CHAPTER 4
DEFINITION OF THE DIANA OPERATIONS

Recall that DIANA is an abstract data type. By the nature of an abstract data type as implemented in a programming language, all that need be known about the type are the functions and procedures that operate on objects of the type. Thus to realize the abstract type DIANA in some programming language, all that is needed is to write those functions and procedures. In a language like ADA it is possible to separate the *specification* of these functions and procedures from their *implementation*.

In this chapter we provide an ADA specification (but not implementation) of the interface to the necessary functions and procedures to define DIANA. Further, we suggest how, in general, an implementation-specific package may be derived from an IDL definition. Since the derivation of packages from an IDL description is a complex topic, we only sketch one possible derivation for one particular language. A detailed discussion of the package derivation process is given in the IDL Formal Description [9].

4.1. The DIANA Operations

Every object of type DIANA is the representation of some specific ADA program (or portion of an ADA program). Specifically, it may be thought of as the output from passing that program through the Front End of an ADA compiler. A minimum set of operations on the DIANA type must include the following functions and procedures:

type_getter — Such a function permits the user to determine of a given object what its type is. In DIANA terms, if an object is known to belong to some specific node class, the function determines the object's node type.

selector — Such a function returns the value of a specific attribute of a node.

constructor — Such a procedure builds a node from its constituent parts, or changes the value of an attribute of a node.

In addition, operators are necessary to determine the *equality* of DIANA objects. Specifically, are a given pair of instances of a DIANA type in fact the *same*

instance, as opposed to *equivalent* ones[1]? In case there are variables of this abstract data type, an *assignment* operator is necessary as well.

4.2. DIANA's Use of Other Abstract Data Types

An IDL definition (such as the definition of DIANA in Chapter 2) is built upon subsidiary abstract data types. These include those used in the IDL notation (such as `Integer`, `Boolean`, `Seq OF`) as well as implementation-defined attribute types (such as **source_position**, **symbol_rep**, and so on). All of these except `Seq OF` have the same operations as described above. It must be carefully noted that for the scalar types (`Integer`, `Boolean`) there is usually no distinction drawn between equality and equivalence. Whenever doing so is necessary, we carefully draw such a distinction.

The sequence type `Seq OF` can be considered as a built-in type that has a few special operators. Specifically, there must be a way to check if a sequence is empty and to fetch items from a sequence. Additionally, there must be operators for adding and removing items from a sequence.

The implementation defined types must have all the operations appropriate to them as well as those described above for attributes and nodes.

4.3. Summary of Operators

This section summarizes the operations described above.

The operations on nodes are

- create a node;
- fetch the value of an attribute of a given node;
- set the value of an attribute of a given node;
- compare two nodes to see if they are the same node; and
- assign a specific node to a variable.

The operators defined for the IDL sequence type (an ordered list of nodes of the same class) are

- create a sequence of a given type;
- select an element of a sequence;
- add an element to a sequence;

---

[1] This distinction is addressed further in Section 3.9 on page 123.

- remove an element from a sequence;
- compare two sequences to see if they are the same sequence; and
- assign a sequence to a variable of sequence type.

The operators required for the IDL scalar types (`Integer`, `Rational`, and `Boolean`) are

- create a scalar;
- compare two scalars to see if they are equal (*i.e.*, the same scalar); and
- assignment.

4.4. General Method for Deriving a Package Specification for DIANA

To derive a general package specification for defining this abstract data type called DIANA, further decisions concerning the implementation model need to be made. For example, one must decide how to represent the various DIANA objects. After these decisions have been made, a straightforward process can be applied to derive the package specification from the DIANA domain. A formal method for specifying these decisions is presented in the IDL formal description. Indeed, an IDL tool would produce such a package automatically from the definition of DIANA in Chapter 2. For the purposes of this document, the following discussion is sufficient.

The *implementation model* must deal with two separate areas of concern. First, there are the implementation restrictions imposed by the choice of the source language that the DIANA type is being implemented in. Secondly, there is the choice of corresponding entities in the implementation language for entities in the DIANA domain (*i.e.*, how DIANA objects are represented). These decisions can be driven by the design considerations of tools that expect to use the DIANA type, as well as by specific restrictions of the host system.

The general steps are as follows:

- *representation of IDL types.* An implementation for each of the IDL types must be chosen. Normally for the scalar types, the implementation language supports an equivalent (or close enough) abstract type. For the sequence type and the implementation defined types, the same decisions need to be made, and an abstract data type for these derived and specified. (The DIANA domain specification provides a handle on the abstract data types for the implementation defined types.)

- *representation of node classes.* The class names of the DIANA language must be handled by the package derivation process, because

the types of the attributes are defined using these meta-variables.

- *representation of nodes.* The node representation choice must permit attribute values to be associated with the node, since each specific instance of a node may have different attribute values.

- *method of defining operators.* The operators in the language must be specified either as functions and procedures in the implementation language or by equating them to specific operations already in the implementation language.

4.5. Deriving a Specific ADA Package

To derive a specific ADA package, we apply the general method as outlined in the previous section. First, we choose an implementation model of an abstract data type defined as a single package. A single ADA **private** type is used to define all nodes in the DIANA domain. All operations are calls on procedures or functions specified in the package. Having made these decisions, we then address the following points:

- *representation of IDL types.* The IDL `Boolean` type could be implemented directly by the ADA BOOLEAN predefined type. However, the IDL `Integer` and `Rational` types would have to be represented somehow so as to be able to represent arbitrarily large quantities, and (in the case of rationals) to represent them exactly with no approximation[2]
  Using the ADA predefined types INTEGER and FLOAT would not be adequate.

  For the sequence type `Seq of`, we include a private type definition and primitive operations. The operations permit creation of an empty sequence (**Make**), functions to add an element at the beginning (**Insert**) and end (**Append**) of a sequence, and functions for selecting the first element of a sequence (**Head**) and the remainder of a sequence (**Tail**). There is also a function to determine if a list is empty (**Is_Empty**). Note that additional functions and procedures for this type could be added.

- *representation of node classes.* Since a single type is being used to represent all nodes in the domain, the distinction between different classes is not necessary.

- *representation of nodes.* A single private type (called **Tree**) is provided for all the node names defined in the DIANA domain. An enumerated type (called **Node_Name**) is defined which provides a

[2] These requirements are spelled out in the Ada LRM, which requires that some arithmetic performed at compile time be done exactly.

name for all the various nodes defined in the DIANA domain. An additional function (named **Kind** and returning a result of type **Node_Name**) is added to the **Tree** type to distinguish between different node kinds.

- *method of defining operators*. The create operator for the various nodes becomes a single function that takes a **Node_Name** and returns a new **Tree** node with most of its attributes not defined. Each of the DIANA attributes has a corresponding procedure and function in the package specification that respectively modify and fetch the value of an attribute. The procedure and function both take the specific **Tree** node as an argument. The procedure takes an additional argument which gives the new value for the attribute; the function returns the corresponding attribute value.

The *comparison* operators for the nodes and for sequences are the built-in ADA comparison operators ('=', '/=') which are defined for private types. The comparison operators for the scalar types are not defined in this package. The ADA language provides all *create* operations for the scalar types. The *assignment* operators are the pre-defined ADA assignment operators for variables of the private types. Except for these assumptions on the use of built-in operations, the full ADA package is given.

A few facts are important:

- Because some of the DIANA node types conflict with ADA reserved words, we choose to prefix all **node_names** with the prefix "**dn_**" (short for DIANA).

- Remember that this specification defines a minimal set of operations; implementations may augment it with other useful ones for particular applications.

- We have added an additional type (**ARITIES**) and several procedures and functions (**ARITY**, **Son1**, **Son2**, and **Son3**) which are mentioned in the ADA Formal Definition and which are very useful in the tree traversals essential to many phases of compilers, as well as other tools.

4.6. The DIANA Package in ADA

A summary of essential points of the ADA package specification for DIANA appears in Figure 4-1 on page 134. For ease of understanding, the figure contains only as much of the package as fits onto one page.

The package defines and makes available the following types, functions, and procedures:

```
package Diana is
    type Tree is private;                    -- a Diana node
    type SEQ_TYPE is private;                -- sequence of nodes

    type NODE_NAME is                        -- enumeration class for node names
     (  ...                                  -- about 160 different node types
      );

    -- Tree constructors.
    function  MAKE     (c: in NODE_NAME)      return TREE;
    procedure DESTROY  (t: in TREE);

    function  KIND     (t: in TREE)           return NODE_NAME;

    -- Tree traversers from the Ada Formal Definition.

    type ARITIES is (nullary, unary, binary, ternary, arbitrary);

    function  ARITY    (t: in TREE)           return ARITIES;
    function  SON1     (t: in TREE)           return TREE;
    procedure SON1     (t: in out TREE; v: in TREE);
    function  SON2     (t: in TREE)           return TREE;
    procedure SON2     (t: in out TREE; v: in TREE);
    function  SON3     (t: in TREE)           return TREE;
    procedure SON3     (t: in out TREE; v: in TREE);

    -- Handling of list constructs.
    function  HEAD     (l: in SEQ_TYPE)       return TREE;      -- LISP CAR
    function  TAIL     (l: in SEQ_TYPE)       return SEQ_TYPE;  -- LISP CDR
    function  MAKE                            return SEQ_TYPE;
                                                 -- return empty list
    function  IS_EMPTY (l: in SEQ_TYPE)       return BOOLEAN;
    function  INSERT (l: in out SEQ_TYPE;
                      i: in TREE)             return SEQ_TYPE;
                                                 -- inserts i at start of l
    function  APPEND (l: in out SEQ_TYPE;
                      i: in TREE)             return SEQ_TYPE;
                                                 -- inserts i at end of l

    -- Handling of LIST attribute of list constructs.
    procedure LIST     (t: in out TREE; v: in SEQ_TYPE);
    function  LIST     (t: in TREE)           return SEQ_TYPE;

    -- Structural Attributes.

    procedure AS_ACTUAL            (t: in out TREE; v: in TREE);
    function  AS_ACTUAL            (t: in TREE) return TREE ; -- assoc
    ...

    -- followed by functions and procedures for about 100 attributes .....

private

    -- To be filled in...

end Diana;
```

Figure 4-1: Sketch of the DIANA Package

type TREE — An object of this private type is a node of the DIANA structure.

type SEQ_TYPE — An object of this private type is a sequence of nodes of the same class.

type NODE_NAME — This is an enumeration type providing an enumeration literal for each kind of DIANA node.

function MAKE — This function creates and returns a DIANA node of the kind which is its argument. Note that it is overloaded so as also to be able to create an empty list.

procedure DESTROY — This procedure indicates that a node is no longer required.

function KIND — Given a node, this function returns its node-kind.

type ARITIES — This enumeration type provides a literal for each number of structural children a node might have.

function SON_k — For k = 1, 2, 3, each such function returns the k^{th} offspring of a node.

procedure SON_k — For k = 1, 2, 3, each such procedure stores a new k^{th} offspring of the node.

list processing — A collection of functions and procedures implement the usual list-processing primitives.

attributes — For each possible attribute, there is a function to return the value of that attribute at a node, and a procedure to store a new value for the attribute.

A complete listing of the entire DIANA package specification concludes this chapter.

```
with USERPK; use USERPK;
-- Package USERPK provides the following items (see page 77):

-- source_position:   Defines source position in original source program.
--                    Used for error messages.
-- symbol_rep:        Representation of identifiers, strings and characters.
-- value:             Implementation defined.
--                        Gives value of an expression.
--                        Can indicate that no value is computed
-- operator:          Enumeration type for all operators.
-- number_rep:        Representation of numeric literals.
-- comments:          Representation of comments from source program.

package Diana is

   type TREE is private;

   type SEQ_TYPE is private;
```

```
type NODE_NAME is
  (  dn_abort,                dn_accept,              dn_access,
     dn_address,              dn_aggregate,           dn_alignment,
     dn_all,                  dn_allocator,           dn_alternative,
     dn_alternative_s,        dn_and_then,            dn_argument_id,
     dn_array,                dn_assign,              dn_assoc,
     dn_attr_id,              dn_attribute,           dn_attribute_call,
     dn_binary,               dn_block,               dn_box,
     dn_case,                 dn_choice_s,            dn_code,
     dn_comp_id,              dn_comp_rep,            dn_comp_rep_s,
     dn_comp_unit,            dn_compilation,         dn_cond_clause,
     dn_cond_entry,           dn_const_id,            dn_constant,
     dn_constrained,          dn_context,             dn_conversion,
     dn_decl_s,               dn_def_char,            dn_def_op,
     dn_deferred_constant,    dn_delay,               dn_derived,
     dn_dscrmt_aggregate,     dn_dscrmt_id,           dn_dscrmt_var,
     dn_dscrmt_var_s,         dn_dscrt_range_s,       dn_entry,
     dn_entry_call,           dn_entry_id,            dn_enum_id,
     dn_enum_literal_s,       dn_exception,           dn_exception_id,
     dn_exit,                 dn_exp_s,               dn_fixed,
     dn_float,                dn_for,                 dn_formal_dscrt,
     dn_formal_fixed,         dn_formal_float,        dn_formal_integer,
     dn_function,             dn_function_call,       dn_function_id,
     dn_generic,              dn_generic_assoc_s,     dn_generic_id,
     dn_generic_param_s,      dn_goto,                dn_id_s,
     dn_if,                   dn_in,                  dn_in_id,
     dn_in_op,                dn_in_out,              dn_in_out_id,
     dn_index,                dn_indexed,             dn_inner_record,
     dn_instantiation,        dn_integer,             dn_item_s,
     dn_iteration_id,         dn_label_id,            dn_labeled,
     dn_loop,                 dn_l_private,           dn_l_private_type_id,
     dn_membership,           dn_name_s,              dn_named,
     dn_named_stm,            dn_named_stm_id,        dn_no_default,
     dn_not_in,               dn_null_access,         dn_null_comp,
     dn_null_stm,             dn_number,              dn_number_id,
     dn_numeric_literal,      dn_or_else,             dn_others,
     dn_out,                  dn_out_id,              dn_package_body,
     dn_package_decl,         dn_package_id,          dn_package_spec,
     dn_param_assoc_s,        dn_param_s,             dn_parenthesized,
     dn_pragma,               dn_pragma_id,           dn_pragma_s,
     dn_private,              dn_private_type_id,     dn_proc_id,
     dn_procedure,            dn_procedure_call,      dn_qualified,
     dn_raise,                dn_range,               dn_record,
     dn_record_rep,           dn_rename,              dn_return,
     dn_reverse,              dn_select,              dn_select_clause,
     dn_select_clause_s,      dn_selected,            dn_simple_rep,
     dn_slice,                dn_stm_s,               dn_string_literal,
     dn_stub,                 dn_subprogram_body,     dn_subprogram_decl,
     dn_subtype,              dn_subtype_id,          dn_subunit,
     dn_task_body,            dn_task_body_id,        dn_task_decl,
     dn_task_spec,            dn_terminate,           dn_timed_entry,
     dn_type,                 dn_type_id,             dn_universal_fixed,
     dn_universal_integer,    dn_universal_real,      dn_use,
     dn_used_bltn_id,         dn_used_bltn_op,        dn_used_char,
     dn_used_name_id,         dn_used_object_id,      dn_used_op,
     dn_var,                  dn_var_id,              dn_variant,
     dn_variant_part,         dn_variant_s,           dn_void,
     dn_while,                dn_with
   );
```

```
-- Tree constructors.

function  MAKE      (c: in NODE_NAME)       return TREE;
procedure DESTROY   (t: in TREE);
function  KIND      (t: in TREE)            return NODE_NAME;

-- Tree traversers from the Ada Formal Definition.

type ARITIES is (nullary, unary, binary, ternary, arbitrary);

function  ARITY     (t: in TREE)            return ARITIES;
function  SON1      (t: in TREE)            return TREE;
procedure SON1      (t: in out TREE; v: in TREE);
function  SON2      (t: in TREE)            return TREE;
procedure SON2      (t: in out TREE; v: in TREE);
function  SON3      (t: in TREE)            return TREE;
procedure SON3      (t: in out TREE; v: in TREE);


-- Handling of list constructs.

function  HEAD    (l: in SEQ_TYPE)       return TREE;      -- LISP CAR
function  TAIL    (l: in SEQ_TYPE)       return SEQ_TYPE;  -- LISP CDR
function  MAKE                           return SEQ_TYPE;
                                                  -- return empty list
function  IS_EMPTY (l: in SEQ_TYPE)      return BOOLEAN;
function  INSERT (l: in out SEQ_TYPE;
                  i: in TREE)            return SEQ_TYPE;
                                                  -- inserts i at start of l
function  APPEND (l: in out SEQ_TYPE;
                  i: in TREE)            return SEQ_TYPE;
                                                  -- inserts i at end of l
```

```
-- Handling of LIST attribute of list constructs.

procedure  LIST     (t: in out TREE; v: in SEQ_TYPE);
function   LIST     (t: in TREE)          return SEQ_TYPE;
                        -- aggregate          has Seq Of COMP_ASSOC
                        -- alternative_s       has Seq Of ALTERNATIVE
                        -- choice_s            has Seq Of CHOICE
                        -- compilation         has Seq Of COMP_UNIT
                        -- comp_rep_s          has Seq Of COMP_REP
                        -- context             has Seq Of CONTEXT_ELEM
                        -- decl_s              has Seq Of DECL
                        -- dscrmt_aggregate    has Seq Of COMP_ASSOC
                        -- dscrt_range_s       has Seq Of DSCRT_RANGE
                        -- enum_literal_s      has Seq Of ENUM_LITERAL
                        -- exp_s               has Seq Of EXP
                        -- generic_assoc_s     has Seq Of GENERIC_ASSOC
                        -- generic_param_s     has Seq Of GENERIC_PARAM
                        -- id_s                has Seq Of ID
                        -- if                  has Seq Of COND_CLAUSE
                        -- inner_record        has Seq Of COMP
                        -- item_s              has Seq Of ITEM
                        -- name_s              has Seq Of NAME
                        -- param_assoc_s       has Seq Of PARAM_ASSOC
                        -- param_s             has Seq Of PARAM
                        -- pragma_id           has Seq Of ARGUMENT
                        -- pragma_s            has Seq Of PRAGMA
                        -- record              has Seq Of COMP
                        -- select_clause_s     has Seq Of SELECT_CLAUSE
                        -- stm_s               has Seq Of STM
                        -- use                 has Seq Of NAME
                        -- variant_s           has Seq Of VARIANT
                        -- var_s               has Seq Of VAR
                        -- with                has Seq Of NAME

-- Structural Attributes.

procedure AS_ACTUAL           (t: in out TREE; v: in TREE);
function  AS_ACTUAL           (t: in TREE) return TREE ; -- assoc
procedure AS_ALIGNMENT        (t: in out TREE; v: in TREE);
function  AS_ALIGNMENT        (t: in TREE) return TREE ; -- record_rep
procedure AS_ALTERNATIVE_S    (t: in out TREE; v: in TREE);
function  AS_ALTERNATIVE_S    (t: in TREE) return TREE ; -- case
                                                         -- block
procedure AS_BINARY_OP        (t: in out TREE; v: in TREE);
function  AS_BINARY_OP        (t: in TREE) return TREE ; -- binary
procedure AS_BLOCK_STUB       (t: in out TREE; v: in TREE);
function  AS_BLOCK_STUB       (t: in TREE) return TREE ; -- package_body |
                                                         -- task_body |
                                                         -- subprogram_body
procedure AS_CHOICE_S         (t: in out TREE; v: in TREE);
function  AS_CHOICE_S         (t: in TREE) return TREE ; -- alternative |
                                                         -- named
                                                         -- variant
procedure AS_COMP_REP_S       (t: in out TREE; v: in TREE);
function  AS_COMP_REP_S       (t: in TREE) return TREE ; -- record_rep
procedure AS_CONSTRAINED      (t: in out TREE; v: in TREE);
function  AS_CONSTRAINED      (t: in TREE) return TREE ; -- access | derived
                                                         -- array | subtype
procedure AS_CONSTRAINT       (t: in out TREE; v: in TREE);
function  AS_CONSTRAINT       (t: in TREE) return TREE ; -- constrained
procedure AS_CONTEXT          (t: in out TREE; v: in TREE);
function  AS_CONTEXT          (t: in TREE) return TREE ; -- comp_unit
```

```
procedure AS_DECL_S              (t: in out TREE; v: in TREE);
function  AS_DECL_S              (t: in TREE) return TREE ; -- task_spec
procedure AS_DECL_S1             (t: in out TREE; v: in TREE);
function  AS_DECL_S1             (t: in TREE) return TREE ; -- package_spec
procedure AS_DECL_S2             (t: in out TREE; v: in TREE);
function  AS_DECL_S2             (t: in TREE) return TREE ; -- package_spec
procedure AS_DESIGNATOR          (t: in out TREE; v: in TREE);
function  AS_DESIGNATOR          (t: in TREE) return TREE ; -- subprogram_body
                                                            -- subprogram_decl
                                                            -- assoc | generic
procedure AS_DESIGNATOR_CHAR     (t: in out TREE; v: in TREE);
function  AS_DESIGNATOR_CHAR     (t: in TREE) return TREE ; -- selected
procedure AS_DSCRMT_VAR_S        (t: in out TREE; v: in TREE);
function  AS_DSCRMT_VAR_S        (t: in TREE) return TREE ; -- type
procedure AS_DSCRT_RANGE         (t: in out TREE; v: in TREE);
function  AS_DSCRT_RANGE         (t: in TREE) return TREE ; -- for | reverse
                                                            -- slice
procedure AS_DSCRT_RANGE_S       (t: in out TREE; v: in TREE);
function  AS_DSCRT_RANGE_S       (t: in TREE) return TREE ; -- array
procedure AS_DSCRT_RANGE_VOID    (t: in out TREE; v: in TREE);
function  AS_DSCRT_RANGE_VOID    (t: in TREE) return TREE ; -- entry
procedure AS_EXCEPTION_DEF       (t: in out TREE; v: in TREE);
function  AS_EXCEPTION_DEF       (t: in TREE) return TREE ; -- exception
procedure AS_EXP                 (t: in out TREE; v: in TREE);
function  AS_EXP                 (t: in TREE) return TREE ; -- delay | case
                                                            -- fixed | float
                                                            -- membership | while
                                                            -- address | assign
                                                            -- code | conversion
                                                            -- named | number
                                                            -- qualified
                                                            -- simple_rep
                                                            -- unary | comp_rep
                                                            -- parenthesized
procedure AS_EXP1                (t: in out TREE; v: in TREE);
function  AS_EXP1                (t: in TREE) return TREE ; -- binary | range
procedure AS_EXP2                (t: in out TREE; v: in TREE);
function  AS_EXP2                (t: in TREE) return TREE ; -- range
                                                            -- binary
procedure AS_EXP_CONSTRAINED     (t: in out TREE; v: in
TREE);
function  AS_EXP_CONSTRAINED     (t: in TREE) return TREE ; -- allocator
procedure AS_EXP_S               (t: in out TREE; v: in TREE);
function  AS_EXP_S               (t: in TREE) return TREE ; -- indexed
                                                            -- attribute_call
procedure AS_EXP_VOID            (t: in out TREE; v: in TREE);
function  AS_EXP_VOID            (t: in TREE) return TREE ; -- return
                                                            -- cond_clause
                                                            -- in | in_out | exit
                                                            -- out | record_rep
```

```
procedure AS_GENERIC_ASSOC_S   (t: in out TREE; v: in TREE);
function  AS_GENERIC_ASSOC_S   (t: in TREE) return TREE ; -- instantiation
procedure AS_GENERIC_HEADER    (t: in out TREE; v: in TREE);
function  AS_GENERIC_HEADER    (t: in TREE) return TREE ; -- generic
procedure AS_GENERIC_PARAM_S   (t: in out TREE; v: in TREE);
function  AS_GENERIC_PARAM_S   (t: in TREE) return TREE ; -- generic
procedure AS_HEADER            (t: in out TREE; v: in TREE);
function  AS_HEADER            (t: in TREE) return TREE ; -- subprogram_body
                                                          -- subprogram_decl
procedure AS_ID                (t: in out TREE; v: in TREE);
function  AS_ID                (t: in TREE) return TREE ; -- for | attribute
                                                          -- labeled | reverse
                                                          -- named_stm
                                                          -- package_body
                                                          -- package_decl
                                                          -- subtype
                                                          -- task_body
                                                          -- variant_part
                                                          -- type | task_decl
                                                          -- pragma
procedure AS_ID_S              (t: in out TREE; v: in TREE);
function  AS_ID_S              (t: in TREE) return TREE ; -- exception
                                                          -- number | constant
                                                          -- in | in_out
                                                          -- out | var
procedure AS_ITEM_S            (t: in out TREE; v: in TREE);
function  AS_ITEM_S            (t: in TREE) return TREE ; -- block
procedure AS_ITERATION         (t: in out TREE; v: in TREE);
function  AS_ITERATION         (t: in TREE) return TREE ; -- loop
procedure AS_MEMBERSHIP_OP     (t: in out TREE; v: in TREE);
function  AS_MEMBERSHIP_OP     (t: in TREE) return TREE ; -- membership
procedure AS_NAME              (t: in out TREE; v: in TREE);
function  AS_NAME              (t: in TREE) return TREE ; -- accept | address
                                                          -- procedure_call
                                                          -- all | comp_rep
                                                          -- constrained
                                                          -- indexed
                                                          -- instantiation
                                                          -- goto | index
                                                          -- qualified
                                                          -- selected
                                                          -- rename | slice
                                                          -- variant_part
                                                          -- attribute_call
                                                          -- entry_call
                                                          -- record_rep
                                                          -- allocator
                                                          -- assign
                                                          -- attribute | code
                                                          -- conversion
                                                          -- function_call
                                                          -- simple_rep
                                                          -- subunit
```

```
procedure AS_NAME_S             (t: in out TREE; v: in TREE);
function  AS_NAME_S             (t: in TREE) return TREE ; -- abort
                                                           -- with | use
procedure AS_NAME_VOID          (t: in out TREE; v: in TREE);
function  AS_NAME_VOID          (t: in TREE) return TREE ; -- raise | exit
procedure AS_OBJECT_DEF         (t: in out TREE; v: in TREE);
function  AS_OBJECT_DEF         (t: in TREE) return TREE ; -- constant | var
procedure AS_PACKAGE_DEF        (t: in out TREE; v: in TREE);
function  AS_PACKAGE_DEF        (t: in TREE) return TREE ; -- package_decl
procedure AS_PARAM_ASSOC_S      (t: in out TREE; v: in TREE);
function  AS_PARAM_ASSOC_S      (t: in TREE) return TREE ; -- procedure_call
                                                           -- entry_call
                                                        -- pragma
                                                        -- function_call
procedure AS_PARAM_S            (t: in out TREE; v: in TREE);
function  AS_PARAM_S            (t: in TREE) return TREE ; -- procedure
                                                        -- function
                                                        -- entry | accept
procedure AS_PRAGMA_S           (t: in out TREE;  v: in TREE);
function  AS_PRAGMA_S           (t: in TREE) return TREE ; -- comp_unit
procedure AS_RANGE              (t: in out TREE; v: in TREE);
function  AS_RANGE              (t: in TREE) return TREE ; -- integer
                                                        -- comp_rep
procedure AS_RANGE_VOID         (t: in out TREE; v: in TREE);
function  AS_RANGE_VOID         (t: in TREE) return TREE ; -- fixed | float
procedure AS_RECORD             (t: in out TREE; v: in TREE);
function  AS_RECORD             (t: in TREE) return TREE ; -- variant
procedure AS_SELECT_CLAUSE_S    (t: in out TREE; v: in TREE);
function  AS_SELECT_CLAUSE_S    (t: in TREE) return TREE ; -- select
procedure AS_STM                (t: in out TREE; v: in TREE);
function  AS_STM                (t: in TREE) return TREE ; -- labeled
                                                        -- named_stm
procedure AS_STM_S              (t: in out TREE; v: in TREE);
function  AS_STM_S              (t: in TREE) return TREE ; -- alternative
                                                        -- cond_clause
                                                        -- loop | select
                                                        -- accept | block
procedure AS_STM_S1             (t: in out TREE; v: in TREE);
function  AS_STM_S1             (t: in TREE) return TREE ; -- cond_entry
                                                        -- timed_entry
procedure AS_STM_S2             (t: in out TREE; v: in TREE);
function  AS_STM_S2             (t: in TREE) return TREE ; -- cond_entry
                                                        -- timed_entry
procedure AS_SUBPROGRAM_DEF     (t: in out TREE; v: in TREE);
function  AS_SUBPROGRAM_DEF     (t: in TREE) return TREE ; -- subprogram_decl
procedure AS_SUBUNIT_BODY       (t: in out TREE; v: in TREE);
function  AS_SUBUNIT_BODY       (t: in TREE) return TREE ; -- subunit
procedure AS_TASK_DEF           (t: in out TREE; v: in TREE);
function  AS_TASK_DEF           (t: in TREE) return TREE ; -- task_decl
procedure AS_TYPE_RANGE         (t: in out TREE; v: in TREE);
function  AS_TYPE_RANGE         (t: in TREE) return TREE ; -- membership
procedure AS_TYPE_SPEC          (t: in out TREE; v: in TREE);
function  AS_TYPE_SPEC          (t: in TREE) return TREE ; -- constant | in
                                                        -- in_out | out
                                                        -- var
                                                        -- type
procedure AS_UNIT_BODY          (t: in out TREE; v: in TREE);
function  AS_UNIT_BODY          (t: in TREE) return TREE ; -- comp_unit
procedure AS_VARIANT_S          (t: in out TREE; v: in TREE);
function  AS_VARIANT_S          (t: in TREE) return TREE ; -- variant_part
```

```
-- Lexical Attributes.

procedure LX_COMMENTS          (t: in out TREE;  v: comments);
function  LX_COMMENTS          (t: in TREE)  return comments ;
procedure LX_DEFAULT           (t: in out TREE;  v: Boolean);
function  LX_DEFAULT           (t: in TREE)  return Boolean;
procedure LX_NUMREP            (t: in out TREE;  v: number_rep);
function  LX_NUMREP            (t: in TREE)  return number_rep ;
procedure LX_PREFIX            (t: in out TREE;  v: Boolean);
function  LX_PREFIX            (t: in TREE)  return Boolean;
procedure LX_SRCPOS            (t: in out TREE;  v: source_position);
function  LX_SRCPOS            (t: in TREE)  return source_position ;
procedure LX_SYMREP            (t: in out TREE;  v: symbol_rep);
function  LX_SYMREP            (t: in TREE)  return symbol_rep ;


-- Semantic Attributes.

procedure SM_ACTUAL_DELTA   (t: in out TREE;  v: Float);
function  SM_ACTUAL_DELTA   (t: in TREE)  return Float;
procedure SM_ADDRESS        (t: in out TREE;  v: in TREE);
                                          -- v: EXP_VOID
function  SM_ADDRESS        (t: in TREE)  return TREE ;
                                          -- returns EXP_VOID
procedure SM_BASE_TYPE      (t: in out TREE;  v: in TREE);
                                          -- v: TYPE_SPEC
function  SM_BASE_TYPE      (t: in TREE)  return TREE ;
                                          -- returns TYPE_SPEC
procedure SM_BITS           (t: in out TREE;  v: Integer);
function  SM_BITS           (t: in TREE)  return Integer;
procedure SM_BODY           (t: in out TREE;  v: in TREE);
                                       --       v: SUBP_BODY_DESC,
                                       --          PACK_BODY_DESC,
                                       --          BLOCK_STUB_VOID
function  SM_BODY           (t: in TREE)  return TREE ;
                                       -- returns SUBP_BODY_DESC,
                                       --         PACK_BODY_DESC,
                                       --         BLOCK_STUB_VOID
procedure SM_COMP_SPEC      (t: in out TREE;  v: in
TREE);
function  SM_COMP_SPEC      (t: in TREE)  return TREE ;
procedure SM_CONSTRAINT     (t: in out TREE;  v: in TREE);
                                          -- v: CONSTRAINT
function  SM_CONSTRAINT     (t: in TREE)  return TREE ;
                                          -- returns CONSTRAINT
procedure SM_CONTROLLED     (t: in out TREE;  v: Boolean);
function  SM_CONTROLLED     (t: in TREE)  return Boolean ;
procedure SM_DECL_S         (t: in out TREE;  v: in TREE);  -- v: DECL_S
function  SM_DECL_S         (t: in TREE)  return TREE ;  -- returns DECL_S
procedure SM_DEFN        (t: in out TREE;  v: in TREE);
                                                 --       v: DEF_OCCURRENCE
function  SM_DEFN        (t: in TREE)  return TREE ;
                                                 -- returns DEF_OCCURRENCE
procedure SM_DISCRIMINANTS  (t: in out TREE;  v: in TREE);    --       v: VAR_S
function  SM_DISCRIMINANTS  (t: in TREE)  return TREE ;    -- returns VAR_S
procedure SM_EXCEPTION_DEF  (t: in out TREE;  v: in TREE);
                                                 --       v: EXCEPTION_DEF
function  SM_EXCEPTION_DEF  (t: in TREE)  return TREE ;
                                                 -- returns EXCEPTION_DEF
procedure SM_EXP_TYPE       (t: in out TREE;  v: in TREE);    -- v: TYPE_SPEC
function  SM_EXP_TYPE       (t: in TREE)  return TREE ;  -- returns TYPE_SPEC
procedure SM_FIRST          (t: in out TREE;  v: in TREE);    -- v: DEF_OCCURR
function  SM_FIRST          (t: in TREE)  return TREE;   -- returns DEF_OCCURR
```

```
procedure SM_GENERIC_PARAM_S(t: in out TREE;  v: in TREE);
                                                    --          v: GENERIC_PARAM_S
function  SM_GENERIC_PARAM_S(t: in TREE)  return TREE;
                                                    -- returns GENERIC_PARAM_S
procedure SM_INIT_EXP        (t: in out TREE;  v: in TREE);     --    v: EXP_VOID
function  SM_INIT_EXP        (t: in TREE)  return TREE ;     -- returns EXP_VOID
procedure SM_LOCATION        (t: in out TREE;  v: in TREE);     --    v: LOCATION
function  SM_LOCATION        (t: in TREE)  return TREE ;     -- returns LOCATION
procedure SM_NORMALIZED_PARAM_S (t:TREE;   v: in TREE);     --       v: EXP_S
function  SM_NORMALIZED_PARAM_S (t: in TREE) return TREE ; -- returns EXP_S
procedure SM_OBJ_DEF         (t: in out TREE;  v: in TREE);    --  v: OBJECT_DEF
function  SM_OBJ_DEF         (t: in TREE)  return TREE ;  -- returns OBJECT_DEF
procedure SM_OBJ_TYPE        (t: in out TREE;  v: in TREE);     --  v: TYPE_SPEC
function  SM_OBJ_TYPE        (t: in TREE)  return TREE ;   -- returns TYPE_SPEC
procedure SM_OPERATOR        (t: in out TREE;  v: operator);
function  SM_OPERATOR        (t: in TREE)  return operator ;
procedure SM_PACKING             (t: in out TREE;  v: Boolean);
function  SM_PACKING             (t: in TREE)  return Boolean ;
procedure SM_POS           (t: in out TREE;  v: Integer);
function  SM_POS           (t: in TREE)  return Integer ;
procedure SM_REP           (t: in out TREE;  v: Integer);
function  SM_REP           (t: in TREE)  return Integer ;
procedure SM_SIZE          (t: in out TREE;  v: in TREE);     --       v: EXP_VOID
function  SM_SIZE          (t: in TREE)  return TREE ;     -- returns EXP_VOID
procedure SM_SPEC             (t: in out TREE;  v: in TREE);
                                                    -- v: HEADER
                                                    --    GENERIC_HEADER,
                                                    --    PACK_SPEC
function  SM_SPEC             (t:TREE)   return TREE ;  -- returns HEADER
                                                       -- GENERIC_HEADER,
                                                       -- PACK_SPEC
procedure SM_STM           (t: in out TREE;  v: in TREE);     --     v: STM, LOOP
function  SM_STM           (t: in TREE)  return TREE ;     -- returns STM, LOOP
procedure SM_STORAGE_SIZE     (t: in out TREE;  v: in TREE);     --    v: EXP_VOID
function  SM_STORAGE_SIZE     (t: in TREE)  return TREE ;     -- returns EXP_VOID
procedure SM_STUB             (t: in out TREE;  v: in TREE);
function  SM_STUB             (t: in TREE)  return TREE ;
procedure SM_TYPE_SPEC        (t: in out TREE;  v: in TREE);     --  v: TYPE_SPEC
function  SM_TYPE_SPEC        (t: in TREE)  return TREE ;  -- returns TYPE_SPEC
procedure SM_TYPE_STRUCT      (t: in out TREE;  v: in TREE);     -- v: TYPE_SPEC
function  SM_TYPE_STRUCT      (t: in TREE)  return TREE ;  -- returns TYPE_SPEC
procedure SM_VALUE                (t: in out TREE;  v: value);
function  SM_VALUE                (t: in TREE)  return value ;


-- Code Attribute.

procedure CD_IMPL_SIZE        (t: in out TREE;  v: Integer);
function  CD_IMPL_SIZE        (t: in TREE)  return Integer ;

private

    -- To be filled in...

end Diana;
```