ASIS for GNAT

12 June 1996

ASIS Panel at Ada-Europe'96

- history and current state;
- implementation approach and the main implementation problems;
- the current distribution and the first user experience;
- debugging and testing.

1 Short history

Research and prototyping (09/94-12/95):

- September 1994 - the start of the project;
- January 1995 - the first line of the code was written;
- Spring 1995 - first executable for ASIS-like internal GNAT interface;
- End of spring 1995 - the first prototype of ASIS test harness was implemented;
- October 1995 (Ada-Europe'95) - Most of the ASIS 95 structural queries implemented as ASIS-like internal GNAT interface under the limitation “only one compilation unit at a time”;
- November - December 1995 -migrating
    - from internal GNAT interface to “proper” ASIS interface;
    - from PC/MS-DOS to sparc/ SunOS/Solaris;

Human resources - around 20 man-months

First releases (01/96-05/96):

- January 1996 - the first publicly available ASIS-for-GNAT version;
- January - March 1996 - processing all the source files in the current directory,
- April 1996:
  - the first release of Asis_Text;
  - ASISTint;
  - Traverse_Element;
  - the first experience of user support (massive debugging);
  - the first “external” applications (Display_Source);

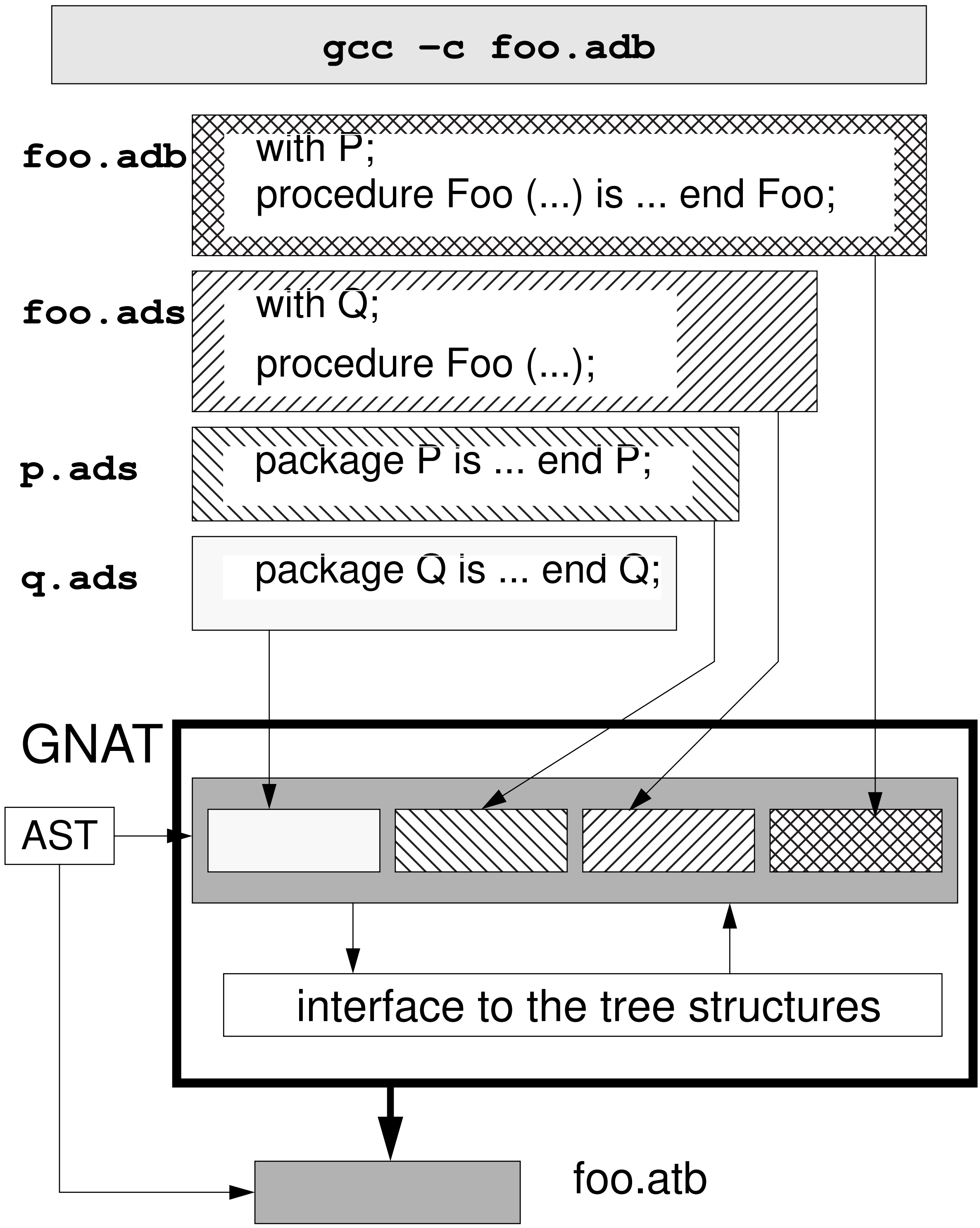
- May 1996 - the latest distribution;

Human resources - around 9 man-months.

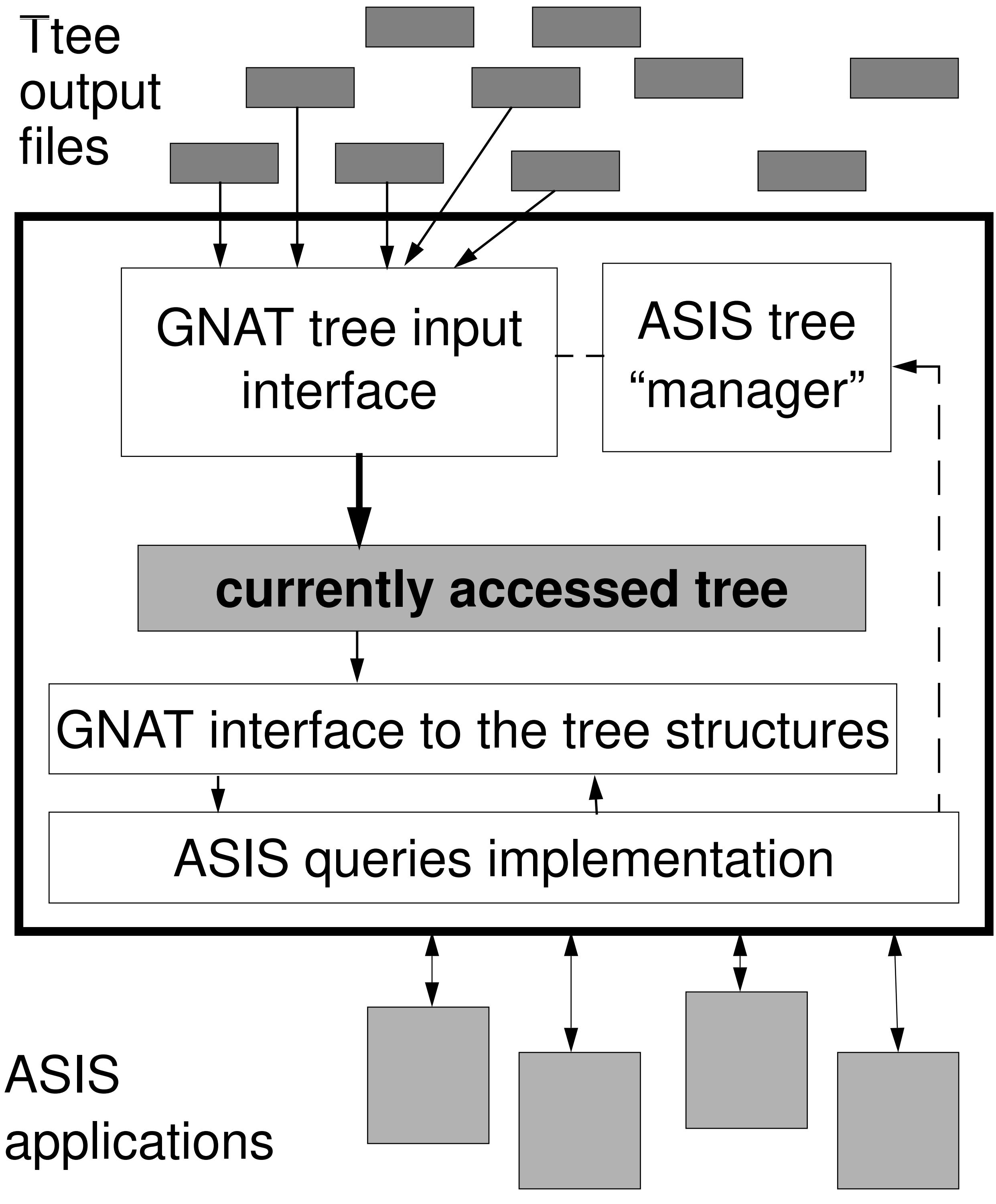
June 1996 - towards the industrial ASIS-for-GNAT release?

2 Implementation approach.

2.1 GNAT overview:



2.2 ASIS implementation architecture:



2.3 Main implementation problems

1. **What to implement?**
   - How to define an ASIS Context for GNAT? (GNAT = the GNAT **compiler** + the GNAT **binder** + **gnatmake** + ?)

2. **How to implement?**
   - mapping between ASIS notions and GNAT tree structure;
   - Lack of information in the GNAT tree (compiler-time optimizations);
   - Tree transformations

     (A'Range -> A'First .. A'Last);

   - Tree swapping;

3. **What about performance?**
   - To compile on the fly or to reuse the existing tree output files?
   - keeping several trees in memory;
   - ASIS "database" for black-box Compilation Unit attributes

3 ASIS-for-GNAT distribution and installation

3.1 lglftp.epfl.ch/pub/ASIS

- Asis-for-GNAT sources
- Documentation

3.2 Installation

What do you need?

- GNAT installed on your site;
- ASIS-for-GNAT sources;
- GNAT sources (from the GNAT source distribution)
- gcc sources;

What should you do?

- To create the GNAT system-dependent Ada source components;
- To compile GNAT system-dependent C components;
- To compile ASIS components and their GNAT supporters (long);

3.3 Some figures

Code size - 2.4 MB;

Number of files - 91;

Number of GNAT components used as supporters - around 250;

Executable size - around 4.5 MB

3.4 Portability considerations

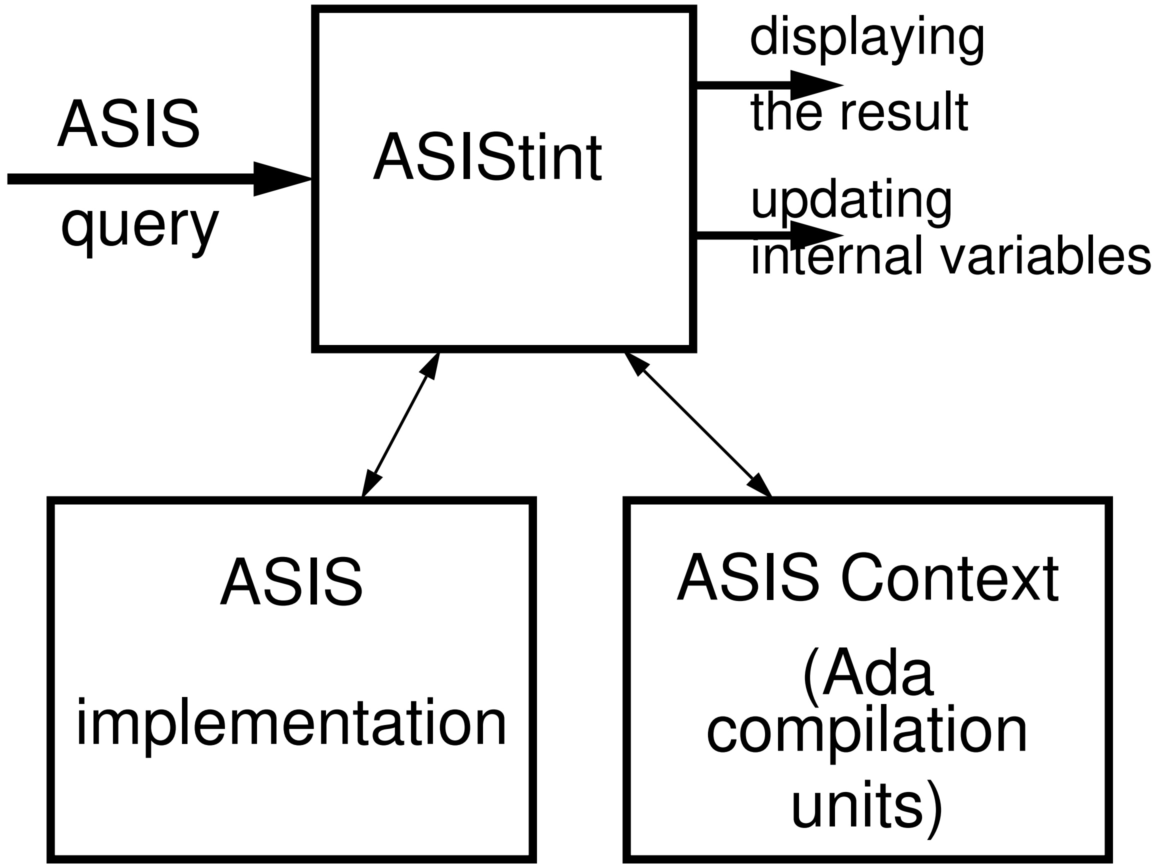
ASIS-for-GNAT is (should be) as portable, as GNAT itself is.

4 Debugging and testing

4.1 ASIStint

ASIStint = ASIS Test INTerpreter

4.1.1 "Necessity is the mother of invention"



4.1.2 An example

```
package bug_40 is
    generic
        type Item is private ;
        type Link is access Item ;
    package buggy is
        tutu : Item ;
    end buggy ;
end bug_40 ;
```

```
set ( lib )
Initialize ( "-asisd" )
Associate ( lib, "", "" )
Open ( lib )
set ( Unit, Library_Unit_Declaration ( "bug_40", lib ))
set ( Main, Unit_Declaration ( Unit ) )
set ( List, Visible_Part_Declarative_Items ( Main, true ))
set ( gen_pack_decl,     List (1) )
set ( gen_form_pars,
        Generic_Formal_Part (gen_pack_decl, false))
set ( gen_acc_type_decl, gen_form_pars (2) )
print ( gen_acc_type_decl )
set (gen_acc_type_def, Type_Declaration_View (
        gen_acc_type_decl) )
print ( gen_acc_type_def )
```

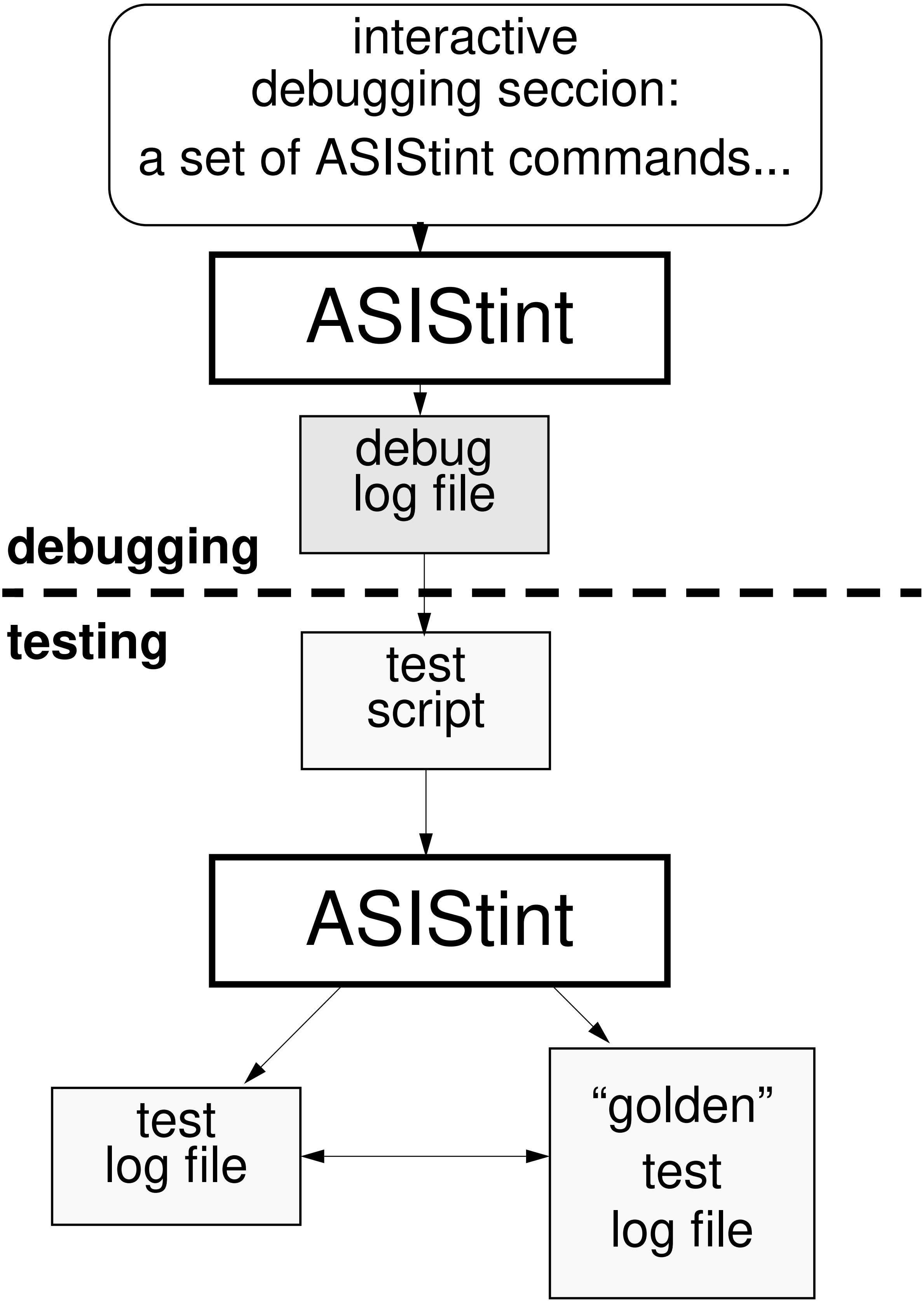
Concole log:

```
>...
> set ( gen_acc_type_decl, gen_form_pars (2) )
> print ( gen_acc_type_decl )

Node    :   1343: N_FORMAL_TYPE_DECLARATION
R_Node :    1343: N_FORMAL_TYPE_DECLARATION
   Elem Kind             : A_DECLARATION
   Elem Internal Kind : A_FORMAL_TYPE_DECLARATION
   Special Case         : NOT_A_SPECIAL_CASE
   Is_Part_Of_Implicit    = FALSE
   Is_Part_Of_Inherited = FALSE
   Is_Part_Of_Instance  = FALSE
==================
> set (gen_acc_type_def, Type_Declaration_View (
        gen_acc_type_decl) )
 Ok.
> print ( gen_acc_type_def )

Node    :   1349: N_ACCESS_TO_OBJECT_DEFINITION
R_Node :  1349: N_ACCESS_TO_OBJECT_DEFINITION
   Elem Kind             : A_DEFINITION
   Elem Internal Kind :
A_FORMAL_POOL_SPECIFIC_ACCESS_TO_VARIABLE
   Special Case         :      NOT_A_SPECIAL_CASE
   Is_Part_Of_Implicit     = FALSE
   Is_Part_Of_Inherited  = FALSE
   Is_Part_Of_Instance  = FALSE
==================
```

4.1.3 ASIStint commands

1. HELP
2. QUIT
3. LOG
4. OUTPUTLEVEL
5. RUN
6. PAUSE
7. SET
8. PRINT

4.1.4 From debugging to testing



4.1.5 ASIStint as an ASIS test harness

Program testing - a test is test data and expected results.

Interface testing (ASIS):

"Traditional" approach - a test is:

- test data (the content of an ASIS Context;
- test application (ASIS application);
- expected results;

ASIStint-based approach for ASIS - a test is:

- test ASIS Context as test data;
- a sequence of ASIS queries instead of a test application;
- expected results ("golden" log files);

Advantages of ASIStint-based approach

- ASIStint scripts are smaller and simpler, then test ASIS applications;
- the only ASIS application, which has been compiled and executed during the testing process is ASIStint itself;
- tests can be developed incrementally;
- debug scripts can be (re)used as (regression) tests;

"Side-effects" of ASIStint-based approach

- Support for both debugging and testing processes;
- ASIStint as ASIS training tool

4.2 “Massive” testing of structural queries

(Jean-Charles Marteau and Serge Reboul)

- Implementing Traverse_Element on top of ASIS structural queries;
- Implementing Traverse_Element-based sample pretty-printer (Display_Source);
- Applying Display_Source to ACVC test suite;
- Applying Display_Source to huge Ada components