# PolyORB AADL personality User's Guide

Version 1.0w
Date: 16 May 2011

**Jérôme Hugues, Thomas Vergnaud, Bechir Zalila**

# Table of Contents

# About This Guide

This guide describes the use of the AADL personality for PolyORB, a schizophrenic middleware.

## What This Guide Contains

This guide contains the following chapters:

- Appendix A [Ada Mapping Rules], page 2 details the mapping rules used by Ocarina to generate Ada code from AADL models
- Appendix B [GNU Free Documentation License], page 28 contains the text of the license under which this document is being distributed.

## Conventions

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

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

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

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

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

# Appendix A AADL to Ada Mapping Rules

ARAO is an AADL runtime built on top of the PolyORB middleware. It provides a smooth integration of AADL concepts on top of a generic middleware, providing many configuration capabilities to the model developer.

We choose to use a middleware in order to ensure the communication between the nodes of the distributed application is the schizophrenic middleware PolyORB. It's obvious that a large part of the code (thread creation for example) is the same for distributed application. This code is written once and used as the middleware API. The use of PolyORB implies that communications between the application node are performed by requests and rely on an ORB (Object Request Broker). A full description of the ARAO API is given in Section A.4 [Description of the ARAO API], page 25.

The present chapter defines the mapping Ocarina uses to generate Ada code and PolyORB primitives.

These rules are triggered when the `PlyORB-QoS-Ada` code generator is selected.

## A.1 Components mapping rules

The unnamed namespace of an AADL description is mapped to a conventional Ada package called `Namespaces`. The several namespaces (which are children of the unnamed namespace) are mapped to subpackages of the `Namespaces` package. For each node of the distributed application, a `Namespaces` package "family" is generated; it contains all the data and subprograms mappings that are used by this node.

### A.1.1 Data components mapping

#### A.1.1.1 Base type mapping

A subcomponent-free `data` component should contain a `ARAO::data_type` property in order to generate the corresponding Ada type. ARAO predefined types are : `integer`, `float`, `null`, `string` and `boolean`. Normal data components are mapped to the related Ada type, as seen in the following AADL example :

```
data message
properties
  ARAO::data_type => integer;
end message;
```

is mapped to the following Ada95 code:

```
type message is new Integer;
```

#### A.1.1.2 Composed type mapping

AADL `data` component implementations may contain others `data` subcomponents. In this case, the `data` component is mapped to an Ada record type.

As an example, the following AADL component implementation:

```
data integer_type
properties
  ARAO::data_type => integer;
end integer_type;

data structure
end structure;
```

```
data implementation structure.impl
subcomponents
  d1 : data integer_type;
  d2 : data integer_type;
end structure.impl;
```

is mapped to the following Ada code :

```
type Integer_Type is new Integer;

type Structure_Impl is
   record
  D1 : Integer_Type;
  D2 : Integer_Type;
   end record;
```

### A.1.1.3 Protected type mapping

AADL protected `data` components must be declared in the same way composed types are, i.e. by encapsulating them within another AADL type declaration.

For each protected `data` components, a new type is declared which contains all data components (i.e. fields of the related composed type) plus a mutex object within a Ada record. As for composed types, all fields must be either a previously user-declared type or a ARAO base type. Accessors and building features for the type will be declared too, as for the data-owned procedures ("methods") designated in the `features` part of the `data` component.

The generated code enforces access protection, and declare type's object-oriented procedures ("methods", as defined by user), using the middleware mutexes. A "method" of a type must always has as `features` a `requires data access` on this data.

For example, the following AADL declaration :

```
data internal_message
properties
  ARAO::data_type => integer;
end internal_message;

data message
subcomponents
  Field : data internal_message;
features
  method : subprogram update;
properties
  ARAO::Access_Control_Protocol => Protected_Access;
end message;
```

would generate a code like this :

```
package Partition is

   type Message is private;

   procedure Build
     (This : out Message);

   procedure Get_Data
     (This  : in Message;
      Value : out Internal_Message);

   procedure Set_Data
     (This  : in out Message;
```

```
      Value : in Internal_Message);

private

   type Message is
      record
         Data  : Partition.Internal_Message;
         Mutex : PolyORB.Tasking.Mutexes.Mutex_Access;
      end record;

end Partition;
```

and the `update` method which will be call by generated code is like this :

```
procedure Update
  (This  : in out Message;
   Value : in Partition.Internal_Message) is
begin
   PolyORB.Tasking.Mutexes.Enter (This.Mutex);
   Repository.Update (This, Value);
   PolyORB.Tasking.Mutexes.Leave (This.Mutex);
end Update;
```

Where the procedures `Enter` and `Leave` are middleware mutexes' `take` and `release` procedures.

We define *protected data type internal type* as the components (usually only one) , excluding the mutex, which are embedded in a protected type as a `subcomponent`. The *protected data type internal type* could be either a protected type or a "normal" (non-protected) type. Eventually, all protected types can be decomposed in a set of basic types.

### A.1.1.4 Accessor usage

Data accessors can be used by the user exactly as data-owned procedure are. In the current version, they are the only ones actually called by the PolyORB AADL runtime, contrary to the generated interfaces which are not called at all.

Protected type accessors include `Set_X` and `Get_X` Ada procedures, where `X` is the name of the Field which contains the real (internal) data type. Those procedures are access-protected, using the protected object's middleware mutex to ensure mutual exclusion. The `Build` procedure will ensure mutex initialization.

An example of safe usage of accessors is :

```
-----------------------
-- Concurrent_Update --
-----------------------

procedure Concurrent_Update (arg : in out Partition.Counter_Type)
is
   Sum  : Partition.Integer_Type;
begin
   --  data initialization
   Partition.Set_Field (Data, 0);

   for I in 0 .. 100 loop
      Partition.Increment (Data);
      Partition.Get_Field (Data, Sum);
      if Integer (Sum) = 100 then
         exit;
      end if;
   end loop;
```

```
end Concurrent_Update;
```

relying on the following AADL declarations :

```
data Integer_Type
properties
  ARAO::data_type => integer;
end Integer_Type;

data Counter_Type
features
  Increment : subprogram Increment;
subcomponents
  field : data Integer_Type;
properties
  Concurency_Control_Protocol => Protected_Access;
end Counter_Type;

subprogram Increment
features
  this : requires data access Counter_Type;
properties
  source_language => Ada95;
  source_name => "Repository";
end Increment;

subprogram Concurrent_Update
features
  arg : requires data access Counter_Type;
properties
  source_language => Ada95;
  source_name => "Repository";
end Concurrent_Update;

thread Task
features
  sh_data : requires data access Counter_Type;
properties
  Dispatch_Protocol => Periodic;
  Period => 1000 Ms;
end Task;

thread implementation Task.impl
calls {
  sp1 : subprogram Concurrent_Update;
};
connections
  Cnx_Th_dat : data access sh_data -> sp1.arg;
end Task.impl;

process implementation global.impl
subcomponents
  th1 : thread Task.impl;
  th2 : thread Task.impl;
  dat : data Counter_Type;
connections
  Cnx_1 : data access dat -> th1.sh_data;
  Cnx_2 : data access dat -> th2.sh_data;
end global.impl;
```

with the following package specification and body generated :

```
package Partition is
```

```
   type Integer_Type is new Integer;
   type Counter_Type is private;

   procedure Build
     (This : out Partition.Integer_Type);

   procedure Get_Field
     (This  : in Message;
      Value : out Partition.Integer_Type);

   procedure Set_Field
     (This  : in out Message;
      Value : in Partition.Integer_Type);

   procedure Increment
     (This : in out Counter_Type);

private
   type Counter_Type is
      record
         Field : Partition.Integer_Type;
         Mutex : PolyORB.Tasking.Mutexes.Mutex_Access;
      end record;

end Partition;



with Repository;

package body Counter_Type_PKG is

   -----------
   -- Build --
   -----------

   procedure Build
     (This : out Message)
   is
   begin
      --  Initialize the middleware's mutex
      PolyORB.Tasking.Mutexes.Create (T.Mutex);
   end Build;

   ---------------
   -- Get_Field --
   ---------------

   procedure Get_Field
     (This  : in Counter_Type;
      Value : out Partition.Integer_Type)
   is
   begin
      PolyORB.Tasking.Mutexes.Enter (T.Mutex);
      Value := This.Field;
      PolyORB.Tasking.Mutexes.Leave (T.Mutex);
   end Get_Field;

   ---------------
   -- Set_Field --
   ---------------

   procedure Set_Field
     (This  : in out Counter_Type;
      Value : in Partition.Integer_Type)
```

```
      is
      begin
         PolyORB.Tasking.Mutexes.Enter (T.Mutex);
         This.Field := Value;
         PolyORB.Tasking.Mutexes.Leave (T.Mutex);
      end Set_Field;

      ---------------
      -- Increment --
      ---------------

      procedure Increment
        (This : in out Counter_Type)
      is
      begin
         PolyORB.Tasking.Mutexes.Enter (This.Mutex);
         Repository.Increment (This.Field);
         PolyORB.Tasking.Mutexes.Leave (This.Mutex);
      end Increment;

   end Counter_Type_PKG;
```

Note that the `Set` usage could had been replaced by an `Initialization` method of Counter_Type, and that the `Get` could had been replaced by a `Test_Value` method.

### A.1.1.5 Middleware mapping

We have seen that in the translation phase, the AADL data components are mapped to Ada95 types. Since the communication between nodes is performed using the PolyORB tools, all data sent in a request must have the neutral type `PolyORB.Any.Any`. So, conversion functions from and to this neutral type must be generated. For a process named `proc` these conversion functions will be generated in the `proc_Helpers` package. Example:

```
data message
properties
  ARAO::data_type => integer;
end message;
```

is a definition for an integer type, the conversion routines generated in `proc_Helpers` are:

```
with Partition;
with PolyORB.Any;

package proc_helpers is
   --  TypeCode variable used to characterize an Any variable

   TC_message : PolyORB.Any.TypeCode.Object :=
      PolyORB.Any.TypeCode.TC_Alias;

   function From_Any (Item : in PolyORB.Any.Any) return Partition.message;

   function To_Any (Item : in Partition.message) return PolyORB.Any.Any;

end proc_helpers;
```

Note that we use the `Namespaces` package created in the translation phase.

### A.1.1.6 AADL Properties support

Available properties for data components can be found in SAE AS5506, in 5.1 page 50 and in Appendix A, pages 197-218.

| | |
|---|---|
| Concurency_control_protocol | Supported : None, Access_Protected |
| Not_Collocated | Not Supported |
| Provided_Access | Not Supported |
| Required_Access | Not Supported |
| Source_Code_Size | Not Supported |
| Source_Language | Not Supported |
| Source_Name | Not Supported |
| Source_Text | Not Supported |
| Type_Source_Name | Not Supported |

### A.1.2 Subprogram components mapping

AADL subprograms are mapped to Ada procedures. In case of data-owned subprograms, they are managed in the related generated package, as seen in Section A.1.1 [Data components mapping], page 2. The parameters of the procedure are mapped from the subprogram features with respect to the following rules:

- The parameter name is mapped from the parameter feature name
- The parameter type is mapped from the parameter feature data type as specified in Section A.1.1 [Data components mapping], page 2
- The parameter orientation is the same as the feature orientation ("in", "out" or "in out").

The body of the mapped procedure depend on the nature of the subprogram component. Subprogram components can be classified in many kind depending on the value of the `Source_Language`, `Source_Name` and `Source_Text` standard AADL properties and the existence or not of call sequences in the subprogram implementation. There are four kinds of subprogram components:

1. The empty subprograms.
2. The opaque subprograms.
3. The pure call sequence subprograms.
4. The hybrid subprograms.

#### A.1.2.1 Mapping of empty subprograms

Empty subprograms correspond to subprograms for which there is neither `Source_Language` nor `Source_Name` nor `Source_Text` values nor call sequences. Such kind of subprogram components has no particular utility. For example:

```
subprogram sp
features
  e : in parameter message;
  s : out parameter message;
end sp;
```

is en empty subprogram. A possible Ada implementation for this subprogram could be:

```
procedure sp (e : in message; s : out message) is
  NYI : exception;
begin
  raise NYI;
end sp;
```

#### A.1.2.2 Mapping of opaque subprograms

Opaque subprograms are the simplest "useful" subprogram components (in code generation point of view). For these subprograms, the `Source_Language` property indicates the program-

ming language of the implementation (C or Ada95). The `Source_Name` property indicates the name of the subprogram implementing the subprogram:

- for Ada95 subprograms, the value of the `Source_Name` property is the **fully qualified name** of the subprogram (e.g. `My_Package.My_Spg`). If the package is stored in a file named according to the GNAT Ada compiler conventions, there is no need to give a `Source_Text` property for Ada95 subprograms. Otherwise the `Source_Text` property is necessary for the compiler to fetch the implementation files.
- for C subprograms, the value of the `Source_Name` property is the **name** of the C subprogram implementing the AADL subprogram. The `Source_Text` is mandatory for this kind of subprogram and it must give one of the following information:
  - the path to the `.c` source file that contains the implementation of the subprogram.
  - the path to one or more precompiled object files (`.o`) that implement the AADL subprogram.
  - the path to one or more precompiled `C` library (`.a`) that implement the AADL subprogram.

  These information can be used together, for example may give the `C` source file that implements the AADL subprogram, an object file that contains entities used by the `C` file and a library that is necessary to the `C` sources or the objects.

In this case, the code generation consist of creating a shell for the implementation code. In the case of Ada subprograms, the generated subprogram renames the implementation subprogram (using the Ada95 renaming facility). Example:

```
subprogram sp
features
  e : in  parameter message;
  s : out parameter message;
end sp;

subprogram implementation sp.impl
properties
  Source_Language => Ada95;
  Source_Name     => "Repository.Sp_Impl";
end sp.impl;
```

The generated code for the `sp.impl` component is:

```
with Repository;
...
procedure sp_impl (e : in message; s : out message)
  renames Repository.Sp_Impl;
```

The code of the `Repository.sp_impl` procedure is provided by the architecture and must be conform with the `sp.impl` signature. The coherence between the two subprograms will be verified by the Ada95 compiler.

The fact that the hand-written code is not inserted in the generated shell allows this code to be written in a programming language other than Ada95. Thus, if the implementation code is C we have this situation:

```
subprogram sp
features
  e : in parameter message;
  s : out parameter message;
end sp;
```

```
subprogram implementation sp.impl
properties
  Source_Language => C;
  Source_Name     => "implem";
end sp.impl;
```

The `Source_Name` value is interpreted as the name of the C subprogram implementing the AADL subprogram. The generated code for the `sp.impl` component is:

```
procedure sp_impl (e : in message; s : out message);
pragma Import (C, sp_impl, "implem");
```

This approach will allow us to have a certain flexibility by separating the generated code and the hand-written code. We can modify the AADL description without affecting the hand-written code (the signature should not be modified of course).

### A.1.2.3 Mapping of pure call sequence subprograms

In addition to the opaque approach which consist of delegating all the subprogram body writing to the user, AADL allows to model subprogram as a pure call sequence to other subprograms. Example:

```
subprogram spA
features
  s : out parameter message;
end spA;

subprogram spB
features
  s : out parameter message;
end spB;

subprogram spC
features
  e : in  parameter message;
  s : out parameter message;
end spC;

subprogram spA.impl
calls {
  call1 : subprogram spB;
  call2 : subprogram spC;};
connections
  cnx1 : parameter call1.s -> call2.e;
  cnx2 : parameter call2.s -> s;
end spA.impl;
```

In this case, the subprogram connects together a number of other subprograms. In addition to the call sequence, the connections clause completes the description by specifying the connections between parameters. The pure sequence call model allows to generate complete code : the calls in the call sequence corresponds to Ada95 procedure calls and the connections between parameters correspond to eventual intermediary variables. The Ada95 code generated for the subprogram `spA.impl` is:

```
procedure spA_impl (s : out message) is
   cnx1 : message;
begin
   spB (cnx1);
   spC (cnx1, s);
end spA_impl;
```

Note that in case of pure call sequence subprograms, the AADL subprogram must contain only one call sequence. If there are more than one call sequence, it's impossible - in this case - to determine the relation between them.

### A.1.2.4 Mapping of hybrid subprograms

The two last kinds of subprogram components describe even an opaque implementation for which all the functional part is written by the user or a pure call sequence for which all the functional part is given by the AADL description. These two cases are relatively simple to implement. However, they don't offer much flexibility. In the general case we want to integrate the maximum of information within the AADL description in order to get an easy assembling of the distributed application components. However, AADL does not provide control structures (conditions, loops). The best way is to combine the opaque model and the pure call sequence model.

To illustrate the problem, let's consider the following example: A subprogram `spA` receives an input integer value . The subprogram behavior depends on the $a$ value:

- If $a < 4$, then $a$ is given to another subprogram `spB`;
- Else, `spA` calls a third subprogram called `spC` which give its return value to `spB`

In all cases, the return value of `spB` is given to a forth subprogram `spD`; the return value of `spD` is returned by `spA`.

The behavior of `spA` is illustrated by this algorithm:

```
if a < 4 then
  b <- spB (a)
else
  c <- spC ()
  b <- spB (c)
end if

d <- spD (b)
return d
```

We assume that the subprograms `spB`, `spC` and `spD` are correctly defined.

We have three call sequences. AADL allows only to describe the architectural aspects of the algorithm (the connections between the different subprograms). The AADL source corresponding to the last example is:

```
data int
properties
  GAIA::Data_Type => Integer;
end int;

subprogram spA
features
  a : in parameter int;
  d : out parameter int;
end spA;

subprogram spB
features
  e : in parameter int;
  s : out parameter int;
end spB;

subprogram spC
```

```
features
  s : out parameter int;
end spC;

subprogram spD
features
  e : in parameter int;
  s : out parameter int;
end spD;

subprogram implementation spA.impl
properties
  Source_Language => Ada95;
  Source_Name     => "Repository.SpA_Impl"
calls
  seq1 : {spB1 : subprogram spB;};
  seq2 : {spC2 : subprogram spC;
          spB2 : subprogram spB;};
  seq3 : {spD3 : subprogram spD;};
connections
  cnx1 : parameter a -> apB1.e;
  cnx2 : parameter spB1.s -> spD3.e;

  cnx3 : parameter spC2.s -> spB2.e;
  cnx4 : parameter spB2.s -> spD3.e;

  cnx5 : parameter spd3.s -> d;
end spA.impl;
```

The first remark is that the subprogram implementation contains at the same time the `Source_[Language|Name]` (and a possible `Source_Text`) properties and call sequences. The hand-written code describes the algorithm. This algorithm should be able to handle each call sequence as being a block and must be as simple as possible: the user should not know the content of the call sequence.

The generated code for each block (call sequence) is almost identical to the generated code for pure call sequence. For each block, a subprogram is generated. To make things simple for the user, these subprograms have the same signature (one parameter called Status):

```
type SpA_Impl_Status is record
  a, b, c, d : int;
end record;

procedure SpA_Seq1 (in out Status : spA_impl_Status) is
begin
   spB (Status.a, Status.b);
end SpA_Seq1;

procedure SpA_Seq2 (in out Status : spA_impl_Status) is
begin
   spC (Status.c);
   spB (Status.c, Status.b);
end SpA_Seq2;

procedure SpA_Seq3 (in out Status : spA_impl_Status) is
begin
   spD (Status.b, d);
end SpA_Seq3;
```

The generated code for the `spA.impl` subprogram is very simple:

```
procedure SpA_Impl (a : in int; d : out int) is
```

```
   Status : spA_impl_Status;
begin
   Status.a := a;
   Repository.SpA_Impl
     (Status,
      SpA_Seq1'Access,
      SpA_Seq2'Access,
      SpA_Seq3'Access);
   d := Status.d;
end SpA_Impl;
```

The subprogram which describes the algorithm and which should be written by the user is relatively simple, and does not require any knowledge of the call sequences contents:

```
type SpA_Impl_Call_Sequence is access
  procedure (in out Status : spA_impl_Status);

procedure SpA_Impl
  (Status : in out spA_impl_Status,
   seq1   : spA_impl_Call_Sequence,
   seq2   : spA_impl_Call_Sequence,
   seq3   : spA_impl_Call_Sequence)
is
begin
   if Status.a > 4 then
      seq1.all (Status);
   else
      seq2.all (Status);
   end if;
   seq3.all (Status);
end SpA_Impl;
```

### A.1.2.5 Data access

If a subprogram has a `requires access` feature to a data, this data is added to the parameters list, with the mode corresponding to data access rights (i.e. `read-only => in`, `write-only => out` and `read-write => in out`).

In the specific case of subprograms requiring protected data access, user should provides different data depending on subprograms' nature.

If the subprogram is a "method" of the protected object (i.e. if it appears in its `features` field), then the user should provides an implementation of the subprogram which take the subprogram access as the first parameter, with the mode chosen following the rule described above. The parameter's name must always be `this`. This parameter type must always be of the protected data type internal type (cf. Section A.1.1 [Data components mapping], page 2).

If the subprogram is a not "method" of the protected object, user work depends of the accessed data's `Actual_Lock_Implementation` property, which defines shared variables update policy. This policy could be either synchronous (`synchronous_lock`) or asynchronous (`asynchronous_lock`). Default is asynchronous update policy.

The user must write a subprogram implementation complying to the following rules :

- For each *asynchronous policy*-defined data accessed, add an parameter at beginning of the data's protected type.
- For each *synchronous policy*-defined data accessed, add an parameter at beginning of the subprogram's parameter list of the data's protected type internal type.

Note that accessed data (found in the subprogram component's `features` field) must always be parsed in the same order they are declared in the AADL specification. In any case, mode is still chosen accordingly to the rule describe above.

Note that only opaque subprograms currently support synchronous data update policy.

If synchronous policy is chosen for a data update policy, the user should be aware that access protection is ensured by the runtime code (cf. Section A.1.3 [Thread components mapping], page 15).

Here is an example of data-owned specification of a protected object :

```
data internal_data
properties
  ARAO::data_type => integer;
end internal_data;

data shared_data
features
  method : subprogram update;
properties
  Concurrency_Control_Protocol => Protected_Access;
  ARAO::Actual_Lock_Implementation => Synchronous_Lock;
end shared_data;

data implementation shared_data.i
subcomponents
  Field : data internal_data;
end shared_data.i;

--  subprograms

subprogram update
features
  this : requires data access shared_data.i;
properties
  source_language => Ada95;
  source_name => "Repository";
end update;
```

The user provides :

```
procedure Update (Field : in out Partition.Internal_Data;
                  I : in Partition.message);

------------
-- Update --
------------

procedure Update (Field : in out Partition.Internal_Data;
                  I : in Partition.message)
is
   use Partition;
begin
   Field := Partition.Internal_Data (Integer (Field) + Integer (I));
end Update;
```

And Ocarina will generate the following implementation for the access-protected subprogram :

```
------------
-- update --
------------

procedure Update
  (This : in out Partition.Shared_Data_I;
   I : Partition.Message)
```

```
is
begin
   PolyORB.Tasking.Mutexes.Enter
     (This.Mutex);
   Repository.Update
     (Field => This.Field,
      I => I);
   PolyORB.Tasking.Mutexes.Leave
     (This.Mutex);
end Update;
```

### A.1.2.6 AADL Properties support

Available properties for subprogram components can be found in SAE AS5506, in 5.2 page 56 and in Appendix A, pages 197-218.

| | |
|---|---|
| Actual_Memory_Binding | Not Supported |
| Actual_Subprogram_Call | Not Supported |
| Client_Subprogram_Execution_Time | Not Supported |
| Compute_Deadline | Not Supported |
| Compute_Execution_Time | Not Supported |
| Concurrency_Control_Protocol | Not Supported |
| Queue_Processing_Protocol | Not Supported |
| Queue_Size | Not Supported |
| Recover_Deadline | Not Supported |
| Recover_Execution_Time | Not Supported |
| Server_Subprogram_Call_Binding | Not Supported |
| Source_Code_Size | Not Supported |
| Source_Data_Size | Not Supported |
| Source_Heap_Size | Not Supported |
| Source_Stack_Size | Not Supported |
| Source_Language | Supported (Ada) |
| Source_Name | Supported |
| Source_Text | Supported |

## A.1.3 Thread components mapping

The mapping of thread components is a little bit more complicated than the mapping of data components. Threads are mapped to an Ada95 parameter-less procedure which executes the thread work (periodically or aperiodically depending on the thread nature). For each periodic thread, a middleware thread is created using the API described in Section A.4 [Description of the ARAO API], page 25. For example˜:

```
thread sender
features
  msg_out : out event data port message;
properties
  Dispatch_Protocol => Periodic;
  Period => 1000 Ms;
end sender;
```

### A.1.3.1 Servant mapping

If this thread belongs to a process `proc`, and if `th1` is the name of the thread subcomponent of `proc` having the type `sender`, then a package `proc_Servants` is created:

```
package proc_Servants is
```

```
   ...
   procedure th1_Ctrler;
   ...
end proc_Servants;
```

In the main subprogram `proc` we find:

```
Aadl_Periodic_Threads.Create_Periodic_Thread
  (TP => sn_Servants.th1_Ctrler'Access);
```

The thread "in" or "in out" ports are mapped in an Ada protected object which allows a protected access to these ports. For each port, a buffer having the port stack size is created, implemented with a cyclic array. Since these ports are the destination of other components requests, for each in port, a PolyORB Reference is created and for each thread containing in ports, a servant is created to handle the incoming requests; Example:

```
thread receiver
features
  msg_in : in event data port message;
end receiver;
```

If this thread belongs to a process `proc`, and if `th2` is the name of the thread subcomponent of `proc` having the type `receiver`, then the following declarations will be generated in the `proc_Servants` package spec:

```
with Partition;

with PolyORB.Components;
with PolyORB.Servants;
with PolyORB.References;

package proc_Servants is
   ...
   procedure th2_Ctrler;

   type th2_Object is new Servant with null record;

   th2_Ref : PolyORB.References.Ref;

   function Execute_Servant
     (Obj : access th2_Object;
      Msg : PolyORB.Components.Message'Class)
     return PolyORB.Components.Message'Class;

   type th2_msg_in_buf_type is array (1 .. 1) of Partition.message;

   protected th2_Ports is
      procedure Put_msg_in (msg_in : Partition.message);
      procedure Get_msg_in (msg_in : out Partition.message);
      procedure Push_Back_msg_in (msg_in : out Partition.message);
   private
      msg_in_Buf : Th2_Msg_In_Buf_Type.Table;
   end th2_Ports;
   ...
end proc_Servants;
```

For each "out" or "in out" port, we declare reference variable for each "in" or "in out" port connected to this port.

### A.1.3.2 Shared variables access

In order to comply to the AADL *input-processing-output* algorithm, shared data (either access-protected or not) are not read or written directly, but through temporary variables.

As seen in Section A.1.4 [Process components mapping], page 18, any thread can access shared variables. In order to ensure protected access when needed, Ocarina will declare a local variable in the `thread_controler` function, whose type is the variable internal type (if the variable has the protected access property) or the variable real type.

Each time the thread controller is activated (i.e. each time the related servant is called), the local variable is put to shared variable value by its `Setter` procedure, then processing is done using the proper user-defined procedure. Then the `Getter` is used to update the shared variable.

Note that both `Setter` and `Getter` procedures are generated by Ocarina and ensure access protection, as described in Section A.1.1 [Data components mapping], page 2.

Here is an example of generated code of the `thread_controler` procedure which manage a `mem_sh` variable.

```
procedure Th1_Controller is
   Msg_In : Partition.Message;
   Msg_In_Present : Standard.Boolean;
   Msg_Out : Partition.Message;

   -- local temporary variable definition
   Mem : Partition.Internal_Data;
begin
   --  Read shared data and store it in local variable
   Partition.Get_Field (Sh_Mem, Mem);

   --  Read in IN ports
   Tr_Servants.Th1_IN_Ports.Get_Msg_In
     (Msg_In,
      Msg_In_Present);
   if (True
       and then Msg_In_Present)
   then
      --  Processing local variable
      Repository.Transmit_Message
        (Msg_In => Msg_In,
         Msg_Out => Msg_Out,
         Mem => Mem);

      --  Write in OUT ports
      ARAO.Requests.Emit_Msg
        (Tr_Helpers.To_Any
         (Msg_Out),
         Tr_Th2_Ref,
         "msg_in");
   else
      if Msg_In_Present
      then
         Tr_Servants.Th1_IN_Ports.Push_Back_Msg_In (Msg_In);
      end if;
   end if;

   --  Write back local variable into shared data
   Partition.Set_Field (Sh_Mem, Mem);
end Th1_Controller;
```

#### A.1.3.3 AADL Properties support

Available properties for thread components can be found in SAE AS5506, in 5.3 page 61 and in Appendix A, pages 197-218.

| | |
|---|---|
| Activate_Deadline | Not Supported |
| Activate_Execution_Time | Not Supported |
| Activate_Entrypoint | Not Supported |
| Active_Thread_Handling_Protocol | Not Supported |
| Active_Thread_Queue_Handling_Protocol | Not Supported |
| Actual_Connection_Binding | Not Supported |
| Actual_Memory_Binding | Not Supported |
| Actual_Processor_Binding | Not Supported |
| Allowed_Connection_Protocol | Not Supported |
| Client_Subprogram_Execution_Time | Not Supported |
| Compute_Deadline | Not Supported |
| Compute_Execution_Time | Not Supported |
| Concurrency_Control_Protocol | Not Supported |
| Deactivate_Deadline | Not Supported |
| Deactivate_Execution_Time | Not Supported |
| Deactivate_Entrypoint | Not Supported |
| Deadline | Not Supported |
| Dispatch_Protocol | Supported (Periodic, Aperiodic) |
| Finalize_Deadline | Not Supported |
| Finalize_Execution_Time | Not Supported |
| Finalize_Entrypoint | Not Supported |
| Initialize_Deadline | Not Supported |
| Initialize_Execution_Time | Not Supported |
| Initialize_Entrypoint | Not Supported |
| Not_Collocated | Not Supported |
| Period | Supported |
| Queue_Size | Not Supported |
| Recover_Deadline | Not Supported |
| Recover_Execution_Time | Not Supported |
| Server_Subprogram_Call_Binding | Not Supported |
| Source_Code_Size | Not Supported |
| Source_Data_Size | Not Supported |
| Source_Heap_Size | Not Supported |
| Source_Stack_Size | Supported |
| Source_Name | Not Supported |
| Source_Text | Not Supported |
| Source_Language | Not Supported |
| Synchronized_Component | Not Supported |

### A.1.4 Process components mapping

The main component in this phase is the `process` component. The distributed application is a set of processes which communicate between each other. Each `process` is mapped to an Ada95 main subprogram which leads to an executable after being compiled.

#### A.1.4.1 Shared variables declaration and initialization

In the case where a `process` contains shared variables declaration (which should always refers to local `data` components, as Ocarina does not support variables shared amongst multiples process), a variable is declared in the '`proc_servant`' body package.

If the shared variable has a protected access property, Ocarina will also add a `initialize` procedure to the package, and set it as the package initialization procedure for the middleware, which will ensure that it is ran before any usage of the package. This procedure calls protected type's `Build` interface (cf. Section A.1.1 [Data components mapping], page 2), initializing middleware's mutexes.

Note that shared variables (either protected or not) are visible from any thread of the process. How those variables are accessed and updated is described in Section A.1.3 [Thread components mapping], page 15.

Here is a AADL specification for declaring a data shared between two threads, with protected access in a process:

```
-- protected data type declaration

data internal_data
properties
  ARAO::data_type => integer;
end internal_data;

data shared_data
properties
  Concurrency_Control_Protocol => Protected_Access;
end shared_data;

data implementation shared_data.i
subcomponents
  Field : data internal_data;
end shared_data.i;

-- Process declaration

process transmitter_node
features
  msg_in : in event data port message;
  msg_out : out event data port message;
end transmitter_node;

process implementation transmitter_node.complex
subcomponents
  th1 : thread transmitter.simple;
  th2 : thread transmitter.simple;
  sh_mem : data shared_data.i;
connections
  event data port msg_in -> th1.msg_in;
  event data port th1.msg_out -> th2.msg_in;
  event data port th2.msg_out -> msg_out;
  data access sh_mem -> th1.mem;
  data access sh_mem -> th2.mem;
end transmitter_node.complex;
```

and here is the related code generated by Ocarina :

```
package body Tr_Servants is

   -- Shared variable declaration

   Sh_Mem : Partition.Shared_Data_I;

   -- Initialization procedure declaration and description

   procedure Initialize;

   ----------------
```

```
   -- Initialize --
   ----------------

   procedure Initialize is
   begin
      Partition.Builder
        (Sh_Mem);
   end Initialize;

   --  Threads-related code
   --  (...)

   --  Bind initialization function with middleware initialization
begin
   declare
      use PolyORB.Utils.Strings;
      use PolyORB.Utils.Strings.Lists;
   begin
      PolyORB.Initialization.Register_Module
        (PolyORB.Initialization.Module_Info'
         (Name => + "tr_Servants",
          Conflicts => PolyORB.Utils.Strings.Lists.Empty,
          Depends => + "any",
          Provides => PolyORB.Utils.Strings.Lists.Empty,
          Implicit => False,
          Init => Initialize'Access,
          Shutdown => null));
   end;
end Tr_Servants;
```

### A.1.4.2 AADL Properties support

Available properties for process components can be found in SAE AS5506, in 5.5 page 77 and in Appendix A, pages 197-218.

| | |
|---|---|
| Active_Thread_Handling_Protocol | Not Supported |
| Active_Thread_Queue_Handling_Protocol | Not Supported |
| Actual_Connection_Binding | Not Supported |
| Actual_Memory_Binding | Not Supported |
| Actual_Processor_Binding | Supported |
| Allowed_Connection_Protocol | Not Supported |
| Deadline | Not Supported |
| Load_Deadline | Not Supported |
| Load_Time | Not Supported |
| Not_Collocated | Not Supported |
| Period | Not Supported |
| Runtime_Protection | Not Supported |
| Server_Subprogram_Call_Binding | Not Supported |
| Source_Code_Size | Not Supported |
| Source_Data_Size | Not Supported |
| Source_Stack_Size | Supported |
| Source_Name | Not Supported |
| Source_Text | Not Supported |
| Source_Language | Not Supported |
| Synchronized_Component | Not Supported |

## A.2 Setup of the application

In order for each executable to work correctly, the middleware must be properly set up. In the case of PolyORB, we used an API named ARAO (AADL Runtime API for Ocarina). the setup consists in two phases :

- adding `with` and `pragma` clauses to initialize the middleware parameters.
- build Portable Object Adapters for each in port.

The nature of these with clauses depends on these factors:

- The number of threads in the node
- The presence or not of periodic threads

The setup is done by including (`with`) static or generated packages. Those packages can be divided into three classes :

- Basic setup package, which are called by all process.
- Tasking package, which are either no_tasking (only one thread in the process) or full_tasking (more than one thread in the process).
- Object Adapter setup package, which can be either static (if no priorities management has been set in AADL description) or generated.

Example:

```
process proc
features
  msg_in : in event data port message;
  msg_out : out event data port message;
end proc;

process implementation proc.simple
subcomponents
  th1 : thread sender.simple;
  th2 : thread receiver.simple;
connections
  event data port msg_in -> th2.msg_in;
  event data port th1.msg_out -> msg_out;
end proc.simple;
```

The process above contains more than one thread, so the Middleware need to be set up in a multitask mode. The execution of a particular node follows this order: first, it put the information concerning its ports in the middleware memory, then collects the information on the other processes (to which it is connected).

The code of the `proc` process is:

```
with PolyORB.Initialization;
with Sn_Servants;
with ARAO.Utils;
with ARAO.Periodic_Threads;
with ARAO.RT_Obj_Adapters;
with PolyORB.Setup;
with PolyORB.ORB;

--  Runtime configuration
with ARAO.Setup.Application;
pragma Warnings (Off, ARAO.Setup.Application);
pragma Elaborate_All (ARAO.Setup.Application);

--  Full tasking mode
with ARAO.Setup.Tasking.Full_Tasking;
```

```
pragma Warnings (Off, ARAO.Setup.Tasking.Full_Tasking);
pragma Elaborate_All (ARAO.Setup.Tasking.Full_Tasking);

with ARAO.Periodic_Threads;
with ARAO.RT_Obj_Adapters;

procedure proc is
   use proc_Servants;
begin
   PolyORB.Initialization.Initialize_World;

   --  Link local RT POA to current node, specifing priority

   ARAO.RT_Obj_Adapters.Link_To_Obj_Adapter
     (new proc_Servants.th2_Object,
      Th2_Ref,
      1);

   --  Collecting the references of the processes to which it's
   --  connected

   ARAO.Utils.Get_GIOP_Ref (tr1_th1_Ref, "127.0.0.1", 4000, 1, "th1", "iiop", 1);

   --  Create a periodic thread

   ARAO.Periodic_Threads.Create_Periodic_Thread
     (TP => proc_Servants.th1_Controller'Access);

   PolyORB.ORB.Run (PolyORB.Setup.The_ORB, May_Poll => True);
end proc;
```

And the code of the generated file `ARAO.Setup.Application` is:

```
with ARAO.Setup.Base;
pragma Warnings (Off, ARAO.Setup.Base);
pragma Elaborate_All (ARAO.Setup.Base);
with PolyORB.Setup.IIOP;
pragma Warnings (Off, PolyORB.Setup.IIOP);
pragma Elaborate_All (PolyORB.Setup.IIOP);
with PolyORB.Setup.Access_Points.IIOP;
pragma Warnings (Off, PolyORB.Setup.Access_Points.IIOP);
pragma Elaborate_All (PolyORB.Setup.Access_Points.IIOP);
--  ORB controller : workers
with PolyORB.ORB_Controller.Workers;
pragma Warnings (Off, PolyORB.ORB_Controller.Workers);
pragma Elaborate_All (PolyORB.ORB_Controller.Workers);
--  Multithreaded no priority mode package
with ARAO.Setup.Ocarina_OA;
pragma Warnings (Off, ARAO.Setup.Ocarina_OA);
pragma Elaborate_All (ARAO.Setup.Ocarina_OA);

package body ARAO.Setup.Application is

   --  No protocol set : default : GIOP/IIOP

   --  No request priority management

end ARAO.Setup.Application;
```

Note that, since no priorities has been set in AADL description, Object Adapter is a generic one.

If thread priorities have been set in AADL description, then ARAO will build a custom Portable Object Adapter. The building of Portable Object Adapter depends of a set of data such has receiver thread priority and stack size, and the number of out ports connected to his thread. A lane will be created for each port, which will contain thread for every connected out port. Lane priority and stack size will be inherited from AADL thread description, or set to default.

Let's modify the previous example by adding priorities to each threads.

```
process proc
features
  msg_in : in event data port message
  msg_out : out event data port message;
end proc;

process implementation proc.simple
subcomponents
  th1 : thread sender.simple {ARAO::Priority => 1};
  th2 : thread receiver.simple {ARAO::Priority => 32};
connections
  event data port msg_in -> th2.msg_in;
  event data port th1.msg_out -> msg_out;
end proc.simple;
```

Then Ocarina will generate another version of `ARAO.Setup.Application`, which will contain calls to a custom Object Adapter generator in `ARAO.Setup.OA.Multithreaded.Prio`.

```
--  General setup
with ARAO.Setup.Base;
pragma Warnings (Off, ARAO.Setup.Base);
pragma Elaborate_All (ARAO.Setup.Base);

--  Low-level setup packages
with PolyORB.Setup.IIOP;
pragma Warnings (Off, PolyORB.Setup.IIOP);
pragma Elaborate_All (PolyORB.Setup.IIOP);
with PolyORB.Setup.Access_Points.IIOP;
pragma Warnings (Off, PolyORB.Setup.Access_Points.IIOP);
pragma Elaborate_All (PolyORB.Setup.Access_Points.IIOP);

--  ORB controller : workers
with PolyORB.ORB_Controller.Workers;
pragma Warnings (Off, PolyORB.ORB_Controller.Workers);
pragma Elaborate_All (PolyORB.ORB_Controller.Workers);

--  Multithreaded mode package
with ARAO.Setup.OA.Multithreaded.Prio;
pragma Warnings (Off, ARAO.Setup.OA.Multithreaded.Prio);
pragma Elaborate_All (ARAO.Setup.OA.Multithreaded.Prio);

--  priorites-related packages
with PolyORB.Types;
with ARAO.Threads;
with PolyORB.Setup.OA.Basic_RT_Poa;
with ARAO.Setup.OA.Multithreaded;

--  Initialization-related packages
with PolyORB.Initialization;
with PolyORB.Utils.Strings;
with PolyORB.Utils.Strings.Lists;

package body ARAO.Setup.Application is
```

```
   --  No protocol set : default : GIOP/IIOP

   Threads_Array_ : constant ARAO.Threads.Threads_Properties_Array :=
     ((Standard.Natural
        (1),                --  thread th1 Priority
      Standard.Natural
        (0),
      PolyORB.Types.To_PolyORB_String
        ("th1"),
      Standard.Natural
        (0)),
      (Standard.Natural
        (32),               --  thread th2 Priority
      Standard.Natural
        (0),
      PolyORB.Types.To_PolyORB_String
        ("th2"),
      Standard.Natural
        (2)));

   package Priority_Manager is
     new ARAO.Setup.OA.Multithreaded.Prio
        (Threads_Array_);

   procedure Initialize;

end ARAO.Setup.Application;
```

## A.3 Node positioning

Node (process) location is done via a native mechanism of PolyORB. By overloading the abstract function Get_Conf of PolyORB.Parameters, we can assign a specific location to a node.

For each process, Ocarina will generate a package `PolyORB.Parameters.Partition` which will contains a static array and a Get_Conf function definition linking the current node location to PolyORB local data. When PolyORB will Initialize itself, this function will be called as it's registered in the Initialize hierarchy.

Example :

```
system implementation position.impl
subcomponents
  proc : process sender_node.simple {ARAO::port_number => 3200;};
  proc_1 : processor a_processor {ARAO::location => "127.0.0.1"};
properties
  actual_processor_binding => reference proc_1 applies to proc;
end position.impl;
```

```
package body PolyORB.Parameters.Partition is

   type Parameter_Entry is
      record
         Key : PolyORB.Utils.Strings.String_Ptr;
         Val : PolyORB.Utils.Strings.String_Ptr;
      end record;

   Conf_Table : constant array (1 .. 2)
     of Parameter_Entry :=
     ((new Standard.String'
```

```
        ("polyorb.protocols.iiop.default_addr"),
       new Standard.String'
        ("127.0.0.1")),
       (new Standard.String'
        ("polyorb.protocols.iiop.default_port"),
       new Standard.String'
        ("3200")));

   type Partition_Source is
     new PolyORB.Parameters.Parameters_Source with null record;

   The_Partition_Source : aliased Partition_Source;

   function Get_Conf
     (Source : access Partition_Source;
      Section : Standard.String;
      Key : Standard.String)
     return Standard.String;
   --  Called by PolyORB Initialization
   --  return the configuration data as in the Conf_Table array

   procedure Initialize;
   --  Initialize PolyORB by registerering Get_Conf function

end PolyORB.Parameters.Partition;
```

## A.4 Description of the ARAO API

ARAO, the middleware API, contains package to use and configure the PolyORB middleware.

### A.4.1 API to manipulate PolyORB

#### A.4.1.1 ARAO.Obj_Adapters

This package defines the following subprograms:

**Link_To_Obj_Adapter**: This procedure performs the link between the object reference (used by a client to send a request) and the servant who does the job specified by the request. This procedure assumes that the middlware is correctly set up and that a object adapter is created.

```
procedure Link_To_Obj_Adapter
     (T_Object :     PolyORB.Servants.Servant_Access;
      Ref      : out PolyORB.References.Ref);
```

#### A.4.1.2 ARAO.RT_Obj_Adapters

This package defines the following subprograms:

**Link_To_Obj_Adapter**: This procedure performs the link between the object reference (used by a client to send a request) and the servant who does the job specified by the request. This procedure assumes that the middlware is correctly set up and that a real-time object adapter is created for that Servant (instead of for the whole node as in ARAO.Obj_Adapter).

```
procedure Link_To_Obj_Adapter
     (T_Object   : PolyORB.Servants.Servant_Access;
      Ref        : out PolyORB.References.Ref;
      Thread_Name : Standard.String;
      Priority   : Integer := System.Default_Priority);
```

#### A.4.1.3 ARAO.Periodic_Threads

This package defines the following subprograms:

**Create_Periodic_Thread**: This procedure creates a periodic thread. The fact that the thread is periodic is handled in the TP procedure. Also, we assume that the PolyORB thread pool was properly created during the setup phase. Storage_size 0 is default size (not really 0 bit).

```
procedure Create_Periodic_Thread
     (TP           : Parameterless_Procedure;
      Priority     : System.Any_Priority := System.Default_Priority;
      Storage_Size : Integer := 0);
```

#### A.4.1.4 ARAO.Requests

This package defines the following subprograms:

**Emit_Msg**: This procedure creates a request whose target is the reference Ref. The PortName argument is used to distinguish the different port of one single thread. The data sent by the request (Item) must be of og the PolyORB neutral type (Any).

```
procedure Emit_Msg
     (Item     : PolyORB.Any.Any;
      Ref      : PolyORB.References.Ref;
      PortName : String);
```

#### A.4.1.5 ARAO.Utils

This package defines the following subprograms:

**Get_Ref**: Get the reference Ref from the properties of the remote servants.

```
procedure Get_Ref
     (Ref           : in out PolyORB.References.Ref;
      Host_Location : String;
      Port_Number   : Positive;
      Servant_Index : Natural;
      Protocol      : String);
```

**Get_GIOP_Ref**: Get the reference Ref from the properties of the remote servants for IIOP profiles.

```
procedure Get_GIOP_Ref
     (Ref           : in out PolyORB.References.Ref;
      Ior_Ref       : String);
```

### A.4.2 PolyORB Setup files

#### A.4.2.1 ARAO.Setup.Ocarina_OA

Set up the Ocarina Object Adapter

This package defines no subprogram

#### A.4.2.2 ARAO.Setup.OA.Multithreaded

Set up translation procedures for PolyORB priorities. Needed by RTPOA setup. has to be called before ARAO.Setup.OA.Multithreaded.Prio.Initialize.

This package defines no subprogram

#### A.4.2.3 ARAO.Setup.OA.Multithreaded.Prio

Setup an object adapter for multithread processes with request priority management.

This package defines the following subprograms:

**Initialize**: Create a Real-Time Object Adapter (RTPOA) for each IN port of the caller process. This procedure assumes that PolyORB was correctly setup, and particulary that PolyORB.RT_POA was previously withed. The RTPOAs will be created with respect to in port thread priority, stack size and number of connected out ports.

```
procedure Initialize;
```

# Appendix B 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.

# Index