_hi_marshall_type_` (or `__po_hi_unmarshall_type_`). Each function has the name of the type or the request it marshalls.

Finally, a function `__po_hi_marshall_request` and `__po_hi_unmarshall_request` is generated to handle all requests. Then, is called the appropriate function to call to marshall or unmarshall the data.

```
#include <types.h>
#include <po_hi_types.h>
#include <po_hi_marshallers.h>

/**************************************/
/* __po_hi_marshall_type_simple_type   */
/**************************************/

void __po_hi_marshall_type_simple_type
      (simple_type value,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_int(value,message,offset);
}


/****************************************/
/* __po_hi_unmarshall_type_simple_type   */
```

```
/****************************************/

void __po_hi_unmarshall_type_simple_type
      (simple_type* value,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_unmarshall_int(value,message,offset);
}


/**********************************************/
/* __po_hi_marshall_request_ping_me_data_sink   */
/**********************************************/

void __po_hi_marshall_request_ping_me_data_sink
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_type_simple_type(request->vars.ping_me_global_data_sink.ping_me_global_data_sink,mess
}


/************************************************/
/* __po_hi_unmarshall_request_ping_me_data_sink   */
/************************************************/

void __po_hi_unmarshall_request_ping_me_data_sink
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_unmarshall_type_simple_type(&(request->vars.ping_me_global_data_sink.ping_me_global_data_sink)
}


/***********************************************/
/* __po_hi_marshall_request_pinger_data_source   */
/***********************************************/

void __po_hi_marshall_request_pinger_data_source
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_marshall_type_simple_type(request->vars.pinger_global_data_source.pinger_global_data_source,me
}


/*************************************************/
/* __po_hi_unmarshall_request_pinger_data_source   */
/*************************************************/
```

```
void __po_hi_unmarshall_request_pinger_data_source
      (__po_hi_request_t* request,
      __po_hi_msg_t* message,
      __po_hi_uint16_t* offset)
{

   __po_hi_unmarshall_type_simple_type(&(request->vars.pinger_global_data_source.pinger_global_data_sourc
}


/*****************************/
/* __po_hi_marshall_request   */
/*****************************/

void __po_hi_marshall_request
      (__po_hi_request_t* request,
      __po_hi_msg_t* message)
{
   __po_hi_uint16_t offset;

   offset = 0;
   __po_hi_marshall_port(request->port,message);
   switch (request->port)
   {
      case ping_me_global_data_sink:
      {
         __po_hi_marshall_request_ping_me_data_sink(request,message,&(offset));

         break;
      }
      case pinger_global_data_source:
      {
         __po_hi_marshall_request_pinger_data_source(request,message,&(offset));

         break;
      }
      default:
      {
         break;
      }
   }
}


/*******************************/
/* __po_hi_unmarshall_request   */
/*******************************/

void __po_hi_unmarshall_request
      (__po_hi_request_t* request,
      __po_hi_msg_t* message)
{
   __po_hi_uint16_t offset;

   offset = 0;
   __po_hi_unmarshall_port(&(request->port),message);
   switch (request->port)
   {
```

```
        case ping_me_global_data_sink:
        {
           __po_hi_unmarshall_request_ping_me_data_sink(request,message,&(offset));

           break;
        }
        case pinger_global_data_source:
        {
           __po_hi_unmarshall_request_pinger_data_source(request,message,&(offset));

           break;
        }
        default:
        {
           break;
        }
    }
}
```

B.6 Subprograms

Subprograms are used to encapsulate behavioural aspects of the distributed application. In this section, we give the AADL entities used to model subprograms. Then we present the C mapping rules to generate code for the modeled subprograms.

B.6.1 AADL entities

To *declare* a subprogram, we use the `subprogram` AADL component. The parameters of the subprogram are specified in the `features` section of the component declaration. If the subprogram does only the job of calling other declared subprograms, then the `calls` section of the subprogram implementation has to contain such calls. To point to the *real* implementation of the subprogram, we use the AADL properties. The following example shows the AADL model for the `Do_Ping_Spg` from the `Ping` example. It precises that the C implementation of the subprogram is located in the function `user_ping`. The file which contains this function must be stored with the aadl model.

Subprograms are generally called by threads or by other subprograms. To express this, we use the `calls` section of a component implementation. Then we perform all the connections between the called subprograms *parameters* and the caller components *ports* (or *parameters* if the caller is a subprogram).

The following listing shows the calls and connections sections of the periodic thread `P` in the `Ping` example.

```
subprogram Do_Ping_Spg
features
  Data_Source : out parameter Simple_Type;
properties
  source_language => C;
  source_name     => "user_ping";
end Do_Ping_Spg;
```

B.6.2 C mapping rules for subprogram components

B.6.2.1 The subprograms package

Each subprogram instance modelize a hand-written function. In the 'subprograms.c' file, we declare the definition of this function and we generate a new one that will call the one provided by the user.

The following listing shows the calls and connections sections of the subprogram ping_spg in the Ping example.

```
#include <types.h>
#include <subprograms.h>
void user_do_ping_spg
      (simple_type* data_source);
/*****************/
/* do_ping_spg */
/*****************/

void do_ping_spg
      (simple_type* data_source)
{

   user_do_ping_spg(data_source);
}
```

B.6.3 C mapping rules for subprogram calls

For each subprogram call in a thread, we generate an C subprogram call to the subprogram implementing the thread and given by mean of the AADL properties.

On the client side, A thread sth_Job begin by calling the subprogram in its call sequence. then it calls the stubs of all the subprogram it is connected to.

On the server side, and in the function of the process_request, the subprogram implementation corresponding to the operation (coded in the message) is called.

B.7 Data

The data are the messages exchanged amongst the nodes of the distributed application. In this section, we present the AADL constructs used to model data. Then we give the C mapping rules to generate code from these constructs.

B.7.1 AADL entities

AADL data components are used to model data exchanged in the distributed application. Properties are used to precise the nature of the data.

To model a data structure (which contains fields of others data types) we use data component implementation and we add a subcomponent for each field of the structure.

The simple data types that can be modeled using AADL are (See example below):

- Booleans
- Integers

- Fixed point types
- Characters
- Wide characters

```
--  Boolean type

data Boolean_Data
properties
  ARAO::Data_Type  => Boolean;
end Boolean_Data;

--  Integer type

data Integer_Data
properties
  ARAO::Data_Type  => Integer;
end Integer_Data;

--  Fixed point type

data Fixed_Point_Type
properties
  ARAO::Data_Type    => Fixed;

  ARAO::Data_Digits  => 10;
  --  The total number of digits is 10

  ARAO::Data_Scale   => 4;
  --  The precision is 10**(-4)
end Fixed_Point_Type;

--  Character type

data Character_Data
properties
  ARAO::Data_Type  => Character;
end Character_Data;

--  Wide character type

data W_Character_Data
properties
  ARAO::Data_Type  => Wide_Character;
end W_Character_Data;
```

The complex data types that can be modeled using AADL are (See example below):

- Bounded strings
- Bounded wide strings
- Bounded arrays of a type that can be modeled
- Structure where the fields types are types that can be modeled

```
-- Bounded string type
```

```
data String_Data
properties
  ARAO::Data_Type  => String;
  ARAO::Max_Length => <User_Defined_Length>;
end String_Data;

-- Bounded wide string type

data W_String_Data
properties
  ARAO::Data_Type  => Wide_String;
  ARAO::Max_Length => <User_Defined_Length>;
end W_String_Data;

-- Bounded array type: Only the component implementation should be
-- used in the ports or parameters!

data Data_Array
properties
  ARAO::Length => <User_Defined_Length>;
end Data_Array;

data implementation Data_Array.i;
subcomponents
  -- Only one subcomponent
  Element : data String_Data;
end Data_Array.i;

-- Data structure type: Only the component implementation should be
-- used in the ports or parameters!

data Data_Structure
end Data_Structure;

data implementation Data_Structure.i;
subcomponents
  Component_1 : data String_Data;
  Component_2 : data W_String_Data;
  Component_3 : data Data_Array.i;
end Data_Structure.i;
```

Data components may also contain subprogram features. Depending on the AADL properties given by the user. These component may denote a protected object or a non protected object. In either case, they are used to model a data structure that can be handled only by the subprograms it exports (which are the feature of the data structure).

```
--  Data type of object field

data Field_Type
properties
  ARAO::Data_Type  => Integer;
end Field_Type;

--  Protected data type

data Protected_Object
features
```

```
  Update : subprogram Protected_Update;
  Read   : subprogram Protected_Read;
properties
  ARAO::Object_Kind => Protected;
  --  This property tells that we have a protected object type
end Protected_Object;

--  The implementation of the protected object

data implementation Protected_Object.Impl
subcomponents
  Field : data Field_Type;
end Protected_Object.Impl;

--  Subprograms

subprogram Protected_Update
features
  this : requires data access Protected_Object.Impl
  {required_access => access Read_Write;}; --  Mandatory
  P    : in parameter Field_Type;
properties
  source_language => Ada95;
  source_name     => "Repository";
end Protected_Update;

subprogram Protected_Read
features
  this : requires data access Protected_Object.Impl
  {required_access => access Read_Only;}; --  Mandatory
  P    : out parameter Field_Type;
properties
  source_language => Ada95;
  source_name     => "Repository";
end Protected_Read;
```

The example above shows an example of a protected data component (`Protected_Object.Impl`). The object has a single field (subcomponent) which is a simple data component. Note that the description of the feature subprograms of these data component is a little bit different from the description of classic subprograms: each feature subprogram must have a full access to the internal structure of the object type. To achieve this, we use the `require data access` facility of AADL. To model a non protected data component, user should simply change the `ARAO::Object_Kind => Protected;` into `ARAO::Object_Kind => Non_Protected;` in the implementation of data component.

B.7.2 C mapping rules

Data component declaration are mapped into C type declaration in the file `types.h`. In the following we give the C type corresponding to each data component type that could be modeled.

B.7.2.1 Simple types

Simple data components are mapped into an C type definition whose defining identifier is mapped from the component declaration identifier (with respect to the naming rules listed in Section B.1 [Whole distributed application], page 13) and whose parent subtypes is:

- `int` for boolean data types
- `int` for integer data types
- `float` for fixed point types
- `chat` for character data types

B.7.2.2 Bounded strings and wide strings

Bounded strings and wide strings are not supported in the C generator at this time.

B.7.2.3 Bounded arrays

Bounded arrays and wide strings are not supported in the C generator at this time.

B.7.2.4 Data structures

Data structures are mapped into a C structure defined in the file 'types.h'. The identifier of the record type is mapped from the data component name with respect to the naming rules given in Section B.1 [Whole distributed application], page 13. Each field defining identifier is mapped from the subcomponent name given in the data component implementation with the same naming rules. The type of the field is the C type mapped from the data corresponding component. The following example shows the C mapping of the data structure defined given earlier in this part.

```
typedef struct
{
   pos_internal_type field1;
   pos_internal_type field2;

} pos_impl;
```

B.7.2.5 Object types

Protected object types are mapped into an a C structure. We add automatically a member in the structure with the type `__po_hi_protected_id` and the name `protected_id`. This member will identify the protected type in the distributed system. All other members of the object are declared as in Data Structures (see previous subsection). The features subprograms of the object types are declared in the 'types.h' file, whereas the body of these functions are defined in the 'types.c' file. Moreover, the value of the `protected_id` must be initialized. This is done in the main function ('main.c'), before the initialization. All the naming conventions given in Section B.1 [Whole distributed application], page 13 have to be respected. The following example shows the specification of the protected type mapped from the `Protected_Object.Impl` shown earlier in this part. We show the files 'types.h', 'types.c' and 'main.c' (that initialize the `protected_id` member of the structure.

```
#ifndef __TYPES_H_
#define __TYPES_H_
```

```
#include <po_hi_types.h>
#include <po_hi_protected.h>
typedef int pos_internal_type;

typedef struct
{
   __po_hi_protected_t protected_id;

   pos_internal_type field;

} pos_impl;

void pos_impl_update
      (pos_impl* value);

void pos_impl_read
      (pos_impl* value);

#endif
```

```
#include <po_hi_protected.h>
#include <subprograms.h>
/********************/
/* pos_impl_update */
/********************/

void pos_impl_update
      (pos_impl* value)
{

   __po_hi_protected_lock(value->protected_id);
   update(&(value->field));
   __po_hi_protected_unlock(value->protected_id);
}

/******************/
/* pos_impl_read */
/******************/

void pos_impl_read
      (pos_impl* value)
{

   __po_hi_protected_lock(value->protected_id);
   read(&(value->field));
   __po_hi_protected_unlock(value->protected_id);
}
```

```
#include <activity.h>
#include <po_hi_common.h>
#include <po_hi_main.h>
#include <po_hi_time.h>
#include <po_hi_task.h>
#include <types.h>
```

```
extern pos_impl pos_data;
/***********************/
/* __PO_HI_MAIN_NAME */
/***********************/

__PO_HI_MAIN_TYPE __PO_HI_MAIN_NAME ()
{

   __po_hi_initialize();
   __po_hi_create_periodic_task(gnc_tmtc_pos_gnc_th_k,__po_hi_milliseconds(1000),250,gnc_th_job);
   __po_hi_create_periodic_task(gnc_tmtc_pos_tmtc_th_k,__po_hi_milliseconds(100),190,tmtc_th_job);
   pos_data.protected_id = 0;
   __po_hi_wait_initialization();
   __po_hi_wait_for_tasks();
   return (__PO_HI_MAIN_RETURN);
}
```

Non protected object types are mapped similarly to protected object types. The only difference, is that instead of creating a protected type, we create a generic parameterless nested package.

Appendix C PolyORB-HI-C API

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

C.1 po_hi_task.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2008, GET-Telecom Paris.
 */

#ifndef __PO_HI_TASK_H__
#define __PO_HI_TASK_H__

#if defined(POSIX) || defined (RTEMS_POSIX)
#include <semaphore.h>
#include <po_hi_time.h>
#include <pthread.h>
#include <sched.h>
#define __PO_HI_MAX_PRIORITY sched_get_priority_max(SCHED_FIFO)
#define __PO_HI_MIN_PRIORITY sched_get_priority_min(SCHED_FIFO)
#define __PO_HI_DEFAULT_PRIORITY ((sched_get_priority_min(SCHED_FIFO) + sched_get_priority_max(SCHED_FIFO

#elif defined(RTEMS_PURE)
#include <rtems.h>
#include <inttypes.h>
#include <bsp.h>
#endif

#include <po_hi_types.h>
#include <deployment.h>

typedef __po_hi_uint16_t __po_hi_priority_t;
typedef size_t __po_hi_stack_t;

/*
 * Initialize tasking entities
 * Returns SUCCESS if there is no error.
 */
int __po_hi_initialize_tasking();

/*
 * Create a periodic task.
 *
 * The task created have the identifier given by the first
 * parameter. It is created according to the period created
 * with __po_hi_* functions (like __po_hi_milliseconds())
 * and priority parameters (use the OS priority). The task execute
 * periodically start_routine.
 *
 * This function returns SUCCESS if there is no error. Else,
```

```
 * it returns the negative value ERROR_CREATE_TASK.
 */
int __po_hi_create_periodic_task (__po_hi_task_id      id,
  __po_hi_time_t       period,
  __po_hi_priority_t   priority,
  __po_hi_stack_t      stack_size,
  void*                (*start_routine)(void));

/*
 * Create a sporadic task.
 *
 * The identifier of the task is the first parameter. The period and
 * the priority of the task are stored in the second and third
 * parameter.  The code executed by the task is stored in the
 * start_routine pointer.
 *
 * Returns SUCCESS value if there is no error. Else, returns the negative
 * value ERROR_CREATE_TASK
 */
int __po_hi_create_sporadic_task (__po_hi_task_id      id,
  __po_hi_time_t       period,
  __po_hi_priority_t   priority,
  __po_hi_stack_t      stack_size,
  void*                (*start_routine)(void));

/*
 * Create a generic task
 *
 * The identifier of the task is the first parameter. The period and
 * the priority of the task are stored in the second and third
 * parameter.  The code executed by the task is stored in the
 * start_routine pointer.
 *
 * Returns SUCCESS value if there is no error. Else, returns the negative
 * value ERROR_CREATE_TASK
 */
int __po_hi_create_generic_task (__po_hi_task_id    id,
                                 __po_hi_time_t     period,
                                 __po_hi_priority_t priority,
                                 __po_hi_stack_t   stack_size,
                                 void*              (*start_routine)(void));

/*
 * Wait the end of all tasks.
 * This function typically never ends, because all tasks
 * are doing an infinite loop and never ends. It just
 * used to avoid an infinite loop in the main thread.
 */
void __po_hi_wait_for_tasks ();

/*
 * Called by a periodic task, to wait for its next period
 * The argument is the task identifier
 * Returns SUCCESS value, and if fails, returns a negative value
 */
int __po_hi_wait_for_next_period (__po_hi_task_id task);

/*
```

```
 * Sleep until the time given in argument. The second
 * argument is the task identifier which will sleep.
 * Return SUCCESS if there is no error. Else, it returns
 * a negative value : ERROR_CLOCK or ERROR_PTHREAD_COND
 */
 int __po_hi_task_delay_until (__po_hi_time_t time, __po_hi_task_id task);

/*
 * Computer the next period for a task, according to the period
 * argument given at initialization time. The argument task
 * is the task-identifier in the node (__po_hi_task_id type).
 */
 int __po_hi_compute_next_period (__po_hi_task_id task);

#endif /* __PO_HI_TASK_H__ */
```

C.2 po_hi_time.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2008, GET-Telecom Paris.
 */

#ifndef __PO_HI_TIME_H__
#define __PO_HI_TIME_H__

#include <po_hi_types.h>

#ifndef HAVE_CLOCK_GETTIME
#include <time.h>
#endif


typedef __po_hi_uint64_t __po_hi_time_t;
/*
 * Represent the time in PolyORB-HI.
 *
 * The value stored in this type depends on the system : on POSIX, it
 * is the epoch (time since 1970), on other systems, it can be the
 * number of elapsed ticks since the beginning of the program.
 *
 * The granularity of the time is in microsecond (10^-6)
 */

int __po_hi_get_time (__po_hi_time_t* mytime);
/*
 * Get the current time and store informations
 * in the structure mytime.
 * If there is an error, returns a negative value
 * (ERROR_CLOCK). Else, returns a positive value.
 */
```

```
__po_hi_time_t __po_hi_add_times (__po_hi_time_t left,
  __po_hi_time_t right);
/*
 * Add the two structures given in parameter. The returned
 * value is the result of the operation.
 */

__po_hi_time_t __po_hi_seconds (__po_hi_uint32_t seconds);
/*
 * Build a __po_hi_time_t value which contains the
 * amount of time (in seconds) represented by the
 * argument seconds.
 */

__po_hi_time_t __po_hi_milliseconds (__po_hi_uint32_t milliseconds);
/*
 * Build a __po_hi_time_t value which contains the
 * amount of time (in milliseconds) represented by the
 * argument milliseconds.
 */

__po_hi_time_t __po_hi_microseconds (__po_hi_uint32_t microseconds);
/*
 * Build a __po_hi_time_t value which contains the
 * amount of time (in microseconds) represented by the
 * argument microseconds.
 */

int __po_hi_delay_until (__po_hi_time_t time);
/*
 * sleep until the time given in argument.
 * Return SUCCESS if there is no error. Else, it returns
 * a negative value : ERROR_CLOCK or ERROR_PTHREAD_COND
 */

#ifdef NEED_CLOCK_GETTIME
#define CLOCK_REALTIME 0
int clock_gettime(int clk_id, struct timespec *tp);
#endif
/*
 * If the system doesn't support the clock_gettime function, we
 * emulate it. For example, Darwin does not support it
 */

#endif /* __PO_HI_TIME_H__ */
```

C.3 po_hi_marshallers.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
```

```
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_MARSHALLERS_H_
#define __PO_HI_MARSHALLERS_H_

#include <deployment.h>
#include <request.h>
#include <po_hi_messages.h>

/*
 * Basic marshallers functions. These functions are then
 * reused in the generated code to marshall application data.
 */

void __po_hi_marshall_port (__po_hi_port_t value, __po_hi_msg_t* msg);
void __po_hi_unmarshall_port (__po_hi_port_t* value, __po_hi_msg_t* msg);

void __po_hi_marshall_array (void* value, __po_hi_msg_t* msg,__po_hi_uint32_t size, __po_hi_uint32_t* offs
void __po_hi_unmarshall_array (void* value, __po_hi_msg_t* msg,__po_hi_uint32_t size, __po_hi_uint32_t* o

void __po_hi_marshall_bool (__po_hi_bool_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_bool (__po_hi_bool_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_char (char value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_char (char* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_int (int value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_int (int* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_float (float value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_float (float* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_float32 (__po_hi_float32_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_float32 (__po_hi_float32_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_float64 (__po_hi_float64_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_float64 (__po_hi_float64_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_int8 (__po_hi_int8_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_int8 (__po_hi_int8_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_int16 (__po_hi_int16_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_int16 (__po_hi_int16_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_int32 (__po_hi_int32_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_int32 (__po_hi_int32_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_int64 (__po_hi_int64_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_int64 (__po_hi_int64_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_uint8 (__po_hi_uint8_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_uint8 (__po_hi_uint8_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_uint16 (__po_hi_uint16_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_uint16 (__po_hi_uint16_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_uint32 (__po_hi_uint32_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
```

```
void __po_hi_unmarshall_uint32 (__po_hi_uint32_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);

void __po_hi_marshall_uint64 (__po_hi_uint64_t value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
void __po_hi_unmarshall_uint64 (__po_hi_uint64_t* value, __po_hi_msg_t* msg,__po_hi_uint32_t* offset);
#endif /* __PO_HI_MARSHALLERS_H_ */
```

C.4 po_hi_giop.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifdef __PO_HI_USE_GIOP

#ifndef __PO_HI_GIOP_H__
#define __PO_HI_GIOP_H__

#include <po_hi_types.h>
#include <po_hi_messages.h>
#include <deployment.h>
#include <string.h>

/*
 * This file defines the structures and functions to support the GIOP
 * protocol. The supported verrsion of GIOP is the 1.3.
 *
 * This implementation was made according to the CORBA 3.1, Part 2,
 * chapter 9 specifications.
 */

#define __PO_HI_GIOP_MSGTYPE_REQUEST           0
#define __PO_HI_GIOP_MSGTYPE_REPLY             1
#define __PO_HI_GIOP_MSGTYPE_CANCELREQUEST     2
#define __PO_HI_GIOP_MSGTYPE_LOCATEREQUEST     3
#define __PO_HI_GIOP_MSGTYPE_LOCATEREPLY       4
#define __PO_HI_GIOP_MSGTYPE_CLOSECONNECTION   5
#define __PO_HI_GIOP_MSGTYPE_MESSAGEERROR      6
#define __PO_HI_GIOP_MSGTYPE_FRAGMENT          7

#define __PO_HI_GIOP_VERSION_MAJOR             1
#define __PO_HI_GIOP_VERSION_MINOR             3

#define __PO_HI_GIOP_MAGIC                     "GIOP"

#define __PO_HI_GIOP_OPERATION_MAX_SIZE        100

#define __PO_HI_GIOP_MAGIC_SIZE                4

#define __PO_HI_GIOP_DISPOSITION_KEY           0
#define __PO_HI_GIOP_DISPOSITION_PROFILE       1
```

```
#define __PO_HI_GIOP_DISPOSITION_REFERENCE      2

#define __PO_HI_GIOP_HAS_MORE_MESSAGES          300

typedef struct
{
  char               magic[4];
  struct {
    __po_hi_uint8_t major; /* __PO_HI_GIOP_VERSION_MAJOR */
    __po_hi_uint8_t minor; /* __PO_HI_GIOP_VERSION_MINOR */
  } version;
  __po_hi_uint8_t   flags;
  __po_hi_uint8_t   message_type;
  __po_hi_uint32_t  message_size;
}__po_hi_giop_msg_hdr_t;

/* The __po_hi_giop_msg_hdr_t gives a structure to fill a message header */
/* for the GIOP protocol. The flags (8 bits) are organized like this :   */
/*  0  0  0  0  0  0  0  0                                               */
/*                    |  |----- byte order : 1 for little-endian         */
/*                    |                     0 for big-endian             */
/* The message_type should be a value of __PO_HI_GIOP_MSGTYPE*           */

typedef struct
{
  __po_hi_uint16_t               disposition;
  union
  {
    struct
    {
      __po_hi_uint32_t               object_size;
      __po_hi_uint32_t               object_addr;
    }key;

    struct
    {
      __po_hi_uint32_t               profile_id;
      __po_hi_uint32_t               profile_length;
      __po_hi_uint32_t               profile_data;
    }profile;

    struct
    {
      __po_hi_uint32_t               profile_index;
      __po_hi_uint32_t               full_ior;
    }reference;
  } values;
}__po_hi_giop_request_target_t;
/*
 * Note: for now, we only support object target. The object-id will
 * always be set to 0.
 */

typedef struct
{
        __po_hi_uint32_t               request_id;
        __po_hi_uint8_t                response_flags;
        __po_hi_uint8_t                reserved[3];
```

```
        __po_hi_giop_request_target_t  target;
        __po_hi_uint32_t               operation_length;
        char                           operation[__PO_HI_GIOP_OPERATION_MAX_SIZE];
        __po_hi_uint32_t               nb_scontext;
}__po_hi_giop_request_hdr_t;
/*
 * The operation is set the a maximum length of 100
 */

void __po_hi_giop_msg_hdr_init (__po_hi_giop_msg_hdr_t* msg_hdr);
/*
 * Initialize the message header, set the magic number, the version
 * and all other needed variables
 */

void __po_hi_giop_msg_hdr_set_message_type (__po_hi_giop_msg_hdr_t* msg_hdr,
                                            __po_hi_uint8_t msg_type);
/*
 * Set the message type in the header. We only support request, so the
 * type should be always __PO_HI_GIOP_MSG_REQUEST
 */

void __po_hi_giop_msg_hdr_set_message_size (__po_hi_giop_msg_hdr_t* msg_hdr,
                                            __po_hi_uint32_t msg_size);

void __po_hi_giop_request_hdr_init (__po_hi_giop_request_hdr_t* request_hdr);

void __po_hi_giop_request_hdr_set_operation (__po_hi_giop_request_hdr_t* request_hdr,
                                             const char* request_name);

int __po_hi_giop_send (__po_hi_entity_t from,
                       __po_hi_entity_t to,
                       __po_hi_msg_t*   msg);

int __po_hi_giop_decode_msg (__po_hi_msg_t* network_flow,
                             __po_hi_msg_t* output_msg,
                             __po_hi_uint32_t* has_more);

#ifdef __PO_HI_DEBUG
void __po_hi_giop_print_msg( __po_hi_msg_t* msg);
#endif

#endif /* __PO_HI_GIOP_H__ */

#endif /* __PO_HI_USE_GIOP */
```

C.5 po_hi_messages.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
```

```
 */

#ifndef __PO_HI_MESSAGES_H_
#define __PO_HI_MESSAGES_H_

#include <po_hi_config.h>
#include <po_hi_types.h>

#include <request.h>
/* This file may not be generated. However, using messages implies
   using request. */

#ifdef __PO_HI_USE_GIOP
#define __PO_HI_MESSAGES_MAX_SIZE (int) sizeof(__po_hi_request_t) + 200
#else
#define __PO_HI_MESSAGES_MAX_SIZE (int) sizeof(__po_hi_request_t) + 4
/* XXX Why + 4 ? to be investigated */
#endif

#define __PO_HI_MESSAGES_CONTENT_BIGENDIAN      1
#define __PO_HI_MESSAGES_CONTENT_LITTLEENDIAN   2

typedef struct
{
  __po_hi_uint8_t   content[__PO_HI_MESSAGES_MAX_SIZE]; /* Content of the message */
  __po_hi_uint32_t  length;
  __po_hi_uint8_t   flags;
} __po_hi_msg_t;


void __po_hi_msg_reallocate (__po_hi_msg_t* message);
/*
 * Reset the message given in parameter
 */

void __po_hi_msg_write (__po_hi_msg_t*  msg,
void*           data,
__po_hi_uint32_t len);
/*
 * Write the data at the beginning of the specified message.  Length
 * of the data are specified by the parameter len
 */

void __po_hi_msg_read (__po_hi_msg_t*  msg,
       void*           data,
       __po_hi_uint32_t len);
/*
 * Read the data in the specified message. The data are taken from the
 * message and copied into the variable data.  Length of the data are
 * specified by the parameter len
 */

int __po_hi_msg_length (__po_hi_msg_t* msg);
/*
 * Return the length is the message
 */

void __po_hi_msg_copy (__po_hi_msg_t* dest,
```

```
       __po_hi_msg_t* src);
/*
 * Copy a message. The first argument is the message destination
 * whereas the second argument is the message source
 */

void __po_hi_msg_append_data (__po_hi_msg_t* msg, void* data, __po_hi_uint32_t length);
/*
 * Append data to a message. The first argument is the message which
 * will contain all the data. The second argument is a pointer to the
 * data and the third argument (length) is the size of the data in
 * byte.
 */

void __po_hi_msg_append_msg (__po_hi_msg_t* dest, __po_hi_msg_t* source);
/*
 * Append a message to another message. The first argument is the
 * message in which we will append the data. The second argument is
 * the source of the data.
 */

void __po_hi_msg_get_data (void* dest, __po_hi_msg_t* source, __po_hi_uint32_t index, __po_hi_uint32_t si
/*
 * Get data from a message at index 'index', and copy it to the dest
 * argument It will copy size bytes from the messages.
 */

void __po_hi_msg_move (__po_hi_msg_t* msg, __po_hi_uint32_t length);
/*
 * Move a part of the message to the beginning. This function will put
 * the part (starting from the length argument) to the beginning of
 * the message.
 */

#ifdef __PO_HI_USE_GIOP
int __po_hi_msg_should_swap (__po_hi_msg_t* msg);
/*
 * The __po_hi_msg_should_swap return 1 if the endianness of the
 * current processor differs with the endianness of the message. Else,
 * it returns 0.
 */

void __po_hi_msg_swap_value (void* from, void* dest, __po_hi_uint8_t size);
/*
 * The function __po_hi_msg_swap_value swap the bytes of the from
 * value and put it to the dest argument. The size of the value is
 * designed by the third argument.
 */
#endif /* __PO_HI_USE_GIOP */

#ifdef __PO_HI_DEBUG
void __po_hi_messages_debug (__po_hi_msg_t* msg);
#endif

#endif /* __PO_HI_MESSAGES_H_ */
```

C.6 po_hi_protocols.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_PROTOCOLS__
#define __PO_HI_PROTOCOLS__

#include <po_hi_messages.h>
#include <po_hi_types.h>

#include <deployment.h>
#include <request.h>

#define __PO_HI_NOPORT 1
#define __PO_HI_NOADDR ""

typedef __po_hi_uint16_t __po_hi_inetport_t;
typedef char*            __po_hi_inetaddr_t;

int __po_hi_protocols_send (__po_hi_entity_t from,
    __po_hi_entity_t to,
    __po_hi_msg_t* msg);
/*
 * Send a message to a specified entity.  The "from" argument is the
 * node which send the message. The argument "to" is used to designate
 * the entity which receive the message. Finally, the last argument
 * (msg) is the message
 */

int __po_hi_protocols_receive (__po_hi_entity_t from,
       __po_hi_msg_t* msg);
/*
 * Receive a message from a specified entity The entity which sent the
 * message is specified by the first argument.  The second argument
 * will contains the received message.
 */

int __po_hi_protocols_nonblocking_receive (__po_hi_entity_t from,
   __po_hi_msg_t* msg);
/*
 * Receive a message from a specified entity The entity which sent the
 * message is specified by the first argument.  The second argument
 * will contains the received message.  Receive 1 if data was received
 */

#endif /* __PO_HI_PROTOCOLS__ */
```

C.7 po_hi_transport.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2008, GET-Telecom Paris.
 */

#ifndef __PO_HI_TRANSPORT__
#define __PO_HI_TRANSPORT__

#include <po_hi_messages.h>
#include <deployment.h>
#include <request.h>

typedef uint8_t __po_hi_queue_id;

int __po_hi_transport_receive (__po_hi_entity_t from,
       __po_hi_msg_t* msg);
/*
 * Receive data from a node. The argument designated the sender of the
 * data. The second argument (msg) is the message which will receive
 * the data. If no message has been received, the function will block
 * the thread.
 */

int __po_hi_transport_nonblocking_receive (__po_hi_entity_t from,
   __po_hi_msg_t* msg);
/* Try to receive data from the node designed by the first
   argument. The data are stored in the second argument.  Returns
   __PO_HI_RECEIVE_SUCCESS if it receives data.  Else, it returns
   __PO_HI_RECEIVE_ERROR if no data are available
*/

void __po_hi_initialize_transport ();
/*
 * Initialize the transport layer (create and initialize
 * variables, ...)
 */

int __po_hi_transport_send (__po_hi_entity_t from,
    __po_hi_entity_t to,
    __po_hi_msg_t* msg);
/*
 * Send a message to a specified entity.  The "from" argument is the
 * node which send the message. The argument "to" is used to designate
 * the entity which receive the message. Finally, the last argument
 * (msg) is the message
 */

void __po_hi_initialize_transport_low_level ();
/*
 * Initialize low-level transport driver. It creates all structures
 * and variables required.
```

```
 */

int __po_hi_transport_low_level_send (__po_hi_entity_t from,
      __po_hi_entity_t to,
      __po_hi_msg_t* msg);
/*
 * Send the data through the low-level driver. The first argument is
 * the node which will receive the data. Argument msg is the message
 * which is sent.
 */

#endif /* __PO_HI_TRANSPORT__ */
```

C.8 po_hi_protected.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_PROTECTED_H__
#define __PO_HI_PROTECTED_H__
#include <stdint.h>
#include <deployment.h>

typedef uint8_t __po_hi_protected_t;

int __po_hi_protected_lock (__po_hi_protected_t protected_id);
/*
 * Lock the variable which has he id given by the argument.

 * Return __PO_HI_SUCCESS if it is successfull.  If there is an error,
 * it can return __PO_HI_ERROR_PTHREAD_MUTEX value
 */

int __po_hi_protected_unlock (__po_hi_protected_t protected_id);
/*
 * Unlock the variable which has he id given
 * by the argument.
 * Return __PO_HI_SUCCESS if it is successfull.
 * If there is an error, it can return
 * __PO_HI_ERROR_PTHREAD_MUTEX value
 */

int __po_hi_protected_init ();
/*
 * Initialize all variables to handle protected
 * objects in PolyORB-HI-C
 */

#endif /*  __PO_HI_PROTECTED_H__ */
```

C.9 po_hi_gqueue.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_GQUEUE_H__
#define __PO_HI_GQUEUE_H__

#define __PO_HI_GQUEUE_FULL      10

#define __PO_HI_GQUEUE_FIFO_INDATA    -1
#define __PO_HI_GQUEUE_FIFO_OUT       -2

#define __PO_HI_GQUEUE_INVALID_PORT invalid_port_t
#define __PO_HI_GQUEUE_INVALID_LOCAL_PORT invalid_local_port_t

#include <deployment.h>
#include <request.h>
#include <po_hi_types.h>

void __po_hi_gqueue_init (__po_hi_task_id       id,
  __po_hi_uint8_t       nb_ports,
  __po_hi_port_t        queue[],
  __po_hi_int8_t        sizes[],
  __po_hi_uint8_t       first[],
  __po_hi_uint8_t       offsets[],
  __po_hi_uint8_t       woffsets[],
  __po_hi_uint8_t       n_dest[],
  __po_hi_port_t*       destinations[],
  __po_hi_uint8_t       used_size[],
  __po_hi_local_port_t  history[],
  __po_hi_request_t     recent[],
  __po_hi_uint8_t       empties[],
  __po_hi_uint16_t      total_fifo_size);
/*
 * Initialize a global queue. In a distributed system, each task has
 * its own global queue. This function is invoked by each thead to
 * create its global queue, according to its information (number of
 * ports, destination of each port ...).
 *
 * The first argument is the task-id in the distributed system. The
 * second argument is the number of ports for the task. The argument
 * sizes contains the size of the FIFO for each port. The offsets
 * argument contains the offset position for each queue in the global
 * queue.  The n_dest argument correspond to the number of
 * destinations for an OUT port. The argument destinations tells what
 * are the ports connected to an OUT port.  Finally, the argument
 * total_fifo_size gives the total size of the global queue
```

```
 */

void __po_hi_gqueue_store_out (__po_hi_task_id id,
                               __po_hi_local_port_t port,
                               __po_hi_request_t* request);
/* Store a value for an OUT port.
 *
 * The id argument correspond to the task-id which own the global
 * queue. The second argument is the port that store the value. The
 * last argument is the request to store in the queue.
 */

int __po_hi_gqueue_send_output (__po_hi_task_id id,
                                 __po_hi_port_t port);
/*
 * Send a value for an out port.
 *
 * The first argument is the id of the task which have the global
 * queue. The second argument is the number of port that will send the
 * data
 */


int __po_hi_gqueue_get_value(__po_hi_task_id id,
     __po_hi_local_port_t port,
     __po_hi_request_t* request);
/*
 * Get the value on the specified port.
 *
 * The id parameter corresponds to the task-id in the local
 * process. The port argument is the number of the port that received
 * the data. The request argument is a pointer to store the received
 * data. If the port is an output, this function will return nothing,
 * but will not produce an error.
 */

int __po_hi_gqueue_next_value(__po_hi_task_id id,
      __po_hi_local_port_t port);
/*
 * Dequeue the value on a port. The argument id is the task identifier
 * in the local process. The second argument is the port number for
 * the thread. This function should not be called several times, until
 * you know what you do.
 */

int __po_hi_gqueue_get_count(__po_hi_task_id id,
     __po_hi_local_port_t port);
/*
 * Return the number of events that are pending of a port. The first
 * argument is the task identifier in the local process. The second
 * argument is the port identifier (or port number) for the thread.
 */

void __po_hi_gqueue_wait_for_incoming_event(__po_hi_task_id id,
    __po_hi_local_port_t* port);
/*
 * Wait until an event is received on any port for a given thread. The
 * first argument is the thread identifier in the local process. The
```

```
 * second argument is a pointer to a port value. When the function
 * returns, the port argument will contrain the port-id that received
 * the event.
 */

__po_hi_uint8_t __po_hi_gqueue_store_in (__po_hi_task_id id,
 __po_hi_local_port_t port,
 __po_hi_request_t* request);
/*
 * Store a value in a IN port. The first argument is the task
 * identifier in the local process. The second argument is the port
 * identifier for the local thread. The request argument contrains the
 * request that will be stored in the queue.
 */

#endif /* __PO_HI_GQUEUE_H__ */
```

C.10 po_hi_types.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_TYPES_H_
#define __PO_HI_TYPES_H_

#include"po_hi_config.h"

#ifdef HAVE_STDINT_H
#include <stdint.h>
#endif

#ifdef HAVE_STDBOOL_H
#include <stdbool.h>
#endif

#define __PO_HI_UNUSED_NODE -1

/*
 * Types are configured according to the ones available
 * on the target host.
 */


#ifdef HAVE_STDBOOL_H
typedef bool __po_hi_bool_t;
#else
#error This configuration is not supported
#endif
```

```
typedef float  __po_hi_float32_t;
typedef double __po_hi_float64_t;

#ifdef HAVE_STDINT_H
  typedef int8_t     __po_hi_int8_t;
  typedef int16_t    __po_hi_int16_t;
  typedef int32_t    __po_hi_int32_t;
  typedef int64_t    __po_hi_int64_t;
  typedef uint8_t    __po_hi_uint8_t;
  typedef uint16_t   __po_hi_uint16_t;
  typedef uint32_t   __po_hi_uint32_t;
  typedef uint64_t   __po_hi_uint64_t;
#else

/*
 * Most modern compilers have stdint.h header file.
 */

#error This configuration is not supported

  #if SIZEOF_INT == 4
  typedef int                    __po_hi_int32_t;
  #elif SIZEOF_LONG_INT == 4
  typedef long int               __po_hi_int32_t;
  #elif SIZEOF_SHORT_INT == 4
  typedef short int              __po_hi_int32_t;
  #endif

  #if SIZEOF_INT == 2
  typedef int                    __po_hi_int16_t;
  typedef unsigned int           __po_hi_uint16_t;
  #elif SIZEOF_SHORT_INT == 2
  typedef short int              __po_hi_int16_t;
  typedef unsigned short int     __po_hi_uint16_t;
  #elif SIZEOF_LONG_INT == 2
  typedef long int               __po_hi_int16_t;
  typedef unsigned long int      __po_hi_uint16_t;
  #endif

  #if SIZEOF_CHAR == 1
    typedef char                 __po_hi_int8_t;
    typedef unsigned char        __po_hi_uint8_t;
  #endif
#endif

void __po_hi_copy_array (void* dst, void* src, __po_hi_uint16_t size);

#endif /* __PO_HI_TYPES_H_ */
```

C.11 po_hi_returns.h

```
/*
 * This is a part of PolyORB-HI-C distribution, a minimal
 * middleware written for generated code from AADL models.
 * You should use it with the Ocarina toolsuite.
 *
```

```
 * For more informations, please visit http://ocarina.enst.fr
 *
 * Copyright (C) 2007-2009, GET-Telecom Paris.
 */

#ifndef __PO_HI_RETURNS_H__
#define __PO_HI_RETURNS_H__

/* Success return code */
#define __PO_HI_SUCCESS                     1

/* Errors from the API */
#define __PO_HI_ERROR_CREATE_TASK         -10
#define __PO_HI_ERROR_CLOCK               -15
#define __PO_HI_ERROR_QUEUE_FULL          -20

/* Errors related to the pthread library */
#define __PO_HI_ERROR_PTHREAD_COND        -50
#define __PO_HI_ERROR_PTHREAD_MUTEX       -51
#define __PO_HI_ERROR_PTHREAD_CREATE      -52
#define __PO_HI_ERROR_PTHREAD_ATTR        -53
#define __PO_HI_ERROR_PTHREAD_SCHED       -54
#define __PO_HI_ERROR_TRANSPORT_SEND      -55
#define __PO_HI_ERROR_PTHREAD_BARRIER     -56

/* GIOP error code */
#define __PO_HI_GIOP_INVALID_SIZE         -100
#define __PO_HI_GIOP_INVALID_VERSION      -120
#define __PO_HI_GIOP_INVALID_REQUEST_TYPE -150
#define __PO_HI_GIOP_INVALID_OPERATION    -180
#define __PO_HI_GIOP_UNSUPPORTED          -200

#endif /* __RETURNS_H__ */
```

Appendix D Porting PolyORB-HI-C to another architecture

This section gives some hints to help the developer to port PolyORB-HI-C. We will give the name of the files you need to change and what part of them should be modified to support new architectures and operating systems.

D.1 POSIX compliance

PolyORB-HI-C is POSIX compliant. It means that all the functions used in the framework should be available if your operating system is POSIX compliant. Even this compliance, you will probably need to make some changes and for each new architecture or operating system, you have to create a specific Makefile, as described in the thrid section.

D.2 Architecture-dependent files

If you need to port the framework on another architecture, some files don't need to be changed. Porting efforts will be focused on the following files :

- `po_hi_task.c` : Create tasks and handle their properties (period, priority, ...). At this time, this file contains only POSIX calls to create and manage thread.
- `po_hi_time.c` : Handle time and provide some functions to wait until an absolute time. The functions defined in this file make calls to POSIX functions like `clock_gettime()`.
- `po_hi_transport_sockets.c` : All the functions defined in this file are used to send or receive data through sockets. It creates a task to receive data and put them on a stack. All the functions made calls to POSIX-compliant functions like `socket()`, `listen` or `bind`.

D.3 Declare a new supported system

In all files that contain architecture-dependent, we need to split code for each system. We make it with MACCRO and include the code for the used architecture when we compile it. At this time, two systems are supported : `POSIX` and `RTEMS_POSIX`. Even if RTEMS use the POSIX implementation, there are some differences between that needs to declare another system.

If you want to support a new architecture, you need to declare a new maccro which will be used to differenciate the code for your architecture from other parts of the code. Then, this maccro will be included in the `CFLAGS` variable in the Makefile created for your architecture.

D.4 Define the compilation process

Each architecture has its own Makefile. It is used to define the compiler name, the linker name and some maccros to compile the code with a specific architecture.

All Makefiles are stored in the `share/make` directory. Each Makefile follow the following naming rule : `Makefile.arch.ostype`. For example, the Makefile created for the LEON architecture with the RTEMS OS has the name `Makefile.leon.rtems`. You need to create a Makefile with name that follow the naming rule and fill it with the rights `CFLAGS` and `CC` for the sytem you port.

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. [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