ASIS for GNAT: From the Prototype to the Full Implementation

MOSCOW STATE UNIVERSITY, SCIENTIFIC RESEARCH COMPUTER CENTER:

- Sergey Rybin
- Alexey Kuchumov
- Vasily Fofanof

SWISS FEDERAL INSTITUTE OF TECHNOLOGY IN LAUSANNE, SOFTWARE ENGINEERING LAB:

- Alfred Strohmeier

11 June 1996

Content:

1 Overview of ASIS

1.1 What is ASIS (from ASIS FAQ):

The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment as defined by ISO/IEC 8652 (the Ada 95 Reference Manual) and any tool requiring information from this environment.

Examples of ASIS-based tools include:

- browsers
- coding style and standards compliance checkers
- cross-referencers
- data flow analysers
- documentation generators
- static correctness verifiers
- test-case generators
- quality and complexity metrics tools
- ...

1.2 Main ASIS Abstractions (private types)

Context

Logical handling of an Ada environment. Provides a view of a particular implementation of an Ada environment. Specifies a set of ASIS *Compilation Units* processed by an ASIS application

Compilation Unit

The black-box view of an Ada compilation unit. Corresponds to the definition of an Ada compilation unit in RM 95 and contains some implementation-dependent characteristics of a physical object acting as a compilation unit in the underlying Ada implementation. May be decomposed and analyzed as a white-box by means of ASIS *Elements*.

Element

Handling of syntax components (both explicit and implicit) of an ASIS *Compilation Units*.

1.3 How to use ASIS

- Basic ASIS application cycle

- Processing a Compilation Unit as a Black-Box

- Analysing a Compilation Unit as a White-Box
  - Strtuctural decomposition
  - Semantic queries

1.3.1 Basic ASIS application cycle

```
with Asis;
procedure My_ASIS_Application is
    My_Cont : Asis.Context;
begin
    Asis.Implementation.Initialize (...);
        -- To initialize the ASIS interface
    Asis.Ada_Environment.Associate ("Env1", My_Cont);
        -- To associate an ASIS Context with
        -- some Ada environment
    Asis.Ada_Environment.Open(My_Cont);
        -- To gain access to an Ada environmenmt
        ...
        -- Use the various Asis queries
        ...
    Asis.Ada_Environment.Close (My_Cont);
        -- To drop access to an Ada environment
    Asis.Ada_Environment.Dissociate (My_Cont);
        -- To drop the connection with an
        -- Ada environment
    Asis.Implementation.Finalize (...);
        -- To release all resources
end My_ASIS_Application;
```

1.3.2 Processing a Compilation Unit as a Black-Box

```
-- (a) How to get access to the unit(s)
-- within an ASIS Context:

--getting the content of the Context:

My_Unit_List : Asis.Compilation_Unit_List :=
   Asis.Compilation_Units.Compilation_Units
            (Context => My_Cont);


-- fetching a Compilation Unit
-- by its Ada name:

My_Unit, My_Unit_1 : Asis.Compilation_Unit;
         ...
My_Unit := Asis.Compilation_Units.
               Library_Unit_Declaration
                  (Name      => "Hello",
                   Context   => My_Cont);
```

```
-- getting the body
-- of an existing Compilation Unit

My_Unit_1 := Asis.Compilation_Units.
          Corresponding_Body
               (Library_Item => My_Unit);


-- getting the children
-- of a parent unit

Children : Asis.Compilation_Unit_List :=
   Asis.Compilation_Units.
          Corresponding_Children
               (Library_Unit => My_Unit);

--etc...
```

(b) Attributes of Compilation Units:

- **Unit_Kind**:
  - *A_Procedure*
  - *A_Function*
  - *A_Package*

  ...
- **Unit_Origin**:
  - *A_Predefined_Unit*
  - *An_Application_Unit*

  ...
- **Enclosing_Context**
- **Is_Equal**
- **Name** (*Ada name, like* “Hello”)
- **Text_Name** (*like “~user/Ada/hello.ads”*)
- **Can_Be_Main_Program**
- **Is_Body_Required**
- **Time_Of_Last_Update**

... etc

(c) Semantic relationships between Compilation Units:

(c.1) **Corresponding_Xxxx** queries; Xxxx may be:

Children
Parent_Declaration
Declaration
Body
Subunit_Parent_Body

(c.2) Querying the general semantic dependencies:

```
type Relation_Kinds is (
    Ancestors,
    Descendants,
        ...
    Supporters, -- compilation units upon which
                -- the given unit depends semantically,
                -- as defined by RM 95

    Dependents, -- reverse of Supporters
                ...
    Extended_Family); -- all units needed by a given
        -- unit to form a partition, as defined by RM 95

function Semantic_Dependence_Order
   (Compilation_Units : in Asis.Compilation_Unit_List;
    Dependent_Units   : in Asis.Compilation_Unit_List;
    Context           : in Asis.Context;
    Relation          : in Asis.Relation_Kinds)
return Asis.Relationship;
```

1.3.3 Analysing a Compilation Unit as a White-Box

(a) Structural decomposition:

```
Unit : Asis.Element := Asis.Elements.Unit_Declaration
    (Compilation_Unit => My_Unit);
begin
    if Declaration_Kinds (Unit) = A_Procedure_Body
    then
        declare
            Decl : Asis.Element_List := Asis.
                    Declarations.Body_Declarative_Items
                            (Declaration       => Unit,
                             Include_Pragmas => Fales);

            Stmts : Asis.Element_List := Asis.
                    Declarations.Body_Statements
                            (Declaration       => Unit,
                             Include_Pragmas => False);

            Hndlrs : Asis.Element_List :=
                    Asis.Declarations.
                        Body_Exception_Handlers
                            (Declaration       => Unit,
                             Include_Pragmas => False);
```

```
    begin
        -- processing declarations ...
        -- processing statements...
        -- processing exception handlers...
    end;

elsif Declaration_Kinds (Unit) =
                              A_Package_Declaration
then
    declare
        ...
    begin
        -- processing a package declaration
    end;
...
    ...
end if;
```

(b) Semantic queries:

```
Var         : Asis.Element;
Expr        : Asis.Element;
Var_Decl    : Asis.Element;
Expr_Type   : Asis.Element;

-- continuing the previous example: supposing that
-- Stmts is a list of statements:
   ...
for I in Stmts'Range loop
   if Statement_Kind (Stmts (I)) =
      An_Assignment_Statement
   then
      Var  := Assignment_Variable_Name (Stmts (I));
      Expr := Assignment_Expression (Stmts (I));

      Var_Decl :=
         Corresponding_Name_Declaration (Var);
      -- analysing the variable declaration...
      Expr_Type :=
         Corresponding_Expression_Type (Expr);
      -- analysing the declaration of expression type...
                  ...
   end if;
end loop;
```

2 ASIS 95 and Ada environment.

- RM 95 defines the notion of an environment as a *language abstraction*, needed to define the legality of a single Ada compilation Unit and the legality and the dynamic semantics of a completed partition consisting of legal compilation units;

- RM 95 leaves the way of creating and modifying the environment as *implementation defined*;

- RM 95 defines the notion of an environment for the situation, when a compiler *compiles a compilation unit* or when some "machine" *executes* a (completed) partition, and it says nothing about what happens with the environment in between;

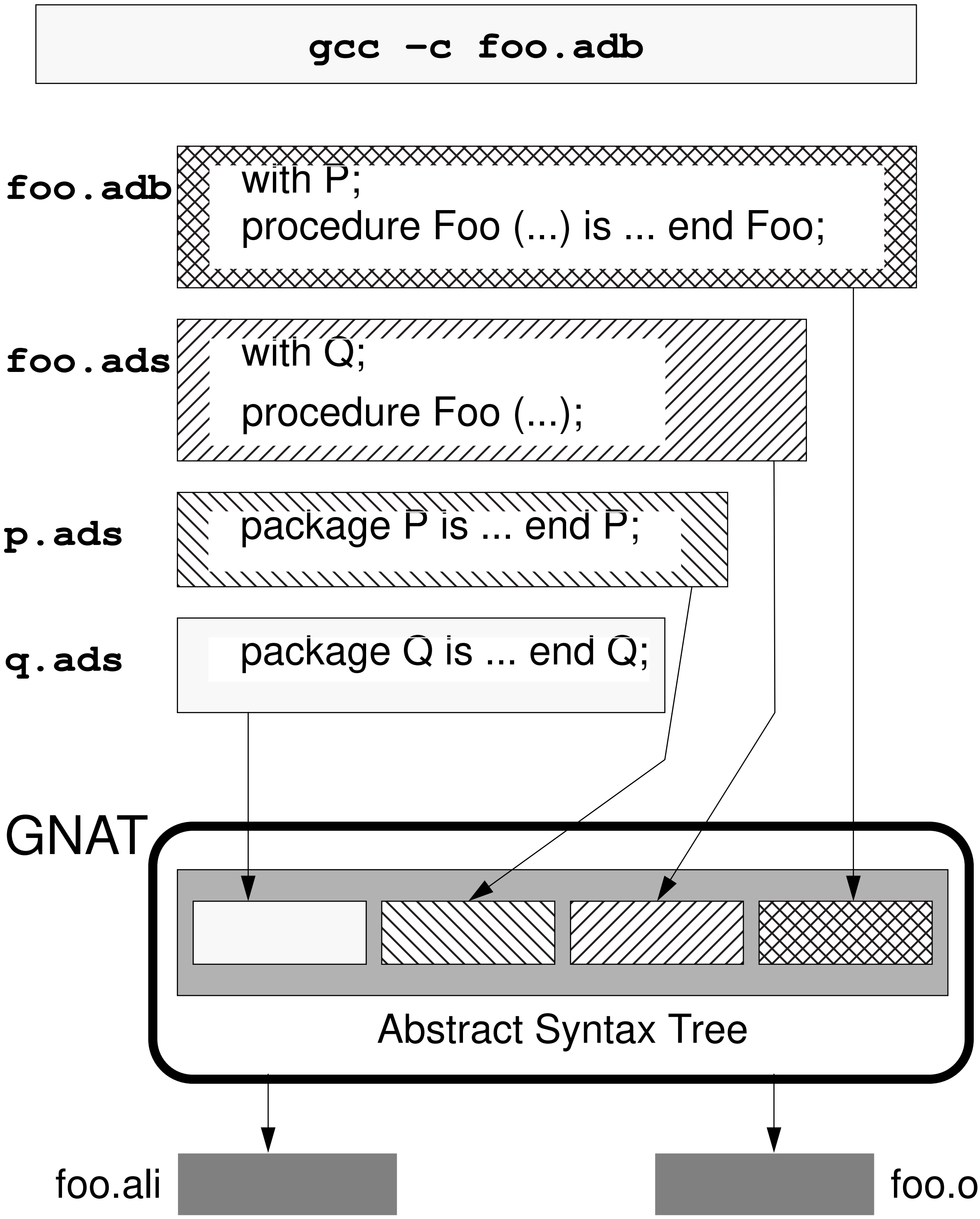
- There is *no* formal *relation* between (the process or the results of) *compilation* and any *changes in the environment*;

- "*Library*" is *no* more a formally-defined *technical term* in Ada.

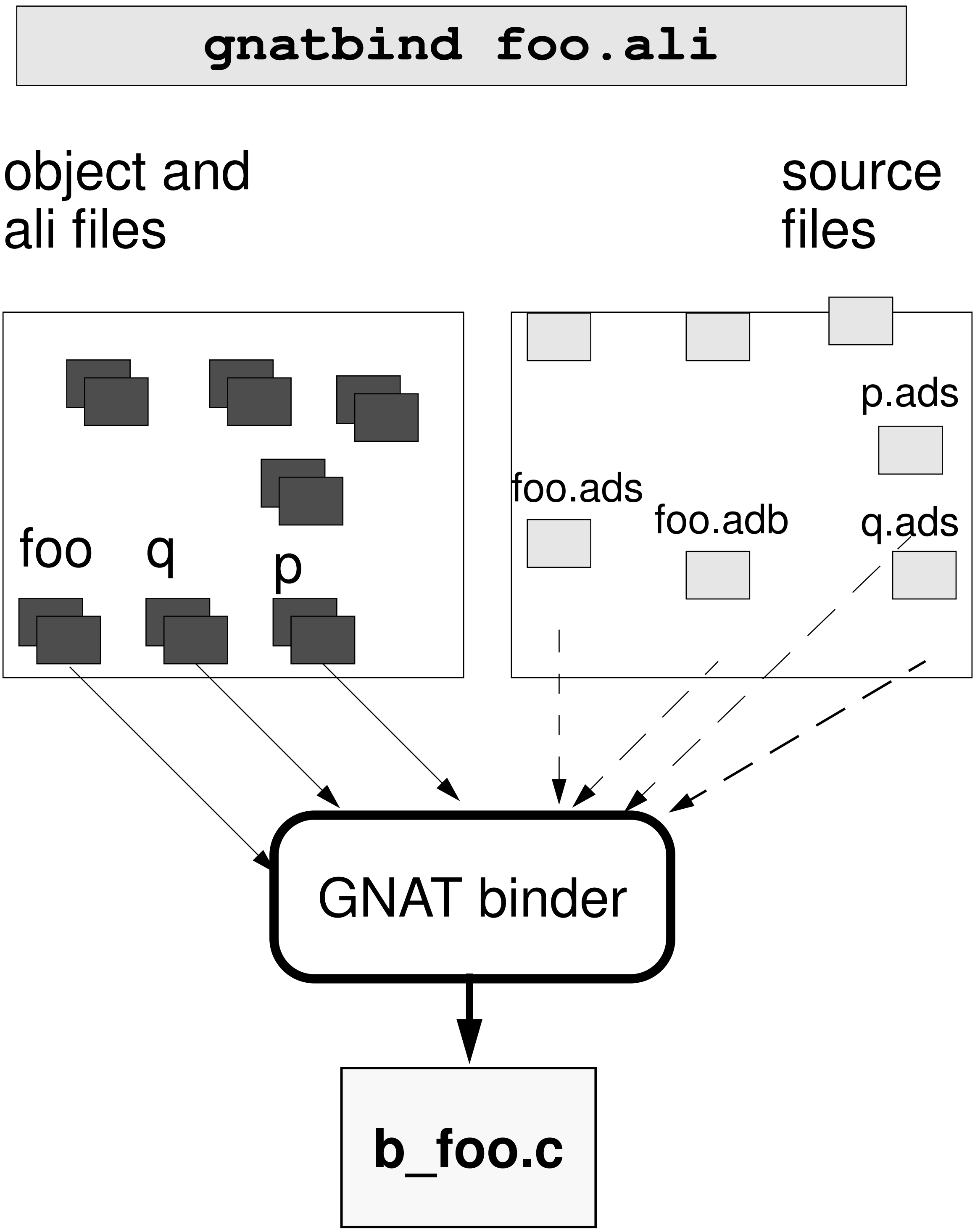
And ASIS 95 is an interface to an Ada environment, planned to be an information technology standard.

3 GNAT and Ada environment.

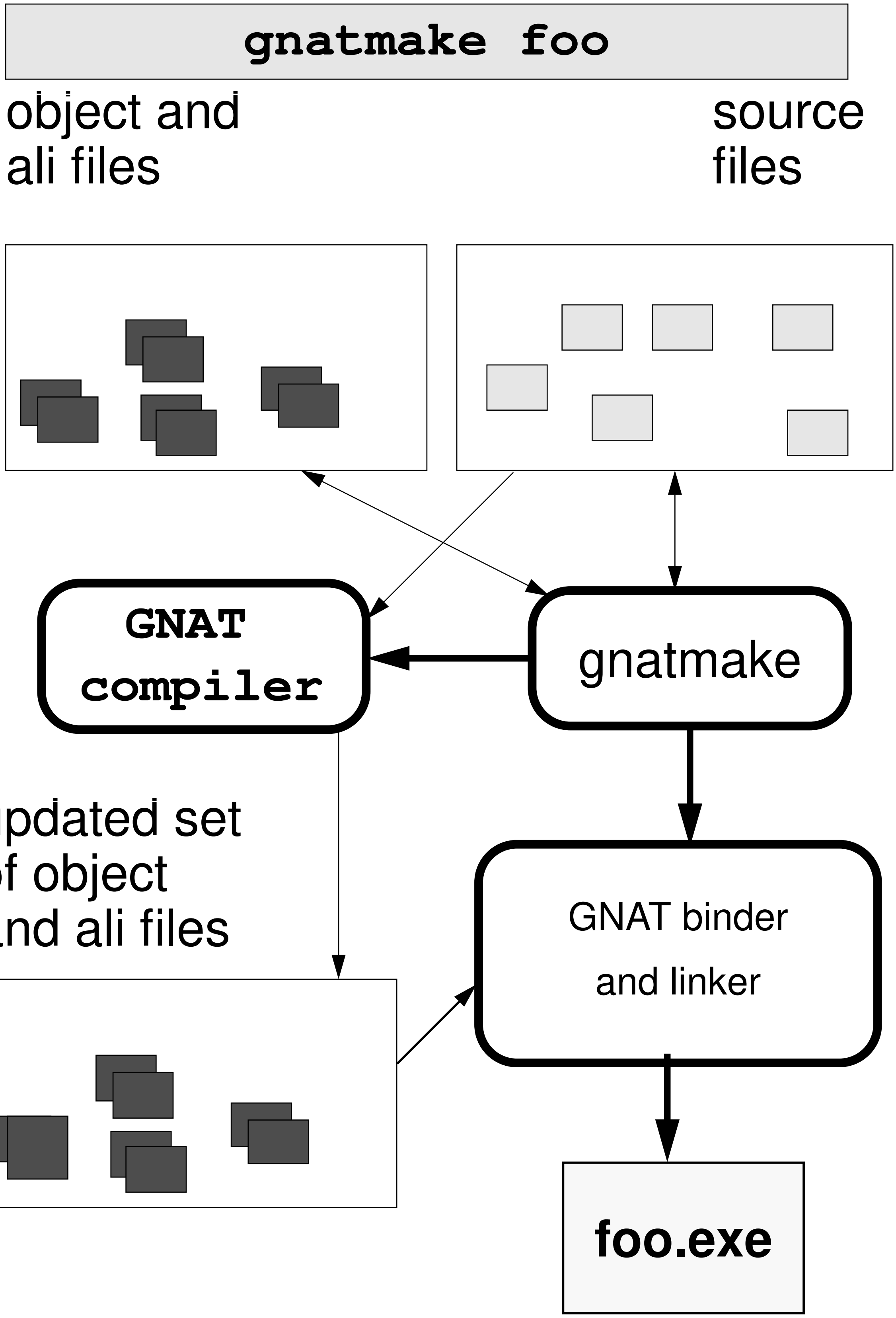
3.1 Compiler's viewpoint



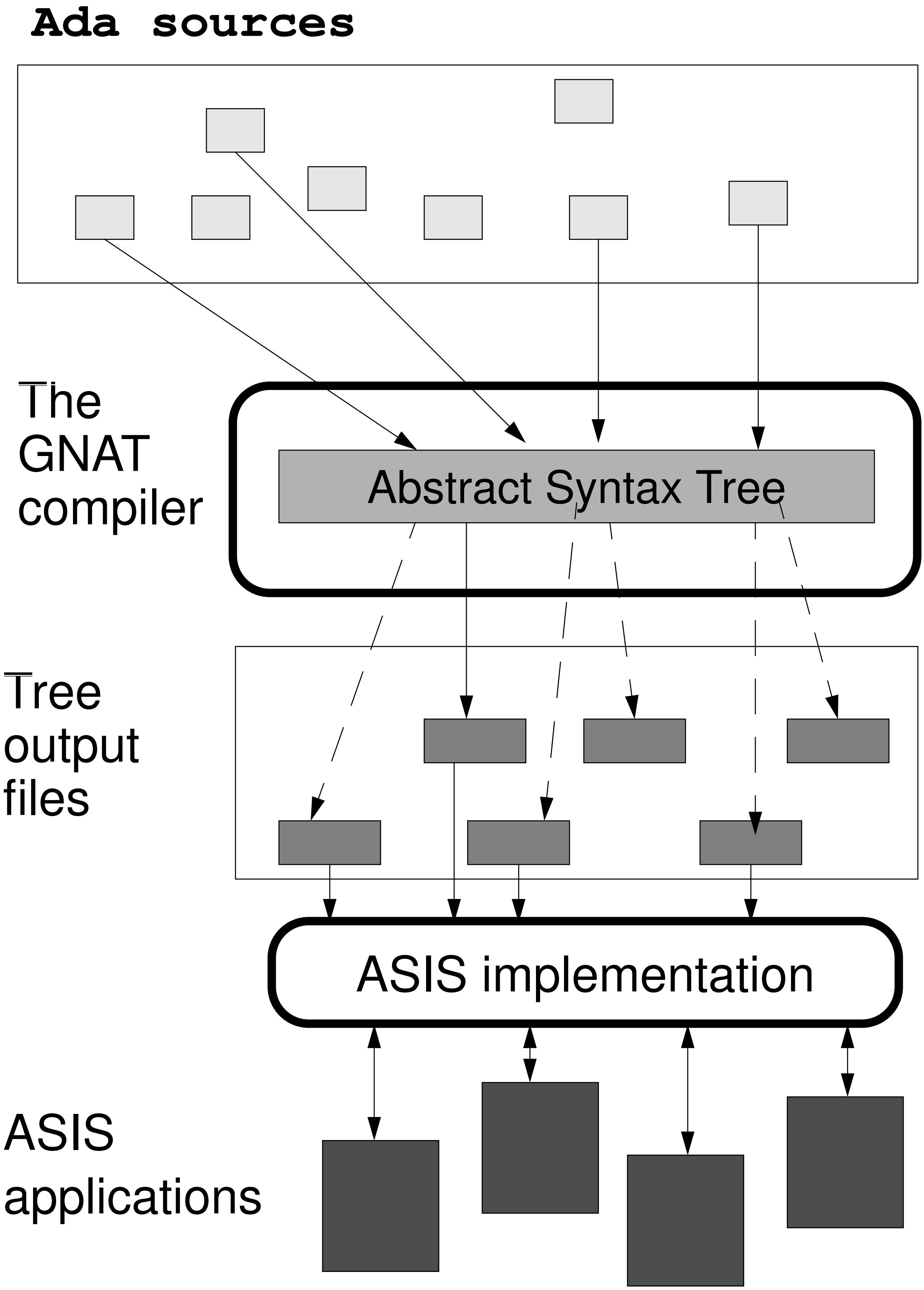
3.2 Binder's viewpoint



3.3 gnatmake's viewpoint

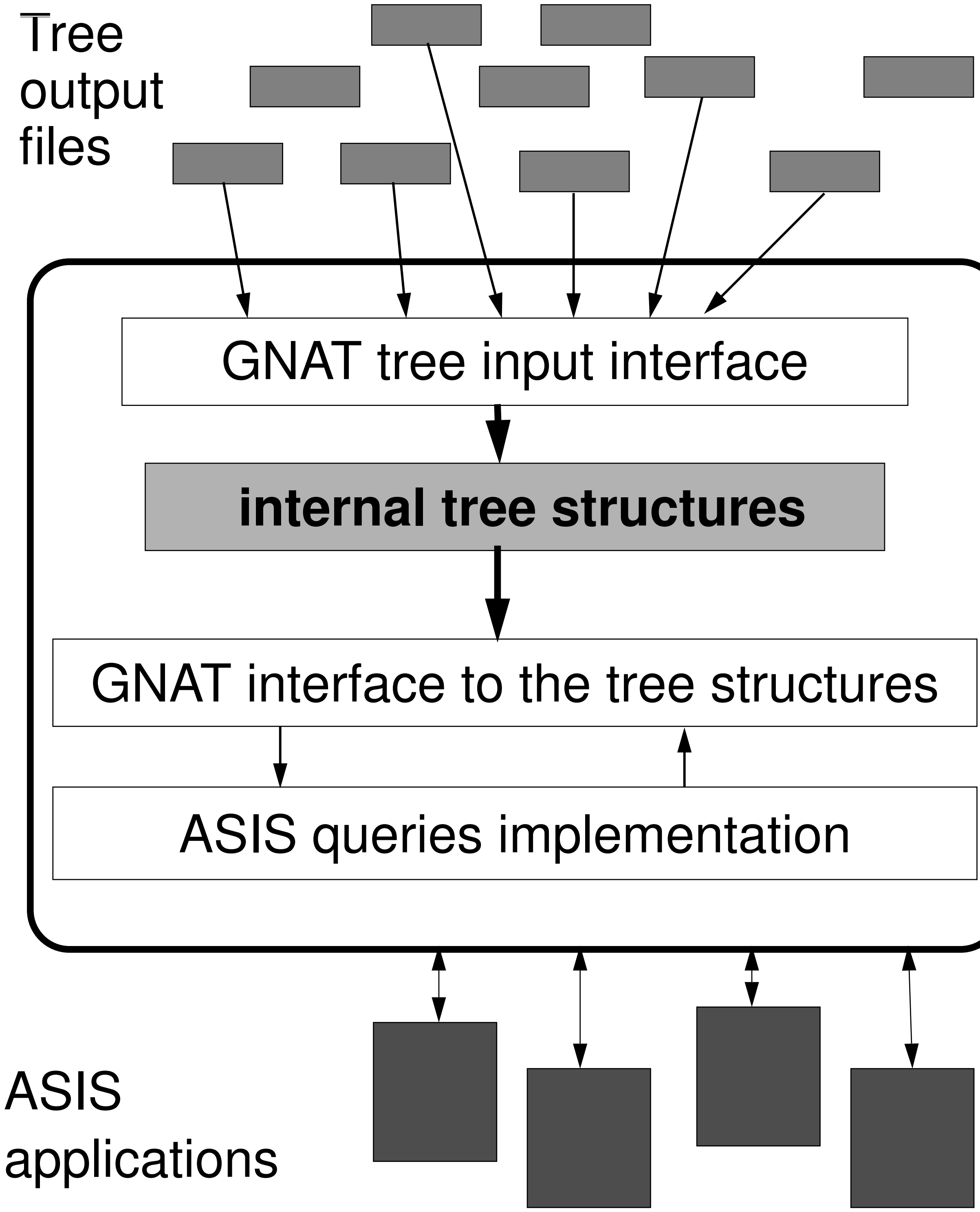


4 Interaction of ASIS with GNAT



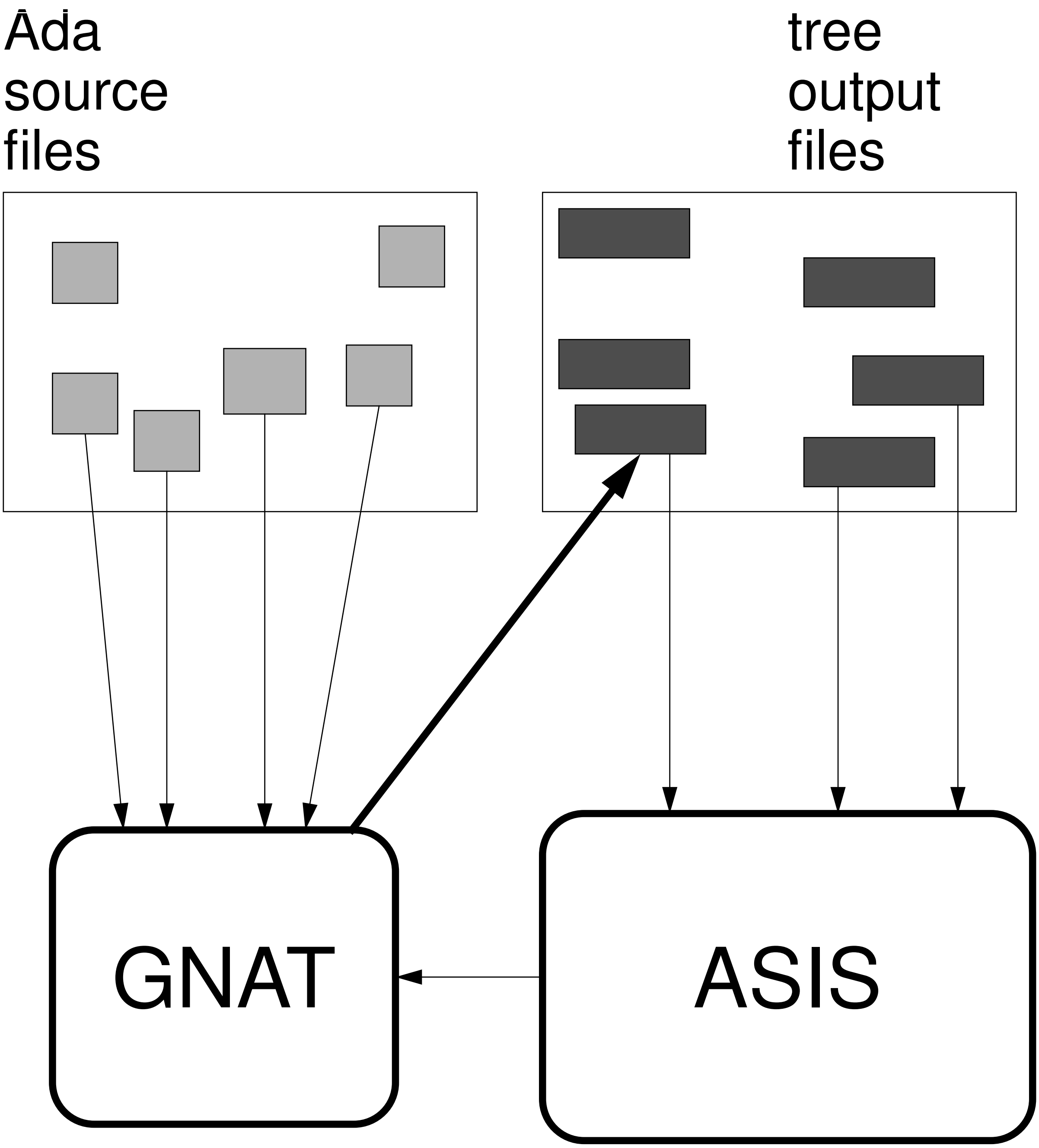
5 ASIS-for-GNAT implementation

5.1 Architecture



5.2 The current state

The view on an Ada environment



ASIS 95 functionalities

- Processing ***all the Ada sources in the current directory*** (only one ASIS Context can be opened);
- ***All structural queries*** of ASIS 2.0.E;
- ***Asis_Text*** (first prototype);
- ***Traverse_Element*** (implemented on top of ASIS structural queries),

ASIS applications

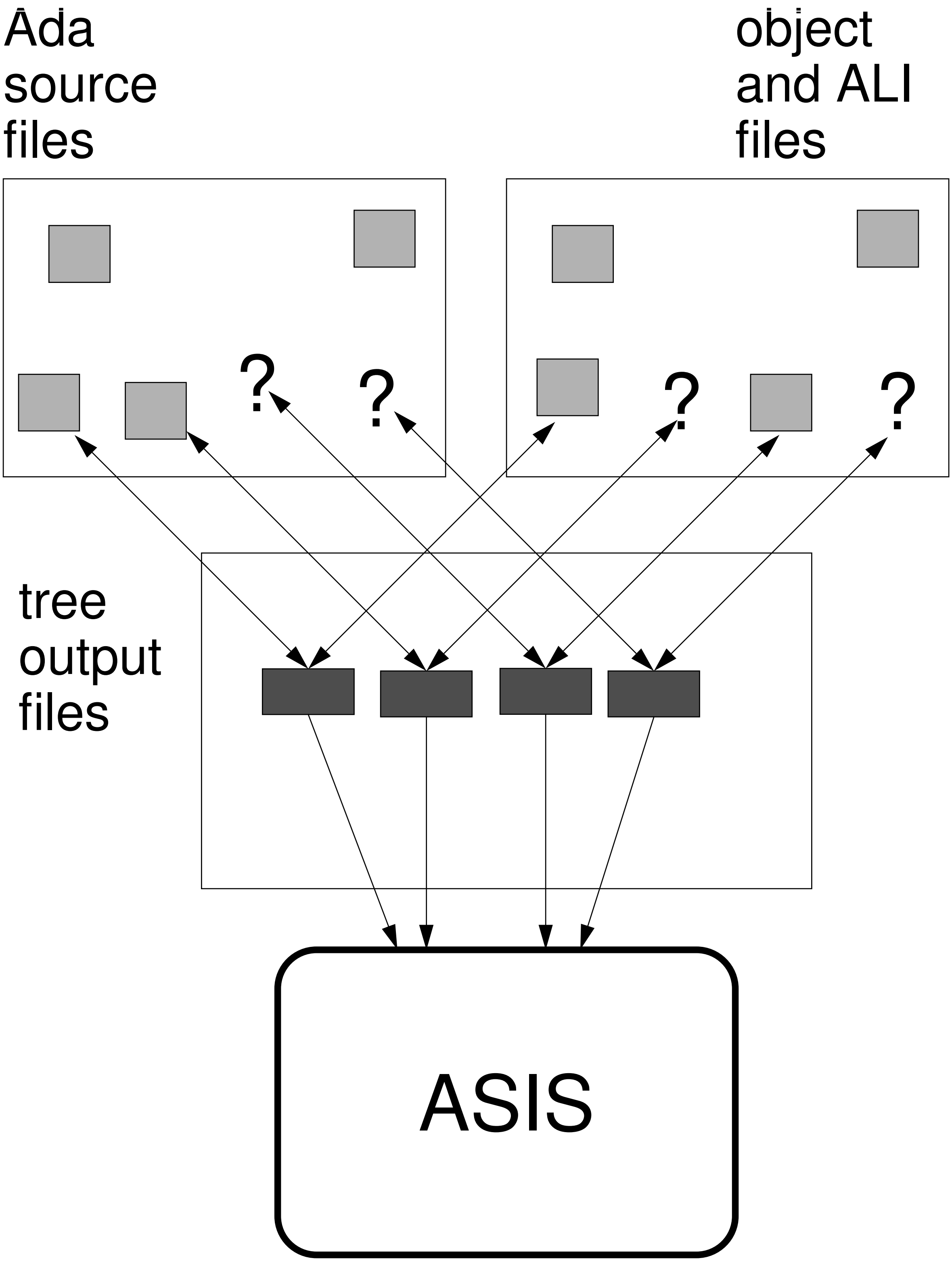
- ASIStint - interactive interpreter of ASIS queries, currently used as ASIS-for-GNAT debugging tool, supposed to be used also as the kernel of ASIS test harness;
- Display_Source - a sample pretty-printer, based on Traverse_Element;

Users

- Sema Group, Grenoble, France (Jean-Charles Marteau and Serge Reboul - implementing Display_Source);
- Uppsala University, Sweden;

5.3 Future plans

Other views of the Ada environment



ASIS 95 functionalities to be implemented by the end of 1996:

- Migrating to ASIS 2.0.F/2.0.D (ASIS as a hierarchy);

- Semantic dependencies queries for Compilation Units

- Semantic queries for explicit Elements

- Completing Asis_Text and Traverce_Element;

ASIS applications:

- Sample metric tools;
- ASIS secondary layer components;

User support

6 ASIS Panel at Ada-Europe'96

Wednesday Afternoon, 12 June 1996
14h00-15h30
continuing from 16h30-18h00

First Session => What is ASIS?

1. Overview of ASIS - Mr. Currie Colket
2. ASIS 95 - Mr. Jesper Jorgensen
3. ASIS for GNAT - Dr. Sergey Rybin
4. ASIS Resources on the Internet - Mr. Clyde Roby

Second Session => ASIS Usage Experience

1. Rational Usage Experience - Mr. Pascal Leroy
2. MITRE ASIS Usage Experience - Mr. Bill Thomas
3. Sema Group Grenoble ASIS Usage Experience - Mr. Bertrand Petitprez
4. Tool Vendor ASIS Usage Experience (GRC) - Mr. Tom Strelich

5. Panel Discussion Questions (30 minutes)

RM 95, 10.1.4(1) says:

Each compilation unit submitted to the compiler is compiled in the context of an *environment* declarative_part (or simply, an *environment*), which is a conceptual declarative_part that forms the outermost declarative region of the context of any compilation. At run time, an environment forms the declarative_part of the body of the environment task of a partition (see 10.2, “Program Execution”).

RM 95 10.2(10-13) says:

The environment task for a partition has the following structure:

```
task Environment_Task;
task body Environment_Task is
    ... (1) -- The environment declarative_part
              -- (that is, the sequence of library_items) goes here.
begin
    ... (2) -- Call the main subprogram, if there is one.
end Environment_Task;
```

The environment declarative_part at (1) is a sequence of declarative_items consisting of copies of the library_items included in the partition. The order of elaboration of library_items is the order in which they appear in the environment declarative_part

RM 95 10.1.4 (5) says:

When a compilation unit is compiled, all compilation units upon which it depends semantically shall already exist in the environment; the set of these compilation units shall be *consistent* in the sense that the new compilation unit shall not semantically depend (directly or indirectly) on two different versions of the same compilation unit, nor on an earlier version of itself.

RM 95 10.2 (19) says:

The full expanded names of the library units and subunits included in a given partition shall be distinct

RM 95 10.2 (27) says:

The implementation shall ensure that all compilation units included in a partition are consistent with one another, and are legal according to the rules of the language.

RM 95 10.1.4 (3) says:

The mechanisms for creating an environment and for adding and replacing compilation units within an environment are implementation defined.