Ocarina

A Compiler for the AADL
for Ocarina 2.0w, 11 November 2012

Jérôme Hugues, Thomas Vergnaud, Bechir Zalila

Table of Contents

About This Guide

This guide describes the use of Ocarina, a compiler for the AADL.

It presents the features of the compiler, related APIs and tools; and details how to use them to build and exploit AADL models.

It also details model transformations of AADL models onto

- Ada code using the ARAO AADL runtime built on top of PolyORB;
- Petri Net models

Companion documents describe other add-ons for Ocarina:

- PolyORB-HI/Ada, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets Ada targets: Native or bare board runtimes;
- PolyORB-HI/C, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets C targets: POSIX and RT-POSIX systems, RTEMS;

What This Guide Contains

This guide contains the following chapters:

- Chapter 1 [Introduction], page 3 provides a brief description of Ocarina.
- Chapter 2 [AADL], page 4 provides a quick overview of the AADL language.
- Chapter 3 [Installation], page 5 details how to configure and install Ocarina on your system.
- Chapter 4 [Using Ocarina], page 7 details the utilization of Ocarina
- Chapter 6 [Ocarina API Reference Manual], page 11
- Chapter 7 [Petri Net Mapping Rules], page 40 details the mapping rules used by Ocarina to generate Petri nets from AADL models
- Chapter 8 [AADL modes for Emacs and vim], page 44 presents the Emacs and vim modes for AADL packaged with Ocarina
- Appendix A [Standard AADL property files], page 45 provides the standard AADL property sets.
- Appendix B [Ocarina AADL property files], page 57 lists Ocarina specific property files.
- Appendix C [Conformance to standards], page 61 discusses the conformance of Ocarina to the AADL standard.
- Appendix D [GNU Free Documentation License], page 62 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.

1 Introduction

Ocarina is an application that can be used to analyze and build applications from AADL descriptions. Because of its modular architecture, Ocarina can also be used to add AADL functions to existing applications. Ocarina supports the AADL 1.0 and AADLv2 standards and proposes the following features :

1. Parsing and pretty printing of AADL models
2. Semantics checks
3. Code generation, using one of the four code generators
   - ARAO/Ada, an Ada AADL runtime built on top of PolyORB;
   - PolyORB-HI/Ada, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets Ada targets: Native or bare board runtimes;
   - PolyORB-HI/C, a High-Integrity AADL runtime and its code generator built on top of Ocarina that targets C targets: POSIX systems, RTEMS;
   - POK, a partioned operating system compliant with the ARINC653 standard.
4. Model checking using Petri nets;
5. Computation of Worst-Case Execution Time using the Bound-T tool from Tidorum Ltd.;
6. REAL, Requirement Enforcement and Analysis Language, an AADLv2 annex language to evaluate properties and metrics of AADLv2 architectural models;
7. Scheduling analysis of AADL models, with a gateway to the Cheddar scheduling analysis tool from the Université de Bretagne Occidentale.

Companion documents describe some of these add-ons for Ocarina:

2 The Architecture Analysis & Design Language

AADL stands for Architecture Analysis & Design Language. It can be expressed using graphical and textual syntaxes; an XML representations is also defined to ease the interoperability between tools.

AADL aims at allowing for the description of Distributed Real-Time Embedded (DRE) systems by assembling blocks separately developed. Thus it focuses on the definition of clear block interfaces, and separates the implementations from those interfaces. AADL allows for the description of both software and hardware parts of a system. Here is a brief presentation of the language; more information can be found on the AADL web site.

An AADL description is made of *components*. The AADL standard defines software components (data, threads, thread groups, subprograms, processes), execution platform components (memory, buses, processors, devices) and hybrid components (systems).

Components model well identified elements of the actual system. **Subprograms** model procedures such as those in C or Ada. **Threads** model the active part of an application (such as POSIX threads). **Processes** are memory spaces that contain the **threads**. **Thread groups** are used to create an hierarchy among threads. **Processors** model micro-processors and a minimal operating system (mainly a scheduler). **Memories** model hard disks, RAMs, etc. **Buses** model all kinds of networks, wires, etc. **Devices** model sensors, etc. Unlike other components, **systems** do not represent anything concrete; they actually create building blocks to help structure the description.

Component declarations have to be instantiated into subcomponents of other components in order to model an architecture. At the top-level, a system contains all the component instances. Most components can have subcomponents, so that an AADL description is hierarchical.

Each component has an interface (called **component type**) that provides **features** (e.g. communication ports). Components communicate one with another by **connecting** their features. To a given component type correspond zero or several implementations. Each of them describe the internals of the components: subcomponents, connections between those subcomponents, etc. An implementation of a thread or a subprogram can specify **call sequences** to other subprograms. This helps describe the whole execution flows in the architecture.

AADL defines a set of standard **properties** that can be attached to most elements (components, connections, features, etc.). Standard properties are used to specify things such as the clock frequency of a processor, the execution time of a thread, the bandwidth of a bus, etc. In addition, it is possible to add user-defined properties, to express specific description constraints.

By default, all elements of an AADL description are declared in a global name space. To avoid possible name conflicts in the case of a large description, it is possible to gather components within **packages**. Thus, packages help structure the description, while systems help structure the architecture. A package can have a public part and a private part; only the elements of the package can have a visibility on the private part. Packages can contain components declarations. So, they can be used to structure the description from a logical point of view. Unlike systems, they do not impact the architecture.

3 Installation

3.1 Supported Platforms

Ocarina has been compiled and successfully tested on the following platforms:

- Linux
- Mac OS X
- Solaris
- FreeBSD
- Windows

Note: Ocarina should compile and run on every target for which GNAT is available.

3.2 Build requirements

An Ada compiler:

- GNAT Pro 6.0.1 or later
- GNAT GPL 2006 or later
- GNAT GCC 4.4.0 or later

Optional:

- PolyORB (`http://libre.adacore.com/polyorb/`) if you want to build distributed applications using Ocarina.
- Xcov and Qemu from the project Couverture if you want to get coverage information about the distributed applications. (`http://forge.open-do.org/projects/couverture/`)

Note: per construction, the macro `configure` used to find your GNAT compiler looks first to the executable `gnatgcc`, then `adagcc` and finally to `gcc` to find out which Ada compiler to use. You should be very careful with your path and binaries if you have multiple GNAT versions installed. See below explanations on the `ADA` environment variable if you need to override the default guess.

3.3 Build instructions

To compile and install Ocarina, execute:

```
     % ./configure [some options]
     % make              (or gmake if your make is not GNU make)
     % make install      (ditto)
```

This will install files in standard locations. If you want to choose another prefix than '`/usr/local`', give configure a '`--prefix=whereveryouwant`' argument.

Note: at this time, you MUST use GNU make to compile this software.

3.4 Additional instructions for cross platforms

Only one Ocarina installation is currently possible with a given `--prefix`. If both a native and a cross installation are needed on the same machine, distinct prefixes must be used.

3.5 Building the documentation

Ocarina documentation is built automatically with the Ocarina libraries and tools. It is installed in the '`${prefix}/share/doc/ocarina`'

3.5.1 Build Options

Available options for the '`configure`' script include:

- '`--enable-debug`': enable debugging information generation and supplementary runtime checks. Note that this option has a significant space and time cost, and is not recommended for production use.
- '`--with-ocarina-runtimes=x`': enable building Ocarina along with the requested runtimes. '`x`' is a set of valid runtimes located in the '`resources/runtimes`' directory. '`x`' is case insensitive. Examples of use:
  - '`--with-ocarina-runtimes=all`': compile Ocarina along with all the runtimes. All the Ocarina runtimes MUST be located in the '`resources/runtimes`' directory.
  - '`--with-ocarina-runtimes=none`': compile Ocarina along without any runtime.
  - '`--with-ocarina-runtimes=PolyORB-HI-Ada`': compile Ocarina along with the '`PolyORB-HI-Ada`' runtime. The directory '`polyorb-hi-ada`' MUST exist in the '`resources/runtimes`' directory.
  - '`--with-ocarina-runtimes="polyorb-hi-c PolyORB-HI-Ada"`': compile Ocarina along with the '`PolyORB-HI-Ada`' and the '`PolyORB-HI-C`' runtimes. The directories '`polyorb-hi-ada`' and '`polyorb-hi-c`' MUST exist in the '`resources/runtimes`' directory.
  - No option: compile Ocarina along with all the runtimes found in the '`resources/runtimes`' directory.

3.5.2 Compiler, Tools and Run-Time libraries Options

The following environment variables can be used to override configure's guess at what compilers to use:

- `CC`: the C compiler
- `ADA`: the Ada 95 compiler (e.g. gcc, gnatgcc or adagcc)

For example, if you have two versions of GNAT installed and available in your `PATH`, and configure picks the wrong one, you can indicate what compiler should be used with the following syntax:

```
% ADA=/path/to/good/compiler/gcc ./configure [options]
```

Ocarina will be compiled with GNAT build host's configuration, including run-time library. You may override this setting using `ADA_INCLUDE_PATH` and `ADA_OBJECTS_PATH` environment variables. See GNAT User's Guide for more details.

NOTE: Developers building Ocarina from the version control repository who need to rebuild the configure and Makefile.in files should use the script '`support/reconfig`' for this purpose. This should be done after each update from the repository. In addition to the requirements above, they will need autoconf 2.57 or newer, automake 1.6.3 or newer.

4 Using Ocarina

4.1 `ocarina`

This command is an extended front-end for the functions provided by Ocarina.

```
Usage:
      ocarina [options] files
      OR
      ocarina -help
  files are a non null sequence of AADL files

  General purpose options:
   -V  Output Ocarina version, then exit
   -s  Output Ocarina search directory, then exit

  Scenario file options:
   -b  build the generated application code
   -z  clean the generated application code
   -ec execute the generated application code and
       retrieve coverage information
   -er execute the generated application code and
       verify that there is no regression
   -p  only parse and instantiate the application model
   -c  only perform schedulability analysis

  Advanced user options:
   -d  Debug mode for developpers
   -q  Quiet mode (default)
   -t  [script] Run Ocarina in terminal interactive mode.
       If a script is given, interpret it then exit.
   -v  Verbose mode for users
   -x  Parse AADL file as an AADL scenario file
   -f  Parse predefined non standard property sets
   -i  Instantiate the AADL model
   -r  <name> The name of the instance tree root
   -o  Specify output file
   -I  Specify the inclusion paths
   -aadlv1  Use AADL v1 standard (default)
   -aadlv2  Use AADL v2 standard
   -real_lib Add a REAL file to be used as a theorem libraries by REAL annexes
   -g  Generate code from the AADL instance tree
       Registered backends:
        petri_nets
        boundt
        polyorb_hi_ada
        polyorb_qos_ada
        polyorb_hi_c
        polyorb_hi_rtsj
        pok_c
        stats
        subprograms
        real_theorem
        carts
        cheddar
        aadl
        aadl_min
        aadl_annex
        behavior_specification
        real_specification
   -arinc653  Generate code for ARINC653 API (POK backend only)
   -b  Generate and build code from the AADL model
   -z  Clean code generated from the AADL model
   -disable-annexes={annexes}  Desactive one or all annexes
```

```
Annexes :
 all
 behavior
 real
```

4.2 AADL Scenario Files

AADL scenario files are a very simple way to configure an AADL application. AADL scenario may consist of more than one AADL file but they all should be located in the same directory. Example:

The following file containing the common part of 2 AADL scenarios:

```
system RMA
properties
  Ocarina_Config::AADL_Files => ("rma.aadl");
  --  "rma.aadl" contains common AADL components (processes,
  --  threads, data types)

  Ocarina_Config::Needed_Property_Sets =>
    (value (Ocarina_Config::ARAO),
     value (Ocarina_Config::Cheddar_Properties));
  --  The non standard predefined property sets needed by the
  --  application.
end RMA;
```

The following files contains a system implementation of the previous one by adding specific parts for an application that will leads to a C code generation:

```
system implementation RMA.Impl_C
properties
  Ocarina_Config::AADL_Files +=> ("software_c.aadl");
  --  Note that this is an additive property
  --  association.

  Ocarina_Config::Generator => PolyORB_HI_C;
  --  The code generator

  Ocarina_Config::Root_System_Name => "RMA.C";
  --  The name of the root of the instance tree (optional if there is
  --  one unique eligible root system).
end RMA.Impl_C;
```

The following files contains a system implementation of the first one by adding specific parts for an application that will leads to a Ada code generation:

```
system implementation RMA.Impl_Ada
properties
  Source_Text +=> ("software_ada.aadl");
  --  Note that this is an additive property
  --  association.

  Ocarina_Config::Generator => PolyORB_HI_Ada;
  --  The code generator

  Ocarina_Config::Root_System_Name => "RMA.Ada";
  --  The name of the root of the instance tree (optional if there is
  --  one unique eligible root system).
end RMA.Impl_Ada;
```

Node that for the 2 last files, we used the “additive” for of AADL properties to “add” AADL files.

If the user invokes ‘ocarina -x’ on both ‘scenario_common.aadl’ and ‘scenario_c.aadl’, then ‘ocarina’ will be invoked to generate C code for the PolyORB-HI middleware.

If the user invokes ‘ocarina_ -x’ on both ‘scenario_common.aadl’ and ‘scenario_ada.aadl’, then ‘ocarina’ will be invoked to generate Ada code for the PolyORB-HI middleware.

4.3 ocarina-config

ocarina-config returns path and library information on Ocarina’s installation.

```
Usage: ocarina-config [OPTIONS]
Options:
        No option:
            Output all the flags (compiler and linker) required
            to compile your program.
        [--prefix[=DIR]]
            Output the directory in which Ocarina architecture-independent
           files are installed, or set this directory to DIR.
        [--exec-prefix[=DIR]]
            Output the directory in which Ocarina architecture-dependent
           files are installed, or set this directory to DIR.
        [--version|-v]
            Output the version of Ocarina.
        [--config]
            Output Ocarina’s configuration parameters.
        [--runtime[=<Runtime_Name>]]
            Checks the validity and the presence of the given runtime and
            then, outputs its path. Only one runtime can be requested at
            a time. If no runtime name is given, outputs the root directory
            of all runtimes.
        [--libs]
            Output the linker flags to use for Ocarina.
        [--properties]
            Output the location of the standard property file.
        [--resources]
            Output the location of resource files
            (typically the standard properties)
        [--cflags]
            Output the compiler flags to use for Ocarina.
        [--help]
            Output this message
```

This script can be used to compile user program that uses Ocarina’s API. See Chapter 6 [Ocarina API Reference Manual], page 11.

5 A Tutorial of AADL & Ocarina

This chapter will appear in a future revision of Ocarina.

6 Ocarina API Reference Manual

This chapter describes Ocarina's API, an API to manipulate AADL models.

6.1 The Ocarina Core Library

6.1.1 Rationale of the core library

The core library holds the tree structures of AADL descriptions. It provides the facilities required to manipulate the AADL descriptions.

6.1.1.1 Internal representation

AADL descriptions are usually sets of declarations. Verifications can be performed on these representations, but the declarations must be instantiated in order to be able to fully compute the architectures. Therefore, the processing of an AADL description is usually performed in two steps:

- the building of an AADL model that corresponds to the AADL declarations;
- then the instantiation of the model to manipulate the actual architecture.

The Ocarina core library provides the necessary functions to build AADL declaration, validate the model and then instantiate it.

Both model and instance AADL descriptions are represented by abstract syntax trees. These trees are made of several nodes, defined in '`src/core/tree/ocarina-nodes.idl`'. As the structure is rather complex, higher lever notions are defined. Thus, Ocarina mainly manipulate *entities*. AADL entities are:

- namespaces:
  - AADL specification,
  - packages,
  - property sets;
- components:
  - component types and implementations,
  - port group types;
- properties:
  - property names, types and constants,
  - property associations;
- subclauses:
  - features (ports, port group specifications, subprograms as features, subcomponent accesses),
  - subcomponents,
  - subprogram call sequences and subprogram calls,
  - connections,
  - modes,
  - flows;
- annexes (libraries and subclauses).

6.1.2 Code organization

The Ocarina core is made of three main parts: the manipulation of the tree nodes, the manipulation of AADL models and the manipulation of AADL architecture instances.

6.1.2.1 AADL nodes

The manipulation of the tree nodes consists of the structures of the tree, and functions to manipulate them at a low level. The corresponding files are located in '`src/core/tree`'.

The functions of lowest level are located in the package `Ocarina.Nodes`. They allow for the direct manipulation of tree, without checking any semantics. Some sort of assembly language to manipulate nodes.

Some higher-level access functions are provided in packages such as `Ocarina.Entities`. Those functions provide higher level access to entity information such as getting their name, etc. without dealing with the actual structure of the nodes. They can be considered as low level functions. However, these functions should always be preferred to lowest level ones as they manipulates entities. These functions will be described in the next section;

6.1.2.2 AADL models

Functions are provided to manipulate trees of AADL models. They allow to build, check and interrogate model trees. The files are located in '`src/core/model`'.

High level functions are provided to manipulate AADL models. They are meant to hide the actual structure of the Ocarina tree and only manipulate AADL notions (components, connections, etc.) Several packages are located in '`src/core/model`' and provide facilities to build, verify and interrogate AADL models.

6.1.2.3 AADL instances

Other functions are provided to manipulate trees of AADL instances. Like for AADL models, it is possible to build, check and interrogate trees. However, instance trees cannot be directly built: they are computed from model trees, thus instantiating AADL models. The files are located in '`src/core/instance`'.

6.1.3 Low level API to manipulate tree nodes

Low level functions that are provided in `Ocarina.Nodes` and the package `Ocarina.Entities` and its child packages.

Functions of `Ocarina.Nodes` allow the direct manipulation of the node tree. Therefore it is difficult to use them. We do not describe this API. The packages `Ocarina.Entities` provide an API to manipulate entities, thus easing the manipulation of the tree elements.

6.1.3.1 Ocarina.ME_AADL.AADL_Tree.Entities

This package defines the following subprograms:

Get_Entity_Category: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Entity_Category (Node : Types.Node_Id) return Entity_Category;
```

Get_Name_Of_Entity: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Name_Of_Entity
      (Entity : Types.Node_Id;
       Get_Display_Name : Boolean := True)
      return Types.Name_Id;
```

Get_Name_Of_Entity: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Name_Of_Entity
      (Entity : Types.Node_Id;
       Get_Display_Name : Boolean := True)
      return String;
```

Get_Name_Of_Entity_Reference: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Name_Of_Entity_Reference
      (Entity_Ref : Types.Node_Id;
       Get_Display_Name : Boolean := True)
      return Types.Name_Id;
```

Get_Name_Of_Entity_Reference: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Name_Of_Entity_Reference
      (Entity_Ref : Types.Node_Id;
       Get_Display_Name : Boolean := True)
      return String;
```

Get_Referenced_Entity: Return the entity referenced by an entity reference, or No_Node if nothing is pointed

```
function Get_Referenced_Entity
      (Entity_Ref : Types.Node_Id)
      return Types.Node_Id;
```

Set_Referenced_Entity: Set the entity that is to be referenced by the entity reference

```
procedure Set_Referenced_Entity (Entity_Ref, Entity : Types.Node_Id);
```

Entity_Reference_Path_Has_Several_Elements: return True if the path has more than one element.

```
function Entity_Reference_Path_Has_Several_Elements
      (Entity_Ref : Types.Node_Id)
      return Boolean;
```

Duplicate_Identifier:

– This following section is relative to Entities Components –

```
function Duplicate_Identifier
      (Identifier : Types.Node_Id)
      return Types.Node_Id;
```

Get_Category_Of_Component: return the category of the component type, implementation or instance.

– This following section is relative to Entities Components Connections –

```
function Get_Category_Of_Component
      (Component : Types.Node_Id)
      return Component_Category;
```

Get_Category_Of_Connection:

– This following section is relative to Entities Components Flows –

```
function Get_Category_Of_Connection
      (Connection : Types.Node_Id)
      return Connection_Type;
```

Get_Category_Of_Flow:

– This following section is relative to Entities Components Subcomponents –

```
function Get_Category_Of_Flow (Flow : Types.Node_Id) return Flow_Category;
```

Get_Category_Of_Subcomponent: Return the category of the subcomponent or subcomponent instance.

```
function Get_Category_Of_Subcomponent
      (Subcomponent : Types.Node_Id)
      return Component_Category;
```

Get_Corresponding_Component: return the corresponding component declaration, if there is any, or No_Node.

– This following section is relative to Entities Components SubprogramCall –

```
function Get_Corresponding_Component
      (Subcomponent : Types.Node_Id)
      return Types.Node_Id;
```

Get_Corresponding_Subprogram: return the corresponding subprogram declaration, if there is any, or No_Node.

– This following section is relative to Entities Namespaces (Packages...) –

```
function Get_Corresponding_Subprogram
      (Call : Types.Node_Id)
      return Types.Node_Id;
```

Package_Has_Public_Declarations_Or_Properties: Returns True if the package has public elements, else False. Pack must reference a package specification.

```
function Package_Has_Public_Declarations_Or_Properties
      (Pack : Types.Node_Id)
      return Boolean;
```

Package_Has_Private_Declarations_Or_Properties: Returns True if the package has private elements, else False. Pack must reference a package specification.

– This following section is relative to Entities Messages –

```
function Package_Has_Private_Declarations_Or_Properties
      (Pack : Types.Node_Id)
      return Boolean;
```

Display_Node_Kind_Error: Add Item to the end of the path that constitutes the reference to the entity.

```
function Display_Node_Kind_Error      (Node : Types.Node_Id)
      return Boolean;
```

DNKE: Add Item to the end of the path that constitutes the reference to the entity.

```
function DNKE (Node : Types.Node_Id) return Boolean
      renames Display_Node_Kind_Error
```

Add_Path_Element_To_Entity_Reference: Add Item to the end of the path that constitutes the reference to the entity.

```
procedure Add_Path_Element_To_Entity_Reference
      (Entity_Ref, Item : Types.Node_Id);
```

6.1.3.2 Ocarina.ME_AADL.AADL_Tree.Entities.Properties

This package provides functions to create or read property names, types, constants and associations.

This package defines the following subprograms:

Value_Of_Property_Association_Is_Undefined:

```
function Value_Of_Property_Association_Is_Undefined
      (Property : Node_Id)
      return Boolean;
```

Type_Of_Property_Is_A_List:

```
function Type_Of_Property_Is_A_List
     (Property : Node_Id)
     return Boolean;
```

Get_Type_Of_Property:

```
function Get_Type_Of_Property
     (Property             : Node_Id;
      Use_Evaluated_Values : Boolean := True)
     return Property_Type;
```

Get_Type_Of_Property_Value:

```
function Get_Type_Of_Property_Value
     (Property_Value       : Node_Id;
      Use_Evaluated_Values : Boolean := True)
     return Property_Type;
```

Get_Integer_Of_Property_Value:

```
function Get_Integer_Of_Property_Value
     (Property_Value : Node_Id)
     return Unsigned_Long_Long;
```

Get_Float_Of_Property_Value:

```
function Get_Float_Of_Property_Value
     (Property_Value : Node_Id)
     return Long_Long_Float;
```

Get_String_Of_Property_Value:

```
function Get_String_Of_Property_Value
     (Property_Value : Node_Id)
     return Name_Id;
```

Get_String_Of_Property_Value:

```
function Get_String_Of_Property_Value
     (Property_Value : Node_Id)
     return String;
```

Get_Enumeration_Of_Property_Value:

```
function Get_Enumeration_Of_Property_Value
     (Property_Value : Node_Id)
     return Name_Id;
```

Get_Enumeration_Of_Property_Value:

```
function Get_Enumeration_Of_Property_Value
     (Property_Value : Node_Id)
     return String;
```

Get_Boolean_Of_Property_Value:

```
function Get_Boolean_Of_Property_Value
     (Property_Value : Node_Id)
     return Boolean;
```

Get_Classifier_Of_Property_Value:

```
function Get_Classifier_Of_Property_Value
     (Property_Value : Node_Id)
     return Node_Id;
```

Get_Reference_Of_Property_Value:

```
function Get_Reference_Of_Property_Value
      (Property_Value : Node_Id)
      return Node_Id;
```

Get_Value_Of_Property_Association:

```
function Get_Value_Of_Property_Association
      (Property : Node_Id)
      return Value_Type;
```

Find_Property_Association_From_Name:

```
function Find_Property_Association_From_Name
      (Property_List : List_Id;
       Property_Name : Name_Id;
       In_Mode       : Name_Id := No_Name)
      return Node_Id;
```

Find_Property_Association_From_Name:

```
function Find_Property_Association_From_Name
      (Property_List : List_Id;
       Property_Name : String;
       In_Mode       : Name_Id := No_Name)
      return Node_Id;
```

Resolve_Term_In_Property:

```
procedure Resolve_Term_In_Property
      (Property  : Node_Id;
       Value     : Node_Id;
       Kind_Node : Node_Kind);
```

6.1.4 API to build and manipulate AADL models

This section describes the high level function's to manipulate AADL models:

1. Builder API~: These functions (declared in the `Ocarina.Builder` hierarchy) are provided to create the main kinds of nodes: components, namespaces, subclauses, annexes, properties, etc. They are named `Add_New_....`. They create a new node and insert it as a child of a parent node, except for the root node. This helps ensure that the description structure is valid, since the API only allows valid constructions. All these functions take at least three parameters:
   - the location of the node; for example the position in a file that corresponds to the element that is parsed; the location is of type Location as defined in the package `Locations`;
   - the name that should be set for the node, since all those nodes can have names; the name is an identifier node; for some entities it can be set to `No_Node` (connections, etc.);
   - the namespace or the parent node, which shall contain the newly created node; this node must be different from `No_Node`, and has to be of a correct kind, according to the AADL standard BNF.

   Other parameters may be required, to specify information such as the category of the component (bus, process, etc.), whether the subclause is a refinement, etc. Some parameters have default values, meaning that they can be omitted if the value is not known when the node is created; the values can be set later, using lower-level access functions. Functions simply named `Add_...` are used internally to add child nodes to their parents. No verification is performed; they just ensure the child lists are created before inserting the node in them.

2. Verification API: Functions are provided to check the validity of AADL models. They ensure that all referenced AADL entities are declared, that their type is correct, etc. They also check that AADL properties are consistent.
3. Interrogation API: These functions (declared in the `Ocarina.Analyzer` hierarchy) are provided to interrogate AADL models in order to find entities, get properties, etc.

6.1.4.1 Ocarina.Builder

This package defines no subprogram

6.1.4.2 Ocarina.Builder.AADL

This package defines no subprogram

6.1.4.3 Ocarina.Builder.AADL.Annexes

This package provides functions to build annex nodes into the AADL tree.

This package defines the following subprograms:

Set_Annex_Content: Set the text of the annex. Annex is the Node_Id of the annex library or subclause, returned by Add_New_Annex_Subclause or Add_New_Annex_Library. Text is the Name_Id referencing the text of the annex. Return True is everything went right, else False.

```
function Set_Annex_Content
     (Annex : Types.Node_Id;
      Text  : Types.Name_Id)
     return Boolean;
```

Add_New_Annex_Subclause: Create a new annex subclause. An annex subclause can be inserted into a component declaration (type or implementation) or a port group declaration. Loc is the location of the annex in the parsed text. Annex_Name is the name of the annex subclause. Namespace is the component or the port group where the annex must be inserted. This functions returns the Node_Id of the newly created annex subclause node, or No_Node if something went wrong.

```
function Add_New_Annex_Subclause      (Loc        : Locations.Location;
      Annex_Name : Types.Node_Id;
      Namespace  : Types.Node_Id;
      In_Modes   : Types.Node_Id)
     return Types.Node_Id;
```

Add_New_Annex_Library: Create a new annex library. An annex library can be inserted into a package or the top level AADL specification (i.e. the unnamed namespace). Loc is the location of the annex in the parsed text. Annex_Name is the name of the annex library. Namespace is the package specification or the top level AADL specification where the annex must be inserted. This functions returns the Node_Id of the newly created annex library node, or No_Node if something went wrong.

```
function Add_New_Annex_Library
     (Loc        : Locations.Location;
      Annex_Name : Types.Node_Id;
      Namespace  : Types.Node_Id;
      Is_Private : Boolean             := False)
     return Types.Node_Id;
```

Add_New_Annex_Path: Create a new annex path node. Loc is the location of the annex path in the parsed text. Container is the namespace which contain the annex path declaration. Annex_Identifier is the identifier of the annex path, it maybe No_Node. List_Identifiers is the list of identifiers declared. This function returns the Node_Id of the newly created annex path node, or No_Node if something went wrong.

```
function Add_New_Annex_Path
     (Loc              : Locations.Location;
      Container        : Types.Node_Id;
      Annex_Identifier : Types.Node_Id;
      List_Identifiers : Types.List_Id)
     return Types.Node_Id;
```

6.1.4.4 Ocarina.Builder.AADL.Components

This package defines the following subprograms:

Add_Annex: Add an annex subclause into a component (type or implementation). Component is a Node_Id referencing the component. Annex is a Node_Id referencing the annex subclause. Returns True if the annex was correctly added into the component, else False.

```
function Add_Annex
     (Component : Node_Id;
      Annex     : Node_Id)
     return Boolean;
```

Add_Connection: Add a connection into a component implementation. Component is a Node_Id referencing the component implementation. Connection is a Node_Id referencing the connection. Returns True if the connection was correctly added into the component, else False.

```
function Add_Connection
     (Component  : Node_Id;
      Connection : Node_Id)
     return Boolean;
```

Add_Feature: Add a feature into a component type. Component is a Node_Id referencing the component type. Feature is a Node_Id referencing the feature. Returns True if the feature was correctly added into the component, else False.

```
function Add_Feature
     (Component : Node_Id;
      Feature   : Node_Id)
     return Boolean;
```

Add_Refined_Type: Add a refined type into a component implementation. Refined types correspond to refinements of the component type features.

```
function Add_Refined_Type
     (Component : Node_Id;
      Refined_Type : Node_Id)
     return Boolean;
```

Add_Subcomponent: Add a subcomponent into a component implementation. Component is a Node_Id referencing the component implementation. Subcomponent is a Node_Id referencing the subcomponent. Returns True if the subcomponent was correctly added into the component, else False.

```
function Add_Subcomponent
     (Component    : Node_Id;
      Subcomponent : Node_Id)
     return Boolean;
```

Add_Prototype: Add a prototype into a component implementation or a component type. Component is a Node_Id referencing the component implementation or the component type. Prototype is a Node_Id referencing the prototype. Returns True if the prototype was correctly added into the component, else False.

```
function Add_Prototype      (Component : Node_Id;
        Prototype : Node_Id)
       return Boolean;
```

Add_Subprogram_Call_Sequence: Add a subprogram call sequence into a component implementation. Component is a Node_Id referencing the component implementation. Call_Sequence is a Node_Id referencing the subcomponent. Returns True if the sequence was correctly added into the component, else False.

```
function Add_Subprogram_Call_Sequence
       (Component     : Node_Id;
        Call_Sequence : Node_Id)
       return Boolean;
```

Add_Flow_Spec: Add a flow specification into a component type. Component is a Node_Id referencing the component type. Flow_Spec is a Node_Id referencing the flow. Returns True if the flow was correctly added into the component, else False.

```
function Add_Flow_Spec
       (Component : Node_Id;
        Flow_Spec : Node_Id)
       return Boolean;
```

Add_Flow_Implementation: Add a flow implementation into a component implementation. Component is a Node_Id referencing the component implementation. Flow_Impl is a Node_Id referencing the flow. Returns True if the flow was correctly added into the component, else False.

```
function Add_Flow_Implementation
       (Component : Node_Id;
        Flow_Impl : Node_Id)
       return Boolean;
```

Add_End_To_End_Flow_Spec: Add an end to end flow specification into a component implementation. Component is a Node_Id referencing the component implementation. Flow_Impl is a Node_Id referencing the flow. Returns True if the flow was correctly added into the component, else False.

```
function Add_End_To_End_Flow_Spec
       (Component       : Node_Id;
        End_To_End_Flow : Node_Id)
       return Boolean;
```

Add_Mode: Add a mode (declaration or transition) into a component implementation. Component is a Node_Id referencing the component implementation. Mode is a Node_Id referencing the mode declaration or mode transition. Returns True if the mode was correctly added into the component, else False.

```
function Add_Mode
       (Component : Node_Id;
        Mode      : Node_Id)
       return Boolean;
```

Add_Property_Association: Add a property association into a component (type or implementation). Component is a Node_Id referencing the component type or implementation. Property_Association is a Node_Id referencing the property association. Returns True if the property was correctly added into the component, else False. Component creation

```
function Add_Property_Association
       (Component            : Node_Id;
        Property_Association : Node_Id)
```

```
     return Boolean;
```

Add_New_Component_Type: Create a new component type node. A component type can be inserted into a package or the top level AADL specification (aka the unnamed namespace). Loc is the location of the component in the parsed text. Identifier is a Node_Id referencing the name of the component. Namespace is either a package specification or the top level AADL specification. Component_Type is the component category (processor, memory, process, etc.). Is_Private indicates if the component is declaraed in the private or the public part of the package; it is only relevant if Namespace references a package specification. Returns the Node_Id of the newly created component type node, or No_Node if something went wrong.

```
function Add_New_Component_Type
     (Loc            : Location;
      Identifier     : Node_Id;
      Namespace      : Node_Id;
      Component_Type : Ocarina.ME_AADL.Component_Category;
      Is_Private     : Boolean := False)
     return Node_Id;
```

Add_New_Component_Implementation: Create a new component implementation node. A component implementation can be inserted into a package or the top level AADL specification (aka the unnamed namespace). Loc is the location of the component in the parsed text. Identifier is a Node_Id referencing the name of the component. Namespace is either a package specification or the top level AADL specification. Component_Type is the component category (processor, memory, process, etc.). Is_Private indicates if the component is declaraed in the private or the public part of the package; it is only relevant if Namespace references a package specification. Returns the Node_Id of the newly created component implementation node, or No_Node if something went wrong.

```
function Add_New_Component_Implementation
     (Loc            : Location;
      Identifier     : Node_Id;
      Namespace      : Node_Id;
      Component_Type : Ocarina.ME_AADL.Component_Category;
      Is_Private     : Boolean := False)
     return Node_Id;
```

Add_New_Feature_Group: Create a new feature group type (AADL_V2) or port group type (AADL_V1). It can be inserted into a package or the top level AADL specification.)

```
function Add_New_Feature_Group
     (Loc        : Location;
      Name       : Node_Id;
      Namespace  : Node_Id;
      Is_Private : Boolean := False)
     return Node_Id;
```

6.1.4.5 Ocarina.Builder.AADL.Components.Connections

This package defines the following subprograms:

Add_Property_Association: Add a property association to the connection declaration. Connection must reference a connection declaration. Property_Association references the property association. Return True if everything went right, else False.

```
function Add_Property_Association
     (Connection           : Node_Id;
      Property_Association : Node_Id)
     return Boolean;
```

Add_New_Connection: Create and add a new connection into a component implementation. Loc is the location of the connection in the parsed text. Name references an identifier which contains the name of the connection, if any. Comp_Impl references the component implementation. Category is the type of the connection. Is_Refinement indicates wether the connection is a refinement or not. Source and Destination are the left and right memebers of the connection. In_Modes contains the list of the modes associated to the connection. Name can be No_Node, if the connection is not nammed. Return the Node_Id of the newly created connection if everything went right, else No_Node.

```
function Add_New_Connection
     (Loc           : Location;
      Name          : Node_Id;
      Comp_Impl     : Node_Id;
      Category      : Ocarina.Me_AADL.Connection_Type;
      Is_Refinement : Boolean := False;
      Is_Bidirect   : Boolean := False;
      Source        : Node_Id := No_Node;
      Destination   : Node_Id := No_Node;
      In_Modes      : Node_Id := No_Node)
     return Node_Id;
```

New_Connection: Create a new connection into a component implementation. Loc is the location of the connection in the parsed text. Name references an identifier which contains the name of the connection, if any. Category is the type of the connection. Is_Refinement indicates wether the connection is a refinement or not. Source and Destination are the left and right memebers of the connection. In_Modes contains the list of the modes associated to the connection. Name can be No_Node, if the connection is not nammed. Return the Node_Id of the newly created connection if everything went right, else No_Node.

```
function New_Connection
     (Loc           : Location;
      Name          : Node_Id;
      Category      : Ocarina.Me_AADL.Connection_Type;
      Is_Refinement : Boolean := False;
      Is_Bidirect   : Boolean := False;
      Source        : Node_Id := No_Node;
      Destination   : Node_Id := No_Node;
      In_Modes      : Node_Id := No_Node)
     return Node_Id;
```

6.1.4.6 Ocarina.Builder.AADL.Components.Features

The core API for the feature subclause of the component types and the port group types.

This package defines the following subprograms:

Add_Property_Association:

```
function Add_Property_Association
     (Feature              : Node_Id;
      Property_Association : Node_Id)
     return Boolean;
```

Add_New_Port_Spec:

```
function Add_New_Port_Spec
     (Loc               : Location;
      Name              : Node_Id;
      Container         : Node_Id;
```

```
          Is_In             : Boolean;
          Is_Out            : Boolean;
          Is_Data           : Boolean;
          Is_Event          : Boolean;
          Is_Feature        : Boolean;
          Is_Refinement     : Boolean := False;
          Associated_Entity : Node_Id := No_Node)
         return Node_Id;
```

Add_New_Port_Group_Spec:

```
function Add_New_Port_Group_Spec
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

Add_New_Feature_Group_Spec:

```
function Add_New_Feature_Group_Spec
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

Add_New_Server_Subprogram:

```
function Add_New_Server_Subprogram
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

Add_New_Data_Subprogram_Spec:

```
function Add_New_Data_Subprogram_Spec
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

Add_New_Subcomponent_Access:

```
function Add_New_Subcomponent_Access
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_Refinement : Boolean := False;
          Category      : Ocarina.Me_AADL.Component_Category;
          Is_Provided   : Boolean)
         return Node_Id;
```

Add_New_Parameter:

```
function Add_New_Parameter
         (Loc           : Location;
          Name          : Node_Id;
          Container     : Node_Id;
          Is_In         : Boolean := True;
          Is_Out        : Boolean := True;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

6.1.4.7 Ocarina.Builder.AADL.Components.Flows

This package defines the following subprograms:

Add_Property_Association: Add a property association to the flow declaration. Flow must reference a flow implementation or a flow specification. Property_Association references the property association. Return True if everything went right, else False.

```
    function Add_Property_Association
         (Flow                 : Node_Id;
          Property_Association : Node_Id)
         return Boolean;
```

Add_New_Flow_Spec: Create a new flow specification inside a component type

```
    function Add_New_Flow_Spec
         (Loc           : Location;
          Name          : Node_Id;
          Comp_Type     : Node_Id;
          Category      : Flow_Category;
          Source_Flow   : Node_Id;
          Sink_Flow     : Node_Id;
          Is_Refinement : Boolean := False)
         return Node_Id;
```

Add_New_Flow_Implementation: Create a new flow implementation inside a component implementation

```
    function Add_New_Flow_Implementation
         (Loc           : Location;
          Container     : Node_Id;
          Name          : Node_Id;
          Category      : Flow_Category;
          In_Modes      : Node_Id;
          Is_Refinement : Boolean;
          Source_Flow   : Node_Id;
          Sink_Flow     : Node_id)
         return Node_Id;
```

Add_New_End_To_End_Flow_Spec: Create a new end to end flow specification inside a component implementation

```
    function Add_New_End_To_End_Flow_Spec
         (Loc           : Location;
          Container     : Node_Id;
          Name          : Node_Id;
          In_Modes      : Node_Id;
          Is_Refinement : Boolean;
          Source_Flow   : Node_Id;
          Sink_Flow     : Node_id)
         return Node_Id;
```

6.1.4.8 Ocarina.Builder.AADL.Components.Modes

This package provides functions to handle modes in the component implementations.

This package defines the following subprograms:

Add_Property_Association: Add a property association to the mode declaration or mode transition. Mode must either reference a mode declaration or a mode transition. Property_Association references the property association. Return True if everything went right, else False.

```
function Add_Property_Association
      (Mode : Node_Id;
       Property_Association : Node_Id)
      return Boolean;
```

Add_New_Mode: Add a new mode declaration into a component implementation. Loc is the location of the mode declaration in the parsed text. Identifier references an identifier containing the name of the mode. Component references the component implementation. Is_Implicit is used by other parts of the builder API, for "in modes" clauses. You should always keep it False. Return a Node_Id referencing the newly created mode if everything went right, else False.

```
function Add_New_Mode
      (Loc : Location;
       Identifier : Node_Id;
       Component : Node_Id)
      return Node_Id;
```

Add_New_Mode_Transition: Add a new empty mode transition into a component implementation. Source, Destination, etc. of the mode transition must be added manually after the node has been created. Loc is the location of the mode transition in the parsed text. Identifier references an identifier containing the name of the mode. Component references the component implementation. Return a Node_Id referencing the newly created mode if everything went right, else False.

```
function Add_New_Mode_Transition
      (Loc : Location;
       Component : Node_Id)
      return Node_Id;
```

Add_New_Mode_Transition_Trigger:

```
function Add_New_Mode_Transition_Trigger
      (Loc : Locations.Location;
       Identifier : Types.Node_Id;
       Is_Self : Boolean;
       Is_Processor : Boolean)
      return Types.Node_Id;
```

6.1.4.9 Ocarina.Builder.AADL.Components.Subcomponents

This package defines the following subprograms:

Add_Property_Association: Add a property association to the subcomponent declaration. Subcomponent must reference a Subcomponent declaration. Property_Association references the property association. Return True if everything went right, else False.

```
function Add_Property_Association
      (Subcomponent         : Node_Id;
       Property_Association : Node_Id)
      return Boolean;
```

Add_New_Subcomponent: Create and add a new subcomponent into a component implementation. Loc is the location of the subcomponent in the parsed text. Name references an identifier which contains the name of the subcomponent. Comp_Impl references the component implementation. Category is the type of the subcomponent. Is_Refinement indicates wether the connection is a refinement or not. In_Modes contains the list of the modes associated to the connection. Return the Node_Id of the newly created subcomponent if everything went right, else No_Node.

```
function Add_New_Subcomponent
     (Loc                 : Location;
      Name                : Node_Id;
      Comp_Impl           : Node_Id;
      Category            : Ocarina.ME_AADL.Component_Category;
      Is_Refinement       : Boolean := False;
      In_Modes            : Node_Id := No_Node;
      Prototypes_Bindings : List_Id := No_List)
     return Node_Id;
```

6.1.4.10 Ocarina.Builder.AADL.Components.Subprogram_Calls

This package defines the following subprograms:

Add_Property_Association: Add a property association to the subprogram call. Subprogram_Call must reference a subprogram call (not a call sequence). Property_Association references the property association. Return True if everything went right, else False.

```
function Add_Property_Association
     (Subprogram_Call      : Node_Id;
      Property_Association : Node_Id)
     return Boolean;
```

Add_Subprogram_Call: Add a subprogram call to the subprogram call sequence. Subprogram_Call must reference a subprogram call (not a call sequence). Call_Sequence references the subprogram call sequence. Return True if everything went right, else False.

```
function Add_Subprogram_Call      (Call_Sequence   : Node_Id;
      Subprogram_Call : Node_Id)
     return Boolean;
```

Add_New_Subprogram_Call: Create and add a new subprogram call into a subprogram call sequence. Loc is the location of the call sequence in the parsed text. Name references an identifier which contains the name of the subprogram call. Call_Sequence references the subprogram call sequence that contains the subprogram call. The function return the Node_Id of the newly created subprogram call if everything went right, else No_Node.

```
function Add_New_Subprogram_Call      (Loc           : Location;
      Name          : Node_Id;
      Call_Sequence : Node_Id)
     return Node_Id;
```

Add_New_Subprogram_Call_Sequence: Create and add a new subprogram call sequence into a component implementation. Loc is the location of the call sequence in the parsed text. Name references an identifier which contains the name of the call sequence, if any. Comp_Impl references the component implementation. In_Modes contains the list of the modes associated to the connection. Name can be No_Node, if the sequence is not nammed. Subprogram calls Return the Node_Id of the newly created call sequence if everything went right, else No_Node.

```
function Add_New_Subprogram_Call_Sequence
     (Loc       : Location;
      Name      : Node_Id;
```

```
         Comp_Impl : Node_Id;
         In_Modes  : Node_Id := No_Node)
        return Node_Id;
```

6.1.4.11 Ocarina.Builder.AADL.Namespaces

This package defines the following subprograms:

Add_Declaration: Insert any component, property_set, package or port_group into the AADL specification. Namespace must reference the node created with Initialize_Unnamed_Namespace or a package specification. Return True if the element was correctly inserted, else False

```
function Add_Declaration
     (Namespace : Types.Node_Id;
      Element : Types.Node_Id)
     return Boolean;
```

Initialize_Unnamed_Namespace: Create the AADL specification node, which corresponds to the top level of the AADL description. This function must be invoked first, as all the other elements of the description will be added to this one. Loc is the location of the AADL specification in the parsed text. Return a reference to the newly created node if everything went right, else False.

```
function Initialize_Unnamed_Namespace
     (Loc : Locations.Location)
     return Types.Node_Id;
```

Add_New_Package: Checks if a package of that name already exists. If so, return this one, else create a new one and return it. Loc is the location of the package specification in the parsed text. Pack_Name is a Node_Id referencing an identifier which contains the name of the package. Namespace must reference the top level AADL specification node.

```
function Add_New_Package
     (Loc : Locations.Location;
      Pack_Name : Types.Node_Id;
      Namespace : Types.Node_Id)
     return Types.Node_Id;
```

Add_New_Package_Name: Add a property association to the list of the package properties, without checking for homonyms or whatever. This function should be only used by other functions of the core API. Namespace must reference a package specification. Return True if the property was added, else False.

```
function Add_New_Package_Name
     (Loc            : Locations.Location;
      Identifiers    : Types.List_Id)
     return Types.Node_Id;
```

Add_Property_Association: Add a property association to the list of the package properties, without checking for homonyms or whatever. This function should be only used by other functions of the core API. Namespace must reference a package specification. Return True if the property was added, else False.

```
function Add_Property_Association
     (Pack : Types.Node_Id;
      Property_Association : Types.Node_Id)
     return Boolean;
```

Add_New_Name_Visibility_Declaration: Create the name visibility declaration node to the list of the package declarations, without checking.

```
function Add_New_Name_Visibility_Declaration
     (Loc          : Locations.Location;
      Namespace    : Types.Node_Id;
      List_Items   : Types.List_Id;
      Is_Private   : Boolean := False)
     return Types.Node_Id;
```

Add_New_Import_Declaration: Create the import node to the list of the name visibility declarations, without checking.

```
function Add_New_Import_Declaration
     (Loc          : Locations.Location;
      Namespace    : Types.Node_Id;
      List_Items   : Types.List_Id;
      Is_Private   : Boolean := False)
     return Types.Node_Id;
```

Add_New_Alias_Declaration: Create the alias node to the list of the name visibility declarations, without checking.

```
function Add_New_Alias_Declaration
     (Loc             : Locations.Location;
      Namespace       : Types.Node_Id;
      Identifier      : Types.Node_Id;
      Package_Name    : Types.Node_Id;
      Classifier_Ref  : Types.Node_Id;
      Entity_Cat      : Ocarina.ME_AADL.Entity_Category;
      Component_Cat   : Ocarina.ME_AADL.Component_Category;
      Is_All          : Boolean := False;
      Is_Private      : Boolean := False)
     return Types.Node_Id;
```

6.1.4.12 Ocarina.Builder.AADL.Properties

This package defines the following subprograms:

Add_New_Property_Set: Either Single_Value /= No_Node and Mulitple_Values = No_Node, then we have a single valued constant; or Single_Value = No_Node, then we have a muli valued constant

```
function Add_New_Property_Set
     (Loc        : Location;
      Name       : Node_Id;
      Namespace  : Node_Id)
     return Node_Id;
```

Add_New_Property_Constant_Declaration: Either Single_Value /= No_Node and Mulitple_Values = No_Node, then we have a single valued constant; or Single_Value = No_Node, then we have a muli valued constant

```
function Add_New_Property_Constant_Declaration
     (Loc              : Location;
      Name             : Node_Id;
      Property_Set     : Node_Id;
      Constant_Type    : Node_Id;
      Unit_Identifier  : Node_Id;
      Single_Value     : Node_Id;
      Multiple_Values  : List_Id)
     return Node_Id;
```

Add_New_Property_Type_Declaration: Either Applies_To_All is set to True and Applies_To is empty, or Applies_To_All is False and Applies_To is not empty

```
function Add_New_Property_Type_Declaration
     (Loc             : Location;
      Name            : Node_Id;
      Property_Set    : Node_Id;
      Type_Designator : Node_Id)
     return Node_Id;
```

Add_New_Property_Definition_Declaration: Either Applies_To_All is set to True and Applies_To is empty, or Applies_To_All is False and Applies_To is not empty

```
function Add_New_Property_Definition_Declaration
     (Loc                     : Location;
      Name                    : Node_Id;
      Property_Set            : Node_Id;
      Is_Inherit              : Boolean;
      Is_Access               : Boolean;
      Single_Default_Value    : Node_Id;
      Multiple_Default_Value  : List_Id;
      Property_Name_Type      : Node_Id;
      Property_Type_Is_A_List : Boolean;
      Applies_To_All          : Boolean;
      Applies_To              : List_Id)
     return Node_Id;
```

Add_New_Property_Association: If Check_For_Conflicts is set to True, then the function checks whether there is a property association of that name already. If override is set to True and there is a conflict, then it is overridden by the new association. Else the new association is ignored. If Check_For_Conflicts is set to False, then the value of Override is ignored.

```
function Add_New_Property_Association
     (Loc                 : Location;
      Name                : Node_Id;
      Property_Name       : Node_Id;
      Container           : Node_Id;
      In_Binding          : Node_Id;
      In_Modes            : Node_Id;
      Property_Value      : Node_Id;
      Is_Constant         : Boolean;
      Is_Access           : Boolean;
      Is_Additive         : Boolean;
      Applies_To          : List_Id;
      Check_For_Conflicts : Boolean := False;
      Override            : Boolean := False)
     return Node_Id;
```

Add_New_Contained_Element_Path:

```
function Add_New_Contained_Element_Path
     (Loc             : Location;
      Container       : Node_Id;
      Applies_To_Elts : List_Id;
      Annex_Path : Node_Id)
     return Node_Id;
```

6.1.4.13 Ocarina.Analyzer.AADL.Finder

This package provides functions to search nodes in the abstract tree. The functions return No_Node if nothing was found.

This package defines the following subprograms:

Select_Nodes: Build a list (chained using the accessor Next_Entity) from Decl_List and appends it to Last_Node. This list will contain the nodes whose kinds correspond to Kinds.

```
procedure Select_Nodes
     (Decl_List  :        List_Id;
      Kinds      :        Node_Kind_Array;
      First_Node : in out Node_Id;
      Last_Node  : in out Node_Id);
```

Find_Property_Entity: Find a property entity (type, name or constant). If Property_Set_Identifier is No_Node and the current scope is the one of a property set, try to find the property in it. Finally, look for the implicit property sets.

```
function Find_Property_Entity
     (Root                    : Node_Id;
      Property_Set_Identifier : Node_Id;
      Property_Identifier     : Node_Id)
     return Node_Id;
```

Find_Component_Classifier: Same as above, but find a component classifier

```
function Find_Component_Classifier
     (Root                 : Node_Id;
      Package_Identifier   : Node_Id;
      Component_Identifier : Node_Id)
     return Node_Id;
```

Find_Port_Group_Classifier: Same as above, but find a port group

```
function Find_Port_Group_Classifier
     (Root                  : Node_Id;
      Package_Identifier    : Node_Id;
      Port_Group_Identifier : Node_Id)
     return Node_Id;
```

Find_Feature: Find a feature in a component type or implementation

```
function Find_Feature
     (Component          : Node_Id;
      Feature_Identifier : Node_Id)
     return Node_Id;
```

Find_Mode: Same as above, but find a mode

```
function Find_Mode
     (Component       : Node_Id;
      Mode_Identifier : Node_Id)
     return Node_Id;
```

Find_Prototype: XXX

```
function Find_Prototype      (Component              : Node_Id;
      Prototype_Identifier : Node_Id)
     return Node_Id;
```

Find_Subcomponent: Find a subcomponent in a component implementation. If In_Modes is specified, return the subcomponent that are set in the given modes.

```
function Find_Subcomponent
     (Component               : Node_Id;
      Subcomponent_Identifier : Node_Id;
      In_Modes                : Node_Id := No_Node)
     return Node_Id;
```

Find_Subprogram_Call: Same as above but find a subprogram call

```
function Find_Subprogram_Call      (Component       : Node_Id;
      Call_Identifier : Node_Id;
      In_Modes        : Node_Id := No_Node)
     return Node_Id;
```

Find_Connection: Same as above but find a connection

```
function Find_Connection
     (Component             : Node_Id;
      Connection_Identifier : Node_Id;
      In_Modes              : Node_Id := No_Node)
     return Node_Id;
```

Find_Flow_Spec: Find a flow in a component type or implementation

```
function Find_Flow_Spec
     (Component       : Node_Id;
      Flow_Identifier : Node_Id)
     return Node_Id;
```

Find_Subclause: Same as above but find a subclause

```
function Find_Subclause      (Component  : Node_Id;
      Identifier : Node_Id)
     return Node_Id;
```

Find_All_Declarations: Returns the first node of a list of declarations corresponding to the Kinds requested. Following nodes are accessed through the Next_Entity accessor. If no Kinds are requested, then return all the declarations found. If the Namespace is not given, search the declaration in the whole AADL specification declarations and its namespaces. Otherwise, search the declaration in the given namespace.

```
function Find_All_Declarations
     (Root      : Node_Id;
      Kinds     : Node_Kind_Array;
      Namespace : Node_Id := No_Node)
     return Node_List;
```

Find_All_Component_Types: Return the first component type found in the Namespace. If Namespace is No_Node, then return the first component type declaration in the whole AADL specification. Following declarations are accessed using the Next_Entity accessor.

```
function Find_All_Component_Types
     (Root      : Node_Id;
      Namespace : Node_Id := No_Node)
     return Node_List;
```

Find_All_Root_Systems: Return all systems implementations whose component type do not have any feature. Those systems correspond to the roots of the instantiated architecture.

```
function Find_All_Root_Systems (Root : Node_Id) return Node_List;
```

Find_All_Subclauses: General function that returns the first node of a list of subclauses corresponding to the Kinds requested. Following nodes are accessed through the Next_Entity accessor.

```
function Find_All_Subclauses
     (AADL_Declaration : Node_Id;
      Kinds            : Node_Kind_Array)
     return Node_List;
```

Find_All_Features: Applicable to component types and implementations, and port group types.

```
function Find_All_Features
     (AADL_Declaration : Node_Id)
     return Node_List;
```

Find_All_Subclause_Declarations_Except_Properties: Applicable to component types and implementations, and port group types.

```
function Find_All_Subclause_Declarations_Except_Properties
     (AADL_Declaration : Node_Id)
     return Node_List;
```

Find_All_Property_Associations: Applicable to component types and implementations, and port group types.

```
function Find_All_Property_Associations
     (AADL_Declaration : Node_Id)
     return Node_List;
```

Find_Property_Association: Find the property association named Property_Association_Name. Return No_Node if nothing was found.

```
function Find_Property_Association
     (AADL_Declaration          : Node_Id;
      Property_Association_Name : Name_Id)
     return Node_Id;
```

Find_Property_Enumeration: The kind of Package_Container is K_Package_Specification, the kind of Node is K_Identifier or K_ENtity_Reference, return True if the Node is 'with' in Package_Container 'with' declarations.

```
function Find_Property_Enumeration
     (Root               : Node_Id;
      Container          : Node_Id;
      Property_Container : Node_Id;
      Default_Value      : Node_Id;
      Designator        : Node_Id)
     return Node_Id;
```

Find_In_Import_Declaration: The kind of Package_Container is K_Package_Specification, the kind of Node is K_Identifier or K_ENtity_Reference, return True if the Node is 'with' in Package_Container 'with' declarations.

```
function Find_In_Import_Declaration
     (Package_Container : Node_Id;
      Node              : Node_Id)
     return Boolean;
```

6.1.4.14 Ocarina.Analyzer.AADL.Queries

This package contains routines that are used to get several information from the AADL tree.

This package defines the following subprograms:

Is_An_Extension: Returns True if Component is an extension of Ancestor, whether by the keyword 'extends' or because Ancestor is a corresponding component type.

```
function Is_An_Extension
      (Component : Node_Id;
       Ancestor  : Node_Id)
      return Boolean;
```

Needed_By: Return True iff N is needed by Entity (for example Entity has a subcompnent of type N). It also return True if N is needed indirectly by Entity (through another intermediate need). In order for this function to work fine, the AADL tree must have been expanded. However, since it acts only on the AADL syntax tree, this function is put in this package. NOTE: If N is a property *declaration* node, the result will be True reguardless the actual need of Entity to N.

```
function Needed_By (N : Node_Id; Entity : Node_Id) return Boolean;
```

Property_Can_Apply_To_Entity: Return True if the property association Property can be applied to Entity. Otherwise, return False. Beware that this function performs exact verification; a property cannot apply to a package.

```
function Property_Can_Apply_To_Entity
      (Property : Node_Id;
       Entity   : Node_Id)
      return Boolean;
```

Is_Defined_Property: Return True if the property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Is_Defined_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Boolean;
```

Is_Defined_String_Property: Return True if the aadlstring property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Is_Defined_String_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Boolean;
```

Is_Defined_Integer_Property: Return True if the aadlinteger property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Is_Defined_Integer_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Boolean;
```

Is_Defined_Boolean_Property: Return True if the aadlboolean property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Is_Defined_Boolean_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
```

```
          return Boolean;
```

Is_Defined_Float_Property: Return True if the aadlreal property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_Float_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Is_Defined_Reference_Property: Return True if the component reference property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_Reference_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Is_Defined_Classifier_Property: Return True if the component classifier property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_Classifier_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Is_Defined_Range_Property: Return True if the component range property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_Range_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Is_Defined_List_Property: Return True if the 'list of XXX' property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_List_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Is_Defined_Enumeration_Property: Return True if the enumeration property named 'Name' is defined for the AADL entity 'Entity'. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
     function Is_Defined_Enumeration_Property
          (Entity  : Node_Id;
           Name    : Name_Id;
           In_Mode : Name_Id := No_Name)
          return Boolean;
```

Get_Property_Association: Return the property association node corresponding to Name. If the propoerty designed by name is not present for Entity, return No_Node. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Property_Association
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Node_Id;
```

Get_Value_Of_Property_Association: Return the value of the property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return No_Node. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Value_Of_Property_Association
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Node_Id;
```

Get_String_Property: Return the value of the aadlstring property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return `""`. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_String_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return String;
```

Get_String_Property: Return the value of the aadlstring property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return No_Name. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_String_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Name_Id;
```

Get_Integer_Property: Return the value of the aadlinteger property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return 0. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Integer_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Unsigned_Long_Long;
```

Get_Float_Property: Return the value of the aadlreal property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return 0.0. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Float_Property
      (Entity  : Node_Id;
       Name    : Name_Id;
       In_Mode : Name_Id := No_Name)
      return Long_Long_Float;
```

Get_Boolean_Property: Return the value of the aadlboolean property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return False. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Boolean_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return Boolean;
```

Get_Reference_Property: Return the value of the component reference property association named 'Name' if it is defined for 'Entity'. Otherwise, return No_Node. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Reference_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return Node_Id;
```

Get_Classifier_Property: Return the value of the component classifier property association named 'Name' if it is defined for 'Entity'. Otherwise, return No_Node. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Classifier_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return Node_Id;
```

Get_List_Property: Return the value of the 'list of XXX' property association named 'Name' if it is defined for 'Entity'. The returned list is a Node_Container list. Otherwise, return No_List. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_List_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return List_Id;
```

Get_Range_Property: Return the values of the range property association named 'Name' if it is defined for 'Entity'. Otherwise, return No_List. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Range_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return Node_Id;
```

Get_Enumeration_Property: Return the value of the enumeration property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return `""`. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Enumeration_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return String;
```

Get_Enumeration_Property: Return the value of the enumeration property association named 'Name' if it is defined defined for 'Entity'. Otherwise, return No_Name. If 'In_Mode' is a valid name, consider only the property defined for the given mode.

```
function Get_Enumeration_Property
     (Entity  : Node_Id;
      Name    : Name_Id;
      In_Mode : Name_Id := No_Name)
     return Name_Id;
```

Compute_Property_Value: Compute the value of a property value and return a Node_Id containing this value. This value does not contain any reference (value ()).

```
function Compute_Property_Value (Property_Value : Node_Id) return Node_Id;
```

6.1.5 API to build and manipulate AADL instances

This section presents the functions that allow the manipulation of AADL instances.

The routines that create the instance tree from the model tree are located in the `Ocarina.Instances` package. The main function that should be used is the `Ocarina.Instances.Instantiate_Model` function. It Instantiates the tree of the model and returns the instance architecture. The first parameter given to this function is the root of the model tree. The second parameter designs the root system implementation, used when several system implementations are electable as root system.

6.1.6 Core parsing and printing facilities

6.1.6.1 Parser

The main parsing function is the `Ocarina.Parser.Parse` function. This function selects automatically the right parser depending on the file suffix: If the file suffix is ".aadl", then the parsing function used is `Ocarina.AADL.Parser.Process`. Therefore, the input files given to the Parse function must have valid suffixes.

The return value of the `Ocarina.Parser.Parse` function is the node corresponding to the root of the tree. If something went wrong during the parsing, the return value is `No_Node`. The first parameter given to the `Ocarina.Parser.Parse` function is the name of the file to parse. The second parameter corresponds to the tree root, this way, it's possible to parse many files by calling the function several times and by giving to it the same root as second parameter. After each call, the returned value is the old tree to which are added the AADL entities of the last parsed file. Hence Ocarina supports multiple file AADL descriptions.

6.1.6.2 Printer

Unlike the parsing facility, the printing facility cannot be selected automatically. The user must precise which printer he wants to use.

The data structure `Ocarina.Printer.Output_Options` contains a field named `Printer_Name` which allows to select a registered printer. There are other fields in this structure, they allow to configure some printing options (output file, output directory...).

6.1.6.3 Using the parser and the printer

The user should not directly use the parsing and printing subprogram supplied by each module. To parse a file, the function that should be used is `Ocarina.Parser.Parse`. To print a file, the user must specify the printer options in a variable of type `Ocarina.Printer.Output_Options`, then call the `Ocarina.Printer.Print` function with the Root of the AADL tree and the options variable as parameters. Here is a sample code that shows how to initialize Ocarina, parse and print a set of AADL files. The example describes a classical way to use the Ocarina libraries to build a basic program that loops indefinitely and parses at each iteration all the AADL files given to its command line. If the parsing has been successful, the program prints the AADL sources corresponding to the built AADL syntax tree. The example illustrates also the *Reset* capabilities of Ocarina allowing to reinitialize all the Ocarina engines at the end of an iteration in order to use them in the incoming iteration.

```
------------------------------------------------------------------------------
--                                                                          --
--                           OCARINA COMPONENTS                             --
--                                                                          --
--                P A R S E _ A N D _ P R I N T _ A A D L                   --
--                                                                          --
--                                B o d y                                   --
--                                                                          --
--               Copyright (C) 2007-2008, GET-Telecom Paris.                --
--                                                                          --
-- Ocarina  is free software;  you  can  redistribute  it and/or  modify    --
-- it under terms of the GNU General Public License as published by the     --
-- Free Software Foundation; either version 2, or (at your option) any      --
-- later version. Ocarina is distributed  in  the  hope  that it will be    --
-- useful, but WITHOUT ANY WARRANTY;  without even the implied warranty of  --
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General --
-- Public License for more details. You should have received  a copy of the --
-- GNU General Public License distributed with Ocarina; see file COPYING.   --
-- If not, write to the Free Software Foundation, 51 Franklin Street, Fifth --
-- Floor, Boston, MA 02111-1301, USA.                                       --
--                                                                          --
-- As a special exception,  if other files  instantiate  generics from this --
-- unit, or you link  this unit with other files  to produce an executable, --
-- this  unit  does not  by itself cause  the resulting  executable to be   --
-- covered  by the  GNU  General  Public  License. This exception does not  --
-- however invalidate  any other reasons why the executable file might be   --
-- covered by the GNU Public License.                                       --
--                                                                          --
--                Ocarina is maintained by the Ocarina team                 --
--                      (ocarina-users@@listes.enst.fr)                      --
--                                                                          --
------------------------------------------------------------------------------

with Ada.Command_Line;
with GNAT.OS_Lib;

with Types;
with Namet;

with Ocarina.Configuration;
with Ocarina.Parser;
with Ocarina.Printers;
with Ocarina.Analyzer;

procedure Parse_And_Print_AADL is

   use Types;
   use Namet;

   Root              : Node_Id;
   Printer_Options   : Ocarina.Printers.Output_Options :=
     Ocarina.Printers.Default_Output_Options;
   Analysis_Options  : constant Ocarina.Analyzer.Analyzer_Options :=
     Ocarina.Analyzer.Default_Analyzer_Options;
   Success           : Boolean;

begin
```

```
     loop
        --  Initialization step

        Ocarina.Initialize;
        Ocarina.Configuration.Init_Modules;

        --  Parse the aadl source file, the right parser is selected
        --  automatically depending on the file suffix. It is important
        --  that the root node is set to No_Node at the very beginning
        --  or parsing.

        Root := No_Node;

        for J in 1 .. Ada.Command_Line.Argument_Count loop
           --  Parse the file corresponding to the Jth argument opf the
           --  commant line and enrich the global AADL tree.

           Root := Ocarina.Parser.Parse (Ada.Command_Line.Argument (J), Root);
        end loop;

        --  If something went wrong, Root = No_Node

        if Root /= No_Node then
           --  Analyze the tree

           Success := Ocarina.Analyzer.Analyze_Model (Root, Analysis_Options);

           if Success then
              --  Select the printer

              Set_Str_To_Name_Buffer ("aadl");
              Printer_Options.Printer_Name := Name_Find;

              --  Print to an AADL File

              Success := Ocarina.Printers.Print
                (Root    => Root,
                 Options => Printer_Options);
           else
              GNAT.OS_Lib.OS_Exit (1);
           end if;
        else
           GNAT.OS_Lib.OS_Exit (1);
        end if;

        --  Pause for 1 second

        delay 1.0;

        --  Reset step

        Ocarina.Configuration.Reset_Modules;
        Ocarina.Reset;
     end loop;

  end Parse_And_Print_AADL;
```

6.2 Input/Output Modules

Ocarina can handle AADL source files written conforming to the AADL grammar.

- **AADL** The `Print_Node` procedure provided by this module allows to print a node and its subnodes. The result is an AADL source file. The printer name associated to this module is `aadl`.

- **Dumper** This module is used for debugging purpose. Its printing facility allows to dump the AADL tree or the AADL instance tree. The printer names associated to this module are `aadl_tree_p` for the parsed tree and `aadl_tree_e` for the instance tree (instance tree).

7 Petri Net Mapping Rules

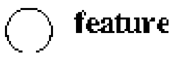
Ocarina lets you generate Petri nets from AADL descriptions. This way, it is possible to achieve verification on AADL architectures before generating the corresponding source code.

7.1 Mapping Patterns

The AADL elements to map into Petri nets are the software components. Indeed, execution platform components are used to model the deployment of the software components; such deployment information is not to in the scope of Petri nets. AADL threads and AADL subprograms are the most important components, since they can host subprogram call sequences: they describe the actual execution flows in the architecture. AADL processes and systems are actually boxes containing threads or other components, hence they do not provide any "active" semantics; data components are not active components either. The following table lists the main rules of the mapping:

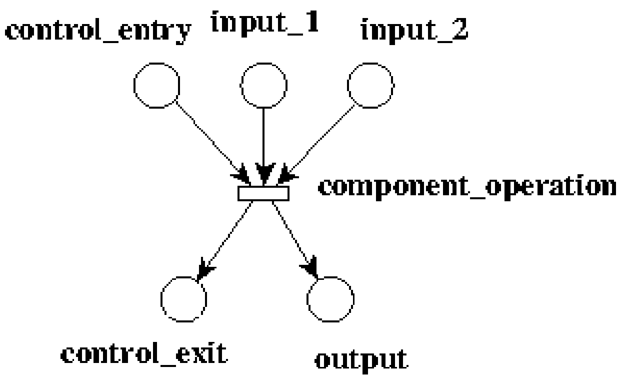
7.1.1 Component Features

```
feature : in data port;
```



7.1.2 Subprograms

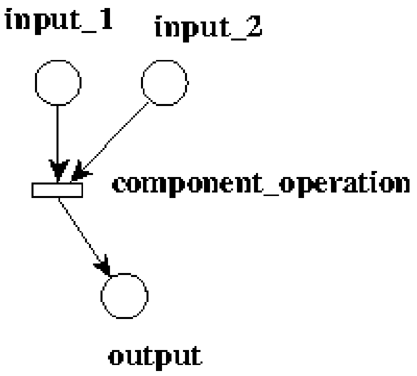
```
subprogram a_subprogram
features
  input_1 : in parameter;
  input_2 : in parameter;
  output : out parameter;
end a_subprogram;
```



7.1.3 Other Components

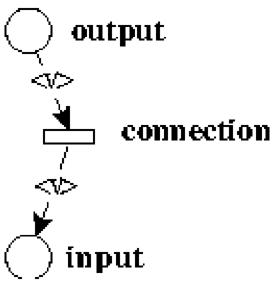
```
process a_process
features
  input_1 : in data port;
  input_2 : in data port;
  output : out data port;
```

```
end a_process;
```



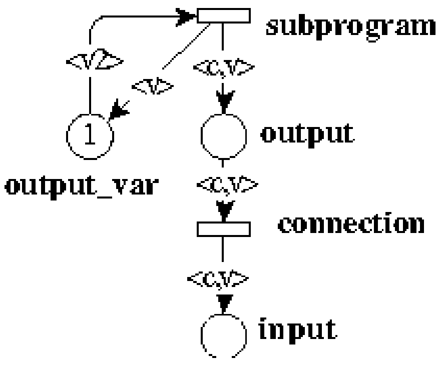
7.1.4 Connections

```
connection : data port output -> input;
```



7.1.5 Subprogram Connections

```
connection : parameter output -> input;
```



This mapping mainly consists of translating the AADL execution flows into Petri nets. Components that do not have any subcomponents nor call sequences are modeled by a transition that consumes inputs and produces outputs. Component features are modeled by places. Tokens stored in input features are to be consumed by the components or connections; tokens produced by component or connection transitions are stored in output features. Components that have subcomponents are modeled by merging the transition with the models of the subcomponents.

We model a place per feature. This systematic approach help the user identify the translation between AADL models and corresponding Petri nets. In addition, it facilitates the expansions of the feature places. For example, we might want to describe the queue protocols defined by the AADL properties: in this case we would replace each place by Petri nets modeling FIFOs or whatever type of queue is specified by the AADL properties.

Connections between features are modeled by transitions. We distinguish connections between subprograms parameters and between other component ports.

If an AADL port is connected to several other ports at a time, the Petri net transition shall be connected to all the corresponding places: a token will be sent to each target place, thus modeling the fact that each destination port receives the output of the initial port.

Connections of subprograms parameters are slightly more complex. Indeed, output places of subprograms model variables, that can be read many times. Therefore, a subprogram produces two tokens: one that carries the output value and the control information, and an extra one that only carries the value. The extra token is stored in a place where subprograms from other call sequences can get it—the transition shall put the token back into the place. In order to ensure the correct replacement of the token whenever a new value is produced by the subprogram, the subprogram itself consumes its old extra output token to produce the new one.

Call sequences are made of subprograms that are connected. We use an extra token to model the execution control. There is a single execution control token in each thread or subprogram, thus reflecting the fact that there is no concurrency in call sequences, and in threads and subprograms in general.

7.2 Examples

Like code generation, the Petri net mapping does not directly handle behavioral description: such descriptions must be provided using AADL properties and then merged with the generated net.

Here is an example of an AADL model:

```
thread micro_broker end micro_broker;

thread implementation micro_broker.receiver
subcomponents
  buffer : data data_buffer;
calls
  listen : {get_data : subprogram protocol.simple;};
  compute : {execute : subprogram execution.simple;};
connections
  data access buffer -> get_data.buffer;
  data access buffer -> execute.buffer;
properties
  Dispatch_Protocol => background;
  Ocarina::formal_implementation => "broker.pn";
end micro_broker.receiver;

data data_buffer end data_buffer;

subprogram protocol
features
  buffer : requires data access data_buffer;
end protocol;

subprogram implementation protocol.simple
calls
  {get_data : subprogram transport;};
connections
  data access buffer -> get_data.buffer;
end protocol.simple;

subprogram transport
features
  buffer : requires data access data_buffer;
end transport;
```

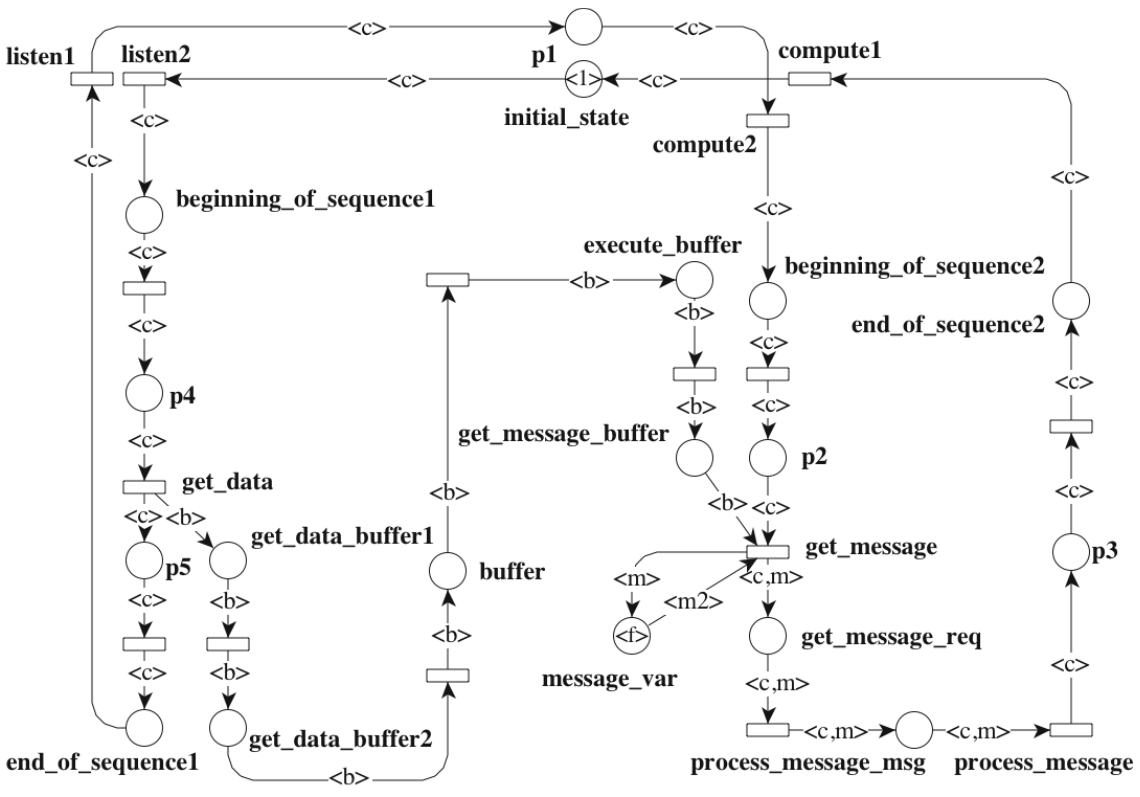
```
subprogram execution
features
  req : in parameter message;
end execution;

subprogram implementation execution.simple
calls
  {get_message : subprogram arguments;
    process_message : subprogram application;};
connections
  data access buffer -> get_message.buffer;
  parameter get_message.req -> process_message.msg;
end execution.simple;

subprogram arguments
features
  req : out parameter message;
  buffer : requires data access data_buffer;
end arguments;

subprogram application
features
  msg : in parameter message;
end application;
```

By applying the mapping, we obtain the following Petri net:



8 AADL modes for Emacs and vim

The AADL modes for Emacs and vim provide syntax coloration and automatic indentation features when editing AADL files.

8.1 Emacs

To load the AADL mode for Emacs, you need to add the following line to your emacs configuration file (usually located in '`~/.emacs`') :

```
(load "/path/to/this/file.el")
```

For more details on this mode, please refer to the emacs contextual help.

8.2 vim

The AADL mode for vim is made of two files '`aadl.vim`': one for syntactic coloration, and the other for indentation. The file for indentation must be placed into '`~/.vim/indent/`' while the one for syntactic coloration must be placed into '`~/.vim/syntax/`'

To load the AADL mode whenever you edit AADL files, create a file named '`~/.vim/filetype.vim`', in which you write:

```
augroup filetypedetect
        au BufNewFile,BufRead *.aadl    setf aadl
augroup END
```

For more details, please read the documentation of vim.

Appendix A Standard AADL property files

A.1 AADL Project

```
--****************************************************
--   AADL Standard AADL_V1.0
--  Appendix A (normative)
--  Predeclared Property Sets
--  03Nov04
--  Revised 14May06
--****************************************************

property set AADL_Project is

  Default_Active_Thread_Handling_Protocol : constant
     Supported_Active_Thread_Handling_Protocols => abort;
  -- one of the choices of Supported_Active_Thread_Handling_Protocols.

  Supported_Active_Thread_Handling_Protocols: type enumeration
    (abort,
     complete_one_flush_queue,
     complete_one_transfer_queue,
     complete_one_preserve_queue,
     complete_all);
  -- a subset may be supported.

  Supported_Connection_Protocols: type enumeration
    (HTTP,
     HTTPS,
     UDP,
     IP_TCP);
  -- The following are example protocols.
  -- (HTTP, HTTPS, UDP, IP_TPC);

  Supported_Concurrency_Control_Protocols: type enumeration
    (NoneSpecified,
     Read_Only,
     Protected_Access,
     Immediate_Priority_Ceiling_Protocol,
     Priority_Ceiling_Protocol,
     Priority_Ceiling);
  -- phf : NoneSpecified instead of None
  -- The following are example concurrency control protocols.
  -- (Interrupt_Masking, Maximum_Priority, Priority_Inheritance,
  --  Priority_Ceiling)

  Supported_Dispatch_Protocols: type enumeration
    (Periodic,
     Aperiodic,
     Sporadic,
     Hybrid,
     Background);
  -- The following are protocols for which the semantics are defined.
  -- (Periodic, Sporadic, Aperiodic, Hybrid, Background);

  Supported_Hardware_Source_Languages: type enumeration
    (VHDL);
  -- The following is an example hardware description language.
  -- (VHDL)

  -- phf A26: added
  Supported_Queue_Processing_Protocols: type enumeration
    (FIFO);
```

```
-- The Supported_Queue_Processing_Protocols property enumeration
-- type specifies the set of queue processing protocols that are
-- supported.

Supported_Scheduling_Protocols: type enumeration
  (PARAMETRIC_PROTOCOL,
   EARLIEST_DEADLINE_FIRST_PROTOCOL,
   LEAST_LAXITY_FIRST_PROTOCOL,
   RATE_MONOTONIC_PROTOCOL,
   DEADLINE_MONOTONIC_PROTOCOL,
   ROUND_ROBIN_PROTOCOL,
   TIME_SHARING_BASED_ON_WAIT_TIME_PROTOCOL,
   POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL,
   D_OVER_PROTOCOL,
   MAXIMUM_URGENCY_FIRST_BASED_ON_LAXITY_PROTOCOL,
   MAXIMUM_URGENCY_FIRST_BASED_ON_DEADLINE_PROTOCOL,
   TIME_SHARING_BASED_ON_CPU_USAGE_PROTOCOL,
   NO_SCHEDULING_PROTOCOL,
   HIERARCHICAL_CYCLIC_PROTOCOL,
   HIERARCHICAL_ROUND_ROBIN_PROTOCOL,
   HIERARCHICAL_FIXED_PRIORITY_PROTOCOL,
   HIERARCHICAL_PARAMETRIC_PROTOCOL);
-- The following are example scheduling protocols.
-- (RMS, EDF, Sporadicserver, SlackServer, ARINC653)

Supported_Source_Languages: type enumeration
  (Ada95,
   Ada,     -- alias for Ada95
   Ada05,   -- alias for Ada95
   ASN1,
   C,
   GUI,
   Lustre,
   Lustre5, -- alias for Lustre
   Lustre6, -- alias for Lustre
   RTDS,    -- alias for SDL_RTDS
   SDL_RTDS,
   RTSJ,    -- Real Time Specification for Java
   SCADE6,  -- alias for Lustre
   SDL,     -- alias for SDL_ObjectGeode
   SDL_ObjectGeode,
   Simulink,
   Simulink_6_5,
   System_C,
   blackbox_device,
   VHDL);

Max_Aadlinteger: constant aadlinteger => 2#1#e32;

Max_Base_Address: constant aadlinteger => 512;

Max_Memory_Size: constant aadlinteger Size_Units => 2#1#e32 B;

Max_Queue_Size: constant aadlinteger => 512;

Max_Thread_Limit: constant aadlinteger => 32;

Max_Time: constant aadlinteger Time_Units => 1000 hr;

Max_Urgency: constant aadlinteger => 12;

Max_Word_Count: constant aadlinteger => 2#1#e32;

Max_Word_Space: constant aadlinteger => 64;
```

```
  Size_Units : type units (
    Bits,
    B     => Bits  * 8,
    KB    => B     * 1000,
    MB    => KB    * 1000,
    GB    => MB    * 1000);

  Time_Units : type units (
    ps,
    Ns    => ps  * 1000,
    Us    => Ns  * 1000,
    Ms    => Us  * 1000,
    Sec   => Ms  * 1000,
    Min   => Sec * 60,
    Hr    => Min * 60);

end AADL_Project;
```

A.2 AADL Properties

```
--*****************************************************
--   AADL Standard AADL_V1.0
--  Appendix A (normative)
--  Predeclared Property Sets
--  03Nov04
--  Update to reflect current standard on 28Mar06
--*****************************************************

property set AADL_Properties is

  ---------------------------------------------------
  ---------------------------------------------------
  Activate_Deadline: Time
  applies to (thread);
  ---------------------------------------------------

  Activate_Execution_Time: Time_Range
  applies to (thread);
  ---------------------------------------------------

  Activate_Entrypoint: aadlstring
  applies to (thread);
  ---------------------------------------------------

  Active_Thread_Handling_Protocol: inherit
    Supported_Active_Thread_Handling_Protocols
  => value(Default_Active_Thread_Handling_Protocol)
  applies to (thread, thread group, process, system);
  ---------------------------------------------------

  Active_Thread_Queue_Handling_Protocol: inherit enumeration
    (flush,
     hold)
  => flush
  applies to (thread, thread group, process, system);
  ---------------------------------------------------

  Actual_Connection_Binding: inherit list of reference
    (bus,
     processor,
     device)
  applies to
    (port connections,
     port group connections,  --  ENST: Added
     thread,
     thread group,
     process,
     system);
  ---------------------------------------------------

  Actual_Latency: Time
  applies to (flow);
  ---------------------------------------------------

  Actual_Memory_Binding: inherit reference (memory)
  applies to (thread, thread group, process, system, processor,
  data port, event data port, subprogram);
  ---------------------------------------------------

  Actual_Processor_Binding: inherit reference (processor)
  applies to (thread, thread group, process, system);
  ---------------------------------------------------
```

```
Actual_Subprogram_Call: reference (server subprogram)
applies to (subprogram);
----------------------------------------------------

Actual_Subprogram_Call_Binding: inherit list of reference
  (bus,
   processor,
   memory,
   device)
applies to (subprogram);
----------------------------------------------------

Actual_Throughput: Data_Volume
applies to (flow);
----------------------------------------------------

Aggregate_Data_Port: aadlboolean => false
applies to (port group);
----------------------------------------------------

Allowed_Access_Protocol: list of enumeration
  (Memory_Access,
   Device_Access)
applies to (bus);
----------------------------------------------------

Allowed_Connection_Binding: inherit list of reference
  (bus,
   processor,
   device)
applies to (port connections, thread group, process, system);
----------------------------------------------------

Allowed_Connection_Binding_Class: inherit list of classifier
  (processor,
   bus,
   device)
applies to (port connections, thread, thread group, process, system);
----------------------------------------------------

Allowed_Connection_Protocol: list of enumeration
  (Data_Connection,
   Event_Connection,
   Event_Data_Connection,
   Data_Access_Connection,
   Server_Subprogram_Call)
applies to (bus, device);
----------------------------------------------------

Allowed_Dispatch_Protocol: list of Supported_Dispatch_Protocols
applies to (processor);
----------------------------------------------------

Allowed_Memory_Binding: inherit list of reference
  (memory,
   system,
   processor)
applies to (thread, thread group, process, system, device, data port,
event data port, subprogram, processor);
----------------------------------------------------

Allowed_Memory_Binding_Class: inherit list of classifier
  (memory,
   system,
   processor)
applies to (thread, thread group, process, system, device, data port,
event data port, subprogram, processor);
----------------------------------------------------

Allowed_Message_Size: Size_Range
applies to (bus);
----------------------------------------------------

Allowed_Period: list of Time_Range
applies to (processor, system);
----------------------------------------------------

Allowed_Processor_Binding: inherit list of reference
  (processor,
   system)
applies to (thread, thread group, process, system, device);
----------------------------------------------------

Allowed_Processor_Binding_Class: inherit list of classifier
  (processor,
   system)
applies to (thread, thread group, process, system, device);
----------------------------------------------------

Allowed_Subprogram_Call: list of reference
  (server subprogram)
applies to (subprogram);
----------------------------------------------------

Allowed_Subprogram_Call_Binding: inherit list of reference
  (bus,
   processor,
   device)
applies to (subprogram, thread, thread group, process, system);
----------------------------------------------------

Assign_Time: Time
applies to (processor, bus);
----------------------------------------------------

Assign_Byte_Time: Time
applies to (processor, bus);
----------------------------------------------------

Assign_Fixed_Time: Time
applies to (processor, bus);
----------------------------------------------------

Available_Memory_Binding: inherit list of reference
  (memory,
   system)
applies to (system);
----------------------------------------------------

Available_Processor_Binding: inherit list of reference
  (processor,
   system)
applies to (system);
----------------------------------------------------

Base_Address: access aadlinteger 0 .. value(Max_Base_Address)
applies to (memory);
----------------------------------------------------
```

```
Client_Subprogram_Execution_Time: Time
applies to (subprogram);
----------------------------------------------------

Clock_Jitter: Time
applies to (processor, system);
----------------------------------------------------

Clock_Period: Time
applies to (processor, system);
----------------------------------------------------

Clock_Period_Range: Time_Range
applies to (processor, system);
----------------------------------------------------

Compute_Deadline: Time
applies to (thread, device, subprogram, event port, event data port);
----------------------------------------------------

Compute_Entrypoint: aadlstring
applies to (thread, subprogram, event port, event data port);
----------------------------------------------------

Compute_Execution_Time: Time_Range
applies to (thread, device, subprogram, event port, event data port);
----------------------------------------------------

Concurrency_Control_Protocol: Supported_Concurrency_Control_Protocols
  => NoneSpecified
applies to (data);
----------------------------------------------------

Connection_Protocol: Supported_Connection_Protocols
applies to (connections);
----------------------------------------------------

Data_Volume: type aadlinteger 0 bitsps .. value(Max_Aadlinteger)
units
  (bitsps,
   Bps    => bitsps * 8,
   Kbps   => Bps    * 1000,
   Mbps   => Kbps   * 1000,
   Gbps   => Mbps   * 1000 );
----------------------------------------------------

Deactivate_Deadline: Time
applies to (thread);
----------------------------------------------------

Deactivate_Execution_Time: Time_Range
applies to (thread);
----------------------------------------------------

Deactivate_Entrypoint: aadlstring
applies to (thread);
----------------------------------------------------

Deadline: inherit Time => value(Period)
applies to (thread, thread group, process, system, device);
----------------------------------------------------

Dequeue_Protocol: enumeration
  (OneItem,
   AllItems)
   => OneItem
applies to (event port, event data port);
----------------------------------------------------

Device_Dispatch_Protocol: Supported_Dispatch_Protocols
  => Aperiodic
applies to (device);
----------------------------------------

Device_Register_Address: aadlinteger
applies to (port, port group);
----------------------------------------------------

Dispatch_Protocol: Supported_Dispatch_Protocols
applies to (thread);
----------------------------------------------------

Expected_Latency: Time
applies to (flow);
----------------------------------------------------

Expected_Throughput: Data_Volume
applies to (flow);
----------------------------------------------------

Finalize_Deadline: Time
applies to (thread);
----------------------------------------------------

Finalize_Execution_Time: Time_Range
applies to (thread);
----------------------------------------------------

Finalize_Entrypoint: aadlstring
applies to (thread);
----------------------------------------------------

Hardware_Description_Source_Text: inherit list of aadlstring
applies to (memory, bus, device, processor, system);
----------------------------------------------------

Hardware_Source_Language: Supported_Hardware_Source_Languages
applies to (memory, bus, device, processor, system);
----------------------------------------------------

Initialize_Deadline: Time
applies to (thread);
----------------------------------------------------

Initialize_Execution_Time: Time_Range
applies to (thread);
----------------------------------------------------

Initialize_Entrypoint: aadlstring
applies to (thread);
----------------------------------------------------

Latency: Time
applies to (flow, connections);
----------------------------------------------------

Load_Deadline: Time
applies to (process, system);
----------------------------------------------------
```

```
Load_Time: Time_Range
applies to (process, system);
----------------------------------------------------

Memory_Protocol: enumeration
  (read_only,
   write_only,
   read_write)
   => read_write
applies to (memory);
----------------------------------------------------

Not_Collocated: list of reference
  (data,
   thread,
   process,
   system,
   connections)
applies to (data, thread, process, system, connections);
----------------------------------------------------

Overflow_Handling_Protocol: enumeration
  (DropOldest,
   DropNewest,
   Error)
   => DropOldest
applies to (event port, event data port, subprogram);
----------------------------------------------------

Period: inherit Time
applies to (thread, thread group, process, system, device);
----------------------------------------------------

Process_Swap_Execution_Time: Time_Range
applies to (processor);
----------------------------------------------------

Propagation_Delay: Time_Range
applies to (bus);
----------------------------------------------------

Provided_Access : access enumeration
  (read_only,
   write_only,
   read_write,
   by_method)
   => read_write
applies to (data, bus);
----------------------------------------------------

Queue_Processing_Protocol: Supported_Queue_Processing_Protocols
  => FIFO
applies to (event port, event data port, subprogram);
----------------------------------------------------

Queue_Size: aadlinteger 0 .. value(Max_Queue_Size)
  => 0
applies to (event port, event data port, subprogram);
----------------------------------------------------

Read_Time: list of Time_Range
applies to (memory);
----------------------------------------------------

Recover_Deadline: Time
```

```
applies to (thread, server subprogram);
----------------------------------------------------

Recover_Execution_Time: Time_Range
applies to (thread, server subprogram);
----------------------------------------------------

Recover_Entrypoint: aadlstring
applies to (thread);
----------------------------------------------------

Required_Access : access enumeration
  (read_only,
   write_only,
   read_write,
   by_method)
   => read_write
applies to (data, bus);
----------------------------------------------------

Required_Connection : aadlboolean => true
applies to (port);
----------------------------------------------------

Runtime_Protection : inherit aadlboolean => true
applies to (process, system);
----------------------------------------------------

Scheduling_Protocol: list of Supported_Scheduling_Protocols
applies to (processor);
----------------------------------------------------

Server_Subprogram_Call_Binding: inherit list of reference
  (thread,
   processor)
applies to (subprogram, thread, thread group, process, system);
----------------------------------------------------
Size: type aadlinteger 0 B .. value (Max_Memory_Size) units Size_Units;

--  OLD DECLARATION:
--  Size: type aadlinteger 0 B .. value (Max_Memory_Size);
--  This is wrong according to the AADL standard 1.0 page 150:

--  "An aadlinteger property type represents an integer value or an
--  integer value and its measurement unit.  If an units clause is
--  present, then the value is a pair of values, and unit may only
--  be one of the enumeration literals specified in the units
--  clause. *IF AN UNITS CLAUSE IS ABSENT, THEN THE VALUE IS AN
--  INTEGER VALUE*. If a simple range is present, then the integer
--  value must be an element of the specified range"

----------------------------------------------------
Size_Range: type range of Size;
----------------------------------------------------

Source_Code_Size: Size
applies to (data, thread, thread group, process, system, subprogram,
processor, device);
----------------------------------------------------

Source_Data_Size: Size
applies to (data, subprogram, thread, thread group, process, system,
processor, device);
----------------------------------------------------
```

```
Ada_Package_Name: aadlstring
applies to (data);
----------------------------------------------------

Source_Heap_Size: Size
applies to (thread, subprogram);
----------------------------------------------------

Source_Language: inherit Supported_Source_Languages
applies to (subprogram, data, thread, thread group, process, bus,
device, processor, system);
----------------------------------------------------

Source_Name: aadlstring
applies to (data, port, subprogram, parameter);
----------------------------------------------------

Source_Stack_Size: Size
applies to (thread, subprogram, processor, device);
----------------------------------------------------

Source_Text: inherit list of aadlstring
applies to (data, port, subprogram, thread, thread group, process,
system, memory, bus, device, processor, parameter, port group);
----------------------------------------------------

Startup_Deadline: inherit Time
applies to (processor, system);
----------------------------------------------------

Subprogram_Execution_Time: Time_Range
applies to (subprogram);
----------------------------------------------------

Supported_Source_Language: list of Supported_Source_Languages
applies to (processor, system);
----------------------------------------------------

Synchronized_Component: inherit aadlboolean => true
applies to (thread, thread group, process, system);
----------------------------------------------------

Thread_Limit: aadlinteger 0 .. value(Max_Thread_Limit)
  => value(Max_Thread_Limit)
applies to (processor);
----------------------------------------------------

Thread_Swap_Execution_Time: Time_Range
applies to (processor, system);
----------------------------------------------------

Throughput: Data_Volume
applies to (flow, connections);
----------------------------------------------------

Time: type aadlinteger 0 ps .. value(Max_Time) units Time_Units;

--  OLD DECLARATION:
--  Time: type aadlinteger 0 ps .. value(Max_Time);
--  This is wrong according to the AADL standard 1.0 page 150:

--  "An aadlinteger property type represents an integer value or an
--  integer value and its measurement unit.  If an units clause is
--  present, then the value is a pair of values, and unit may only
--  be one of the enumeration literals specified in the units
```

```
  --  clause. *IF AN UNITS CLAUSE IS ABSENT, THEN THE VALUE IS AN
  --  INTEGER VALUE*. If a simple range is present, then the integer
  --  value must be an element of the specified range"

  -----------------------------------------------------

  Time_Range: type range of Time;
  -----------------------------------------------------

  Transmission_Time: list of Time_Range
  applies to (bus);
  -----------------------------------------------------

  Type_Source_Name: aadlstring
  applies to (data, port, subprogram);
  -----------------------------------------------------

  Urgency: aadlinteger 0 .. value(Max_Urgency)
  applies to (port);
  ---------------------------------------------------

  Word_Count: aadlinteger 0 .. value(Max_Word_Count)
  applies to (memory);
  ---------------------------------------------------

  Word_Size: Size => 8 bits
  applies to (memory);
  ---------------------------------------------------

  Word_Space: aadlinteger 1 .. value(Max_Word_Space) => 1
  applies to (memory);
  ---------------------------------------------------

  Write_Time: list of Time_Range
  applies to (memory);
  ---------------------------------------------------
  ---------------------------------------------------
end AADL_Properties;
```

Appendix B Ocarina AADL property files

Ocarina includes specific property files to specify some elements of the models that are not covered by AADL 1.0.

B.1 Deployment

```
property set Deployment is

  Allowed_Transport_APIs : type enumeration
    (BSD_Sockets,
     SpaceWire);
  -- Supported transport API

  Transport_API : Deployment::Allowed_Transport_APIs applies to (bus);
  --  Transport API of a bus

  Location : aadlstring applies to (processor, device);
  --  Processor IP address (BSD_Sockets specific)

  Port_Number : aadlinteger applies to (process, device);
  --  IP port number of a process (BSD_Sockets specific)

  Process_ID : aadlinteger applies to (process, device);
  --  Process identifier (SpaceWire specific)

  Channel_Address : aadlinteger applies to (process, device);
  --  Communication channel address (SpaceWire specific)

  Protocol_Type : type enumeration (iiop, diop);
  --  Supported communication protocols
  Protocol : Deployment::Protocol_Type applies to (system);

  Allowed_Execution_Platform : type enumeration
   (Native,                    -- Native platforms (GNU/Linux, Solaris, Windows...)
    Native_Compcert,           -- Native platforms using the Compcert compiler
    bench,                     -- Benchmark platform (native with instrumentation).
    LEON_RTEMS,                -- LEON2 board or tsim-leon (RTEMS)
    LEON_RTEMS_POSIX,          -- LEON2 board or tsim-leon (RTEMS)
    LEON3_SCOC3,               -- LEON3 with RTEMS for SCOC3
    LEON3_XTRATUM,             -- LEON3 with Xtratum
    LEON3_XM3,                 -- RTEMS for XTRATUM/LEON3
    LEON_ORK,                  -- LEON2 board or tsim-leon (ORK)
    LEON_GNAT,                 -- LEON2 board or qemu (GNATPRO/HI-E)
    LINUX32,                   -- Linux 32 bits
    LINUX32_XENOMAI_NATIVE,    -- Linux 32 bits with native Xenomai
    LINUX32_XENOMAI_POSIX,     -- Linux 32 bits with Xenomai and POSIX skin
    LINUX64,                   -- Linux 64 bits
    ERC32_ORK,                 -- ERC32 board or tsim-erc32 (ORK)
    ARM_DSLINUX,               -- Nintendo DS (tm) (DSLinux)
    ARM_N770,                  -- Nokia N770 (tm)
    GUMSTIX_RTEMS,             -- Gumstix under RTEMS
    NDS_RTEMS,                 -- Nintendo DS under RTEMS
    X86_RTEMS,                 -- x86 under RTEMS
    X86_RTEMS_POSIX,           -- x86 under RTEMS with POSIX layer
    X86_LINUXTASTE,            -- TASTE-specific linux distribution
    MARTE_OS,                  -- MaRTE OS
    WIN32,                     -- WIN32
    VXWORKS                    -- VXWORKS
    );
  --  Supported platforms

  Execution_Platform : Deployment::Allowed_Execution_Platform
```

```
    applies to (all);
  --  Execution platform of a processor

  Supported_Execution_Platform : list of Deployment::Allowed_Execution_Platform
    applies to (device);
  --  List execution platforms supported by a particular driver

  Runtime : type enumeration
    (PolyORB_HI_C,
     PolyORB_HI_Ada,
     POK);
  --  List of supported runtime

  Supported_Runtime : Deployment::Runtime applies to (all);
  --  List the runtime compatible with the component

  Priority_Type : type aadlinteger 0 .. 255;

  Priority : Deployment::Priority_Type applies to (data, thread);
  --  Thread and data component priority

  Driver_Name : aadlstring applies to (device);

  Configuration : aadlstring applies to (device);

  Config : aadlstring applies to (device);

  ASN1_Module_Name : aadlstring applies to (all);

  Help : aadlstring applies to (all);

  Version : aadlstring applies to (all);

  Configuration_Type : classifier (data) applies to (all);

end Deployment;
```

B.2 Ocarina_Config

```
--  Property set containing the configuration properties of Ocarina.
--  This property set is not intended to be used by the AADL model of
--  an application, but, by the AADL model of its scenario.

property set Ocarina_Config is

  Generator_Type : type enumeration
   (PolyORB_QoS_Ada,
    PolyORB_HI_Ada,
    PolyORB_HI_C,
    PolyORB_HI_RTSJ,
    POK_C,
    Xtratum_Configuration,
    Petri_Nets);

  Generator : Ocarina_Config::Generator_Type applies to (system);
  --  The code generator that will be used for the current application

  Generator_Options_Type : type enumeration
   (gprof,
    ASN1);

  Generator_Options : list of Ocarina_Config::Generator_Options_Type applies to (system);
  --  Code generation options.

  AADL_Files : list of aadlstring applies to (system);
  --  List of the AADL source files used by the current application

  Data_Model           : constant aadlstring => "Data_Model";
  Deployment           : constant aadlstring => "Deployment";
  Cheddar_Properties   : constant aadlstring => "Cheddar_Properties";
  POK_Properties       : constant aadlstring => "pok_properties";
  ARINC653_Properties  : constant aadlstring => "arinc653_properties";
  ASSERT_Properties    : constant aadlstring => "ASSERT_Properties";
  TASTE_Properties     : constant aadlstring => "taste_properties";
  --  List of the predefined NON STANDARD property sets that can be used
  --  by an AADL application.

  Needed_Property_Sets : list of aadlstring applies to (system);
  --  The actual property sets needed by one particular application.
  --  This avoid to parse systematically all the predefined non
  --  standard property sets. The user can also give the name of a
  --  custom property set (which may be used by many AADL models),
  --  provided that the value of the string matches exactly the base
  --  name (without the .aadl suffix and in a case-sensitive manner)
  --  the user property sey file name and provided that this property
  --  set file is located in the same directory as the Ocarina
  --  non-standard property sets.

  Root_System_Name : aadlstring applies to (system);
  --  If present, indicates the name of the root of the instance tree

  AADL_Version_Type : type enumeration (AADLv1, AADLv2);

  AADL_Version : Ocarina_Config::AADL_Version_Type applies to (system);
  --  AADL version of the model

  Use_Components_Library : aadlboolean applies to (system);

  Referencial_Files : list of aadlstring applies to (system);
  --  The list of referencial files used to compute the regression test
  Timeout_Property : Time applies to (system);
  --  The timeout used to stop an execution
end Ocarina_Config;
```

Appendix C Conformance to standards

Ocarina front-end supports the parsing and semantic analysis of all AADL 1.0 concepts.

Code generators restricts AADL 1.0 semantics to ensure the model can lead to satisfactory code generation, see the corresponding documentation for more details.

Appendix D 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.

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

Index