AdaCore
The GNAT Pro Company

GNAT: On the Road to Ada 2005

Edmond Schonberg and Javier Miranda

SIGAda 2004
Atlanta, Georgia

Ada 2005: The language revision process

- **The ARG has been at work for 10 years.**
- **Ada issues classified as:**
  - Confirmation (the ARM is correct and clear)
  - Ramifications (the ARM is correct but obscure)
  - No action
  - Binding interpretations (the ARM was wrong)
  - Amendments
    - Corrigendum 2000 (WG9 approved, published, implemented)
    - **Corrigendum 200Y** (WG9 approved, will be in new ARM)
    - Working items (still under discussion)

Total of 384 Ada Issues (AIs):

22 high priority issues, 47 medium priority issues

GNAT and Ada 2005 High Priority Issues

- 85: Append_File, Reset, and positioning for Stream_IO
- 147: Optimization of controlled types
- 195: Streams 'Input and initialization
- 204: Language interfacing support is optional
- 214: Distinct names for compilation units
- 217: Limited with clauses
- 220: Subprograms within private compilation units
- 235: Resolving ‘Access
- 239: Controlling inherited default expressions
- 243: Is a subunit of a subunit of L also a subunit of L?
- 249: Ravenscar profile for High-Integrity systems

Issues in Blue = supported by GNAT

GNAT and Ada 2005 High Priority Issues

- **251: Abstract interfaces to provide multiple inheritance**
- **252: Object.Operation notation**
- **254: Anonymous access to subprogram types**
- 265: Partition elaboration policy for high-integrity systems
- 266: Task termination procedure
- 270: Stream item size control
- 280: Allocation, deallocation and use of objects after finalization
- 297: Timing events
- **305: New pragma and additional restriction identifiers for RT-Systems**
- 310: Execution-time clocks
- 353: New restrictions identifier: *No_Synchronous_Control*
- 363: Eliminating access subtype problems

Issues in Blue = implemented in GNAT

GNAT and Ada 2005 Medium Priority Issues

- 161: Default-initialized objects
- 185: **Branch cuts of inverse trigonometric and hyperbolic functions**
- 209: pragma Reviewable; can objects become uninitialized
- **216: Unchecked unions: variant records with no run-time discriminant**
- 218: Accidental overloading when overriding
- **221: Default bit-order is static**
- 224: *Pragma unsupress*
- 225: Aliased current instance for limited types
- 227: Behavior of *Ada.Streams.Read* when at the end of stream
- 229: Accessibility rules and generics
- **230: Generalized use of anonymous access types**

Issues in Blue = implemented in GNAT

GNAT and Ada 2005 Medium Priority Issues

- **231: Access to constant parameters and null-excluding access subtypes**
- 233: Inheritance of components of generic formal derived types
- 241: Testing for *Null_Occurrence*
- 242: Surprising behavior of *Update*
- 246: View conversions between arrays of a by-reference type
- 247: Alignment of composite types
- 248: Directory operations
- 258: Behavior of *Interfaces.C.To_C* when the result is null
- 259: Can accesses to volatile objects be combined?
- **262: Access to private units in the private part**
- 263: Scalar formal derived types are never static

Issues in Blue = (known to be) supported by GNAT

GNAT and Ada 2005 Medium Priority Issues

- 267: Fast float-to-integer conversions
- 268: Rounding of real static expressions
- 272: Pragma atomic and slices
- 280: **Assert pragma**
- **287: Limited aggregates allowed**
- 296: Vector and matrix operations
- 298: Non-preemptive dispatching
- 301: Operations on language-defined string types
- 316: Return accessibility checks and value conversions
- 317: Partial parameter lists for formal packages
- 318: Returning limited objects without copying

Issues in Blue = supported by GNAT

GNAT and Ada 2005 Medium Priority Issues

- 321: Definition of dispatching policies
- 326: Incomplete types
- 327: Dynamic ceiling priorities
- 329: **Pragma *No_Return***
- 340: *Mod* attribute
- 344: Allow nested type extensions
- 345: Protected and task interfaces
- 348: Null procedures
- 351: Time operations
- 360: Types that need finalization

GNAT and Ada 2005 Medium Priority Issues

- 361: Raise with message
- 362: Some predefined packages should be recategorized
- 364: Fixed-point multiply/divide
- 376: ***Interfaces.C* works for C++ as well**
- 381: New restrictions identifier: *No_Dependence*

Issues in Blue = supported by GNAT

AdaCore
The GNAT Pro Company

Brief Overview of Implemented Ada 2005 Issues

200Y Amendments

200Y Amendments

- AI-217: Limited-with clause
- AI-262: Private with clause
- AI-217: Limited-with clause
- AI-262: Private with clause

AI-217: Limited With Clause

- **Ada 95**

```
package Mutually_Recursive_Types  Is
    type T1;
    type T2;

    type Acc_T1 is access T1;
    type Acc_T2 is access T2;

    type T1 is record
         Ref : Acc_T2;
         . . .
    end record;

    type T2 is record
          Ref : Acc_T1;
          . . .
    end record;
end Mutually_Recursive_Types ;
```

Problem: Software Structure

- **Ada 2005**

```
limited with Q;
package P is
    type Acc_T2 is access Q.T2;
    type T1 is record
         Ref : Acc_T2;
         . . .
    end record;
end P;
```

```
limited with P;
package Q is
    type Acc_T1 is access P.T1;
    type T2 is record
         Ref : Acc_T1;
         . . .
    end record;
end Q;
```

The limited view provides <u>incomplete visibility</u> of:

- **Type declarations**
- **Nested packages**

Does not create a semantic dependence !

(and hence no elaboration dependence)

AI-262: Private with clauses

- Ada 95

```
package Lib is
   . . .
private
   type Internal_Type is …
end Lib;
```

```
package Lib.P is

private
   -- Use Internal_Type
   . . .
end Lib.P;
```

```
private package Lib.Q is
   -- Internal_Type should
   --  be declared here


end Lib.Q;
```

- Ada 2005

```
package Lib is
   . . .
end Lib;
```

```
private with Lib.Q;
package Lib.P is

private
   -- Use Internal_Type
   . . .
end Lib.P;
```

```
private package Lib.Q is
   type Internal_Type is …


end Lib.Q;
```

- Ada 95

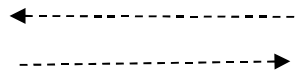
Entities in private-withed units can be used in the private part

AI-217 plus AI-262

```
package Parent is
   . . .
end Parent;
```

```
limited private with Parent.Q;
package Parent.P is

private
. . .Parent.Q.QT
end Parent.P;
```



```
limited private with Parent.P;
package Parent.Q is

private
   type QT is ...
end Parent.Q;
```

Ada 2005: Access type issues

- AI-230: Generalize anonymous access types
- AI-231: Access to constant parameters and null-excluding access subtypes
- AI-254: Anonymous access to subprogram types

AI-230: Generalize Anonymous Access Types

```
type Root  is tagged record . . .
type D1      is new Root with . . .
type D2      is new Root with . . .
```

- **Ada 95:**

```
type Root_Ref  is access all Root'Class;
Table :  array (1 .. 2) of  Root_Ref
                := (Root_Ref  (new D1),
                    Root_Ref  (new D2));

type My_Rec is record
    Data : Root_Ref := Root_Ref (new D1);
end record;
```

- **Ada 2005:**

```
Table :  array (1 .. 2) of access Root'Class
                := (new D1, new D2);

type My_Rec is record
    Data : access Root'Class := new D1;
end record;

Farm_1   : access Root'Class renames Table (1);
Rec      : My_Rec;
My_Best : access Root'Class renames Rec.Component;
```

AI-231: Null-excluding access subtypes and access to constant parameters

- **Ada 95**

```
function Lowercase
   (Name : access String)
   return String;
```

- **The anonymous access CAN NEVER be null**
- **Anonymous access to constants is not provided**

- **Ada 2005**

```
function Lowercase
   (Name : not null access
constant String)
   return String;
```

- **Null-exclusion under control of the programmer**
- **Anonymous access to constants allowed**

AI-254: Anonymous access to subprogram types

- **Ada 2005:**

```
function Integrate (Fn      : access function (X: Float) return Float;
                    From    : Float;
                    To      : Float) return Float is
begin
      -- Fn (X)  callable
      . . .
end Integrate;

--  Use of a local function
Result := Integrate (My_Double'Access, From => 3.0, To => 9.0);

--  Use of a library function
Result := Integrate (Ada.Numerics.Elementary_Functions.Sqrt'Access, 3.0, 9.0);
```

All together (230, 231, 254)

- **Ada 2005**

```
type Farm_1 is array (1 .. 2) of not null access Root'Class := . . .
type Farm_2 is array (1 .. 2) of access constant  Root'Class := . . .
type Farm_3 is array (1 .. 2) of not null access constant Root'Class := . . .
type Funcs   is array (1 .. 2) of not null access function (X : Float) return Float;

-- Available also for array components, discriminants, and record
components (AI-230)

type My_Rec is record
   Pet _1      : not null access Root'Class :=  . . .
   Pet _2      : access constant Root'Class :=  . . .
   Pet _3      : not null access constant Root'Class :=  . . .
   Evaluate  : not null access function (X : Float) return Float;
end record;
```

Ada 2005: Aggregates

- AI-287: Aggregates for limited types

AI-287: Aggregates for Limited Types

AI-287: Aggregates for Limited Types

- **Example (Ada 95): Some package version 1**

```
package ADT is
    type Data is limited private;
    type T_Data_Ptr is access Data;

    function New_Data (Value : … )
        return T_Data_Ptr;
private
    type Data is  record
        Info   : … ;
    end record;
end ADT;
```

```
package body ADT is

    function New_Data (Value : … )
        return T_Data_Ptr is
    begin
        return new Data'(Info => Value);
    end New_Data;

end ADT;
```

AI-287: Aggregates for Limited Types

- **Example (Ada 95): Some package version 2**

```
package ADT is
   type Data is limited private;
   type T_Data_Ptr is access Data;

   function New_Data (Value : … )
      return T_Data_Ptr;
private
   type Data is limited record
      Info   : … ;
      Lock : … ; -- Protected object
   end Data;
end ADT;
```

```
package body ADT is
   function New_Data (Value : … )
      return T_Data_Ptr
   is
      Aux : T_Data_Ptr := new T_Data;
      -- Lock is silently default-initialized
   begin
      Aux.Info := Value;
      return Aux;
   end New_Data;
end ADT;
```

Why is this dangerous?

AI-287: Aggregates for Limited Types

- **Example (Ada 95): Some package version 3**

```
package ADT is
   type Data is limited private;
   type T_Data_Ptr is access Data;

   function New_Data (Value : … )
      return T_Data_Ptr;
private
   type Data is limited record
      Info   : … ;
      Lock : … ; -- Protected object
      More_Info : … ;
   end Data;
end ADT;
```

```
package body ADT is
   function New_Data (Value : … )
      return T_Data_Ptr
   is
      Aux : T_Data_Ptr := new T_Data;
      -- Lock is silently default-initialized
      -- More_Info is orphaned
   begin
      Aux.Info := Value;
      return Aux;
   end New_Data;
end ADT;
```

Because we can forget to initialize additional components

AI-287: Aggregates for Limited Types

- **Example (Ada 2005): New package version**

```
package ADT is
   type Data is limited private;
   type T_Data_Ptr is access Data;

   function New_Data (Value : … )
      return T_Data_Ptr;
private
   type Data is limited record
      Info   : … ;
      Lock : … ; -- Protected object
      More_Info : … ;
   end Data;
end ADT;
```

```
package body ADT is

   function New_Data (Value : … )
      return T_Data_Ptr is
   begin
      return new Data'
        (Info     => Value,
         Lock     => <>,
         others => <>);
   end New_Data;
end ADT;
```

Default initialization can be specified by the programmer

200Y Amendments

- AI-249: Ravenscar profile for high-integrity systems
- AI-305: New pragma and additional restriction identifiers for real-time systems

AI-305: New Pragma and Additional Restriction Identifiers for RT Systems

- New pragma:
  - pragma Detect_Blocking
- New static restriction identifiers:
  - No_Calendar
  - **No_Dynamic_Attachment**
  - No_Local_Protected_Objects
  - No_Protected_Type_Allocators
  - No_Relative_Delay
  - **No_Requeue_Statements**
  - No_Select_Statements
  - **No_Task_Attributes_Package**
  - **Simple_Barriers**
- New dynamic restriction_identifier:
  - No_Task_Termination
- New parameter identifier for dynamic restrictions:
  - Max_Entry_Queue_Length

200Y Amendments

- AI-252: Object.Operation notation
- AI-251: Abstract Interfaces
- AI-252: Object.Operation notation
- AI-251: Abstract Interfaces

AI-252: Object.Operation notation

```
package P is
   type T is tagged record
      Component : Integer :=  . . . ;
   end record;
   function F      (X : T)         return Integer;
   function Self   (X : T'Class) return T'Class;
end P;
```

- **Ada 95**

```
with P; use P;
procedure Test_Ada95 is
   type Ptr_Obj is access all P.T'Class;
   Obj    : P.T;
   Ptr    : Ptr_Obj := new P.T;

   O_1    : P.TP'Class := Self (Obj);
   O_2    : Integer       := F (Self (Obj));
   O_3    : Integer       := Self (Obj).Component;
   O_4    : Integer       := F (Self (Ptr.All));
begin
  null;
end Test_Ada95;
```

```
with P;
procedure Test_Ada2005 is
   type Ptr_Obj is access all P.T'Class;
   Obj    : P.T;
   Ptr    : Ptr_Obj := new P.T;

   O_1    : P.TP'Class := Obj.Self;
   O_2    : Integer       := Obj.Self.F;
   O_3    : Integer       := Obj.Self.Component;
   O_4    : Integer       := Ptr.Self.F; -- Implicit
dereference
begin
  null;
end Test_Ada2005;
```

AI-252: Object.Operation notation

```
package P is
    type T is tagged record . . . ;
    procedure Init  (X : access T);
end P;
```

```
with P;
package Q is
    type T_Ptr is access all P.T;
end Q;
```

- **Ada 95**

```
with Q; with P;
procedure Test_2 is
    Ptr : Q.T_Ptr;
begin
    P.Init (Ptr.all);
end Test_2;
```

```
with Q;
procedure Test_2 is
    Ptr : Q.T_Ptr;
begin
   Ptr.Init;        -- accessible!
end Test_2;
```

AI-251: Abstract Interfaces

```
type I1 is interface;
procedure P (A : I1) is abstract;
procecure Q (X : I1) is null;

type I2 is interface I1;
procedure R (X : I2) is abstract;

type T1 is new I2 with . . . ;
--  It must implement P, Q and R

type T2 is tagged record . . . ;
type DT2 is new T2 and I1 and I3 with . . . ;
. . .
type DT3 is new DT2 with . . . ;
--  Inherits all the primitive operations and
interfaces
--  of the ancestor
```

```
procedure Dispatch_Call (O : I1'Class) is
begin
   if O in I2'Class then    -- Run-time check
      R (O);                -- Dispatching call
   else
      P (O);                -- Dispatching call
   end if;
end Dispatch_Call;

-- Dispatching call to predefined operations
I1'Class'Write (. . .)
```

- AI-216: Unchecked unions: variant records with no run-time discriminant
  - AI-216: Unchecked unions: variant records with no run-time discriminant

AI-216: Unchecked Unions: no run-time discriminant

- C

```
struct T_Data {
   char *name;
   union {
      float f_1;
      int   f_2;
   };
};
```

- Ada 2005

```
type T_Data (Discr : Boolean) is
   Name : Interfaces.C.Strings.Char_Ptr;
   case Discr is
      when False =>
            F_1 : Float;
      when True =>
            F_2 : Integer;
    end case;
end record;

pragma Unchecked_Union (T_Data);
```

C unions can be mapped into Ada records

- The Ada Conformity Assessment Test Suite (ACATS) is the test suite used for Ada processor conformity testing
- In addition to the implementation of the new Ada 2005 issues, we have submitted 44 new tests to the ARG that help to verify Ada 2005 compilers

- The Ada Conformity Assessment Test Suite (ACATS) is the test suite used for Ada processor conformity testing
- In addition to the implementation of the new Ada 2005 issues, we have submitted 44 new tests to the ARG that help to verify Ada 2005 compilers

AdaCore
The GNAT Pro Company

Summary

Summary: GNAT and Ada 2005

High Priority AIs: 22 issues

13 fully implemented, one prototype: Ravenscar, Limited with clause, enhancements to access types, object notation, interfaces, and profiles.

Pending: execution-time clocks, task termination procedure, optimization of controlled types, etc.

Hopefully simpler than previous set

Summary: GNAT and Ada 2005

Medium priority AIs: 47 issues

5 implemented in GNAT: private with clauses, limited aggregates, unchecked union, non-null access types, access to constants.

Pending:

- **Heavy implementation work**: nested type extensions, partial parameter lists for formal packages, overriding / non-overriding declarations, protected interfaces, functions returning limited values
- **No implementation work**: vector and matrix operations
- **Needs study**: accessibility rules for generics, default initialized objects, pragma atomic and slices, etc.

Summary: GNAT and Ada 2005

- Ada 2005 issues already available in the GNAT Academic Program (GAP)

- Major GAP objectives:
  - Encourage and prolong the use of Ada in Academia by providing quality-assured software packages, amongs other materials, that facilitate Ada programming for students
  - Create a collaborative platform for the Ada academic community
  - Create stronger links between academia and the professional Ada community

AdaCore
The GNAT Pro Company

GNAT: On the Road to Ada 2005

Edmond Schonberg and Javier Miranda

End of talk

AdaCore
The GNAT Pro Company

Brief Overview of Implemented Ada 2005 Issues

Binding Interpretations

AI-220: Subprograms within private compilation units

```
with A.B.C;   -- Unclear in Ada 95
              --  Not legal in Ada 2005
procedure A.X is
begin
   . . .
end A.X;
```

```
package A is
   . . .
end A;

private package A.B is
   . . .
end A.B;

package A.B.C is
   . . .
end A.B.C;
```

A public declaration can never depend on a private unit

GNAT implemented this rule as it was originally intended in Ada 95 (not as it was written in the Reference Manual)

AI-235: Resolution of 'Access

```
package P is
    procedure Proc (X : access Integer);
    procedure Proc (X : access Float);
end A;
```

- **Ada 95**

```
with P;
procedure AI_235 is
    type Int_Ptr is access all Integer;
    Value : aliased Integer := 10;
begin
     --  qualification needed
     P.Proc (Int_Ptr'(Value'Access));
end AI_235;
```

- **Ada 2005**

```
with P;
procedure AI_235 is
    Value : aliased Integer := 10;
begin
     P.Proc (Value'Access);
end AI_235;
```

In Ada 2005 the prefix of the access attribute resolves the call

AI-310: Ignore non-dispatching operations during overloading

```
package P is
   type Some_Unit is new Float;
   --  Make some predefined operator unavailable to force descendants to
   --  declare their own non-abstract version
   function "*" (Left, Right : Some_Unit) return Some_Unit is abstract;
   function Image (X : Some_Unit) return String;

   type Derived_Unit is new Some_Unit;
   function "*"  (Left, Right : Derived_Unit) return Derived_Unit;
end A;

use P;
X : Some_Unit := 1.0;
S  : String := Image (X * X);   --  Ambiguous in  Ada 95
```

AdaCore
The GNAT Pro Company

GNAT and Ada 2005

Technical Details

AI-216: Unchecked unions: varian records with no run-time discriminant

- Simple implementation available in GNAT for several years
- The notion of *inferrable discriminant* complicates the implementation:

  Initialization, assignment, and equality are all impacted by the possible presence of such discriminants. Temporaries must be created for them, and they must be used selectively in the expansion

- Instead of a simple mechanism to interface to common C unions, this AI makes Unchecked_Union types into full-blown varian records with off-line discriminants (unclear whether this level of complication is justified by the gain in functionality)
- This is a reminder that grafting small semantic changes into a large compiler may have surprisingly complex consequences!

AI-217: Limited With Clause

- GNAT builds the two views:
  - **Non-limited view**
  - **Limited view**
- Visibility analysis uses one of these views
- For code generation purposes, entities in the limited-view reference their counterparts in the non-limited view

Package Specification

```
package Q is
    type T_1; -- Incomplete type declaration

    package Local is
        type T_2 is tagged;
    end Local;
end P;
```

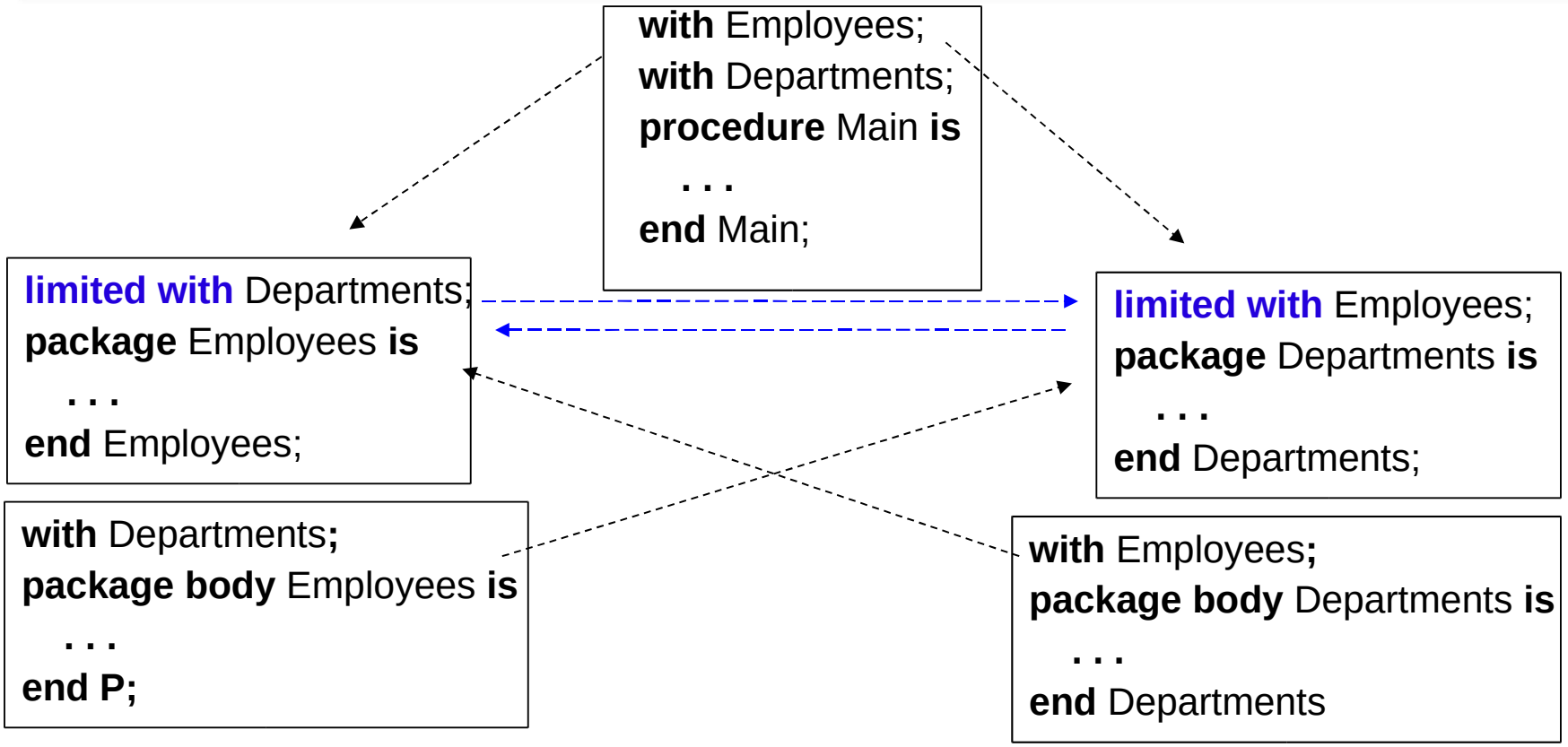
Limited View

```
package Q is
    type T_1 (D : Integer) is record
        . . .
    end record;

    package Local is
        type T_2 is tagged record
            . . .
        end record;
    end Local;
end P;
```

Full View

AI-217: Limited With Clause (example)



AI-262: Private with clause

- In case of private with clause found in a package specification GNAT installs the context clauses in two stages:
  - Non-private with-clauses (before compiling the public part of the package)
  - Private with-clauses (before compiling the private part of the package)
- In case of private with-clause found in a library subprogram the private with-clauses are installed after the specification of the subprogram has been analyzed
- In case of limited-private-with clauses, GNAT builds the incomplete view of the named compilation unit and installs it as described above

AI-230: Generalize anonymous access types

- Relax the strictness of the semantic analyzer to allow the use of access types in:
  - Component definitions (thus covering array types and record components)
  - Discriminants of non-limited types
  - Object renaming declarations

- Incorporate the following operators for the universal-access type in package *Standard*

```
function "="  (Left, Right : Universal_Access) return Boolean;
function "/=" (Left, Right : Universal_Access) return Boolean;
```

AI-230: Generalize anonymous access types

- Set the accesibility level of the anonymous access type:
  - For an access object that cannot be altered during its lifetime (parameter of mode IN or discriminant of a limited type), its level is determined by the accessibility level of its initial value
  - For a component definition or a discriminant of a non-limited type, the level is the same as that of the enclosing composite type
  - For renamings the level is the same as the level of the type of the renamed object

These rules are necessary to simplify the implementation and to avoid dangling references when an access object is updated while being viewed at a deeper level that it truly is.

AI-231: Access to constant parameters and null-excluding access subtypes

- **Access to constant parameters**: the semantic analyzer just has to remember that the designated object is not allowed to be modified

- **Null-excluding access subtypes**
  - Propagation of the null-excluding attribute to subtypes, objects and components
  - Addition of new checks to the semantics to detect bad usages of null-excluding types
  - Generation of the nul-exclusion run-time check when required
  - Relax the semantics to permit the *null* value in anonymous access types

AI-254: Anonymous access to subprogram types

- Ada 2005 rules ensure that accessibility checks are never required for anonymous access to subprograms; thus they don't need to carry an accessibility level
- Given the Ada 2005 semantic rules, anonymous access to subprograms can be represented by its code address (thus allowing easy interfacing with C function pointers)
- Only modification: remove several GNAT semantic checks !

AI-287: Limited Aggregates

- The initialization of limited components of aggregates must be carried out in their final destination ---no copying can take place
- Limited aggregates adds no special complexity to the compiler: **initialization in place** is already required for controlled objects by the ARM (Section 7.6)
- The semantic analysis and expansion of aggregates is an extremely complex portion of the semantics (the initialization of limited components adds infinitesimally to this complexity)
- GNAT converts the aggregate into a set of individual assignments. In case of limited components, we generate calls to default initialization subprograms

AI-249 and AI-305: Real-Time and High-Integrity Issues

- Add no special complexity to the compiler:

If the new restrictions are specified in the source, the front-end increases its strictness and reduces the set of Ada allowed in the applications

AI-252: Object Operation Notation

- Simple support for the basic functionality:

  When the analysis of a selected component fails, instead of immediately generate an error message, the frontend rewrites it using the standard functional Ada notation and repeats the analysis

  Object.Operation (. . .) ----------> Operation (Object, . . .)

- Class-wide calls require more work because the scope of the type of the object does not necessarily designate the scope of the operation: it may be declared in the scope of some ancestor

```
package P is
   type T is tagged record . . . ;
   procedure Init  (X : access T);
end P;
```

```
with P;
package Q is
   type T_Ptr is access all P.T;
end Q;
```

```
with Q;
procedure Test_2
is
   Ptr : Q.T_Ptr;
begin
   Ptr.Init;
end Test_2;
```

AI-251: Abstract Interfaces

- Prototype implementation that uses a combination of dispatch table for the primitive operations of the type, and permutation maps to establish how a given interface is satisfied by existing primitive operations

- We are currently evaluating alternatives that may be more efficient at run-time and simplify interfacing to C++, so that simple cases of multiple inheritance in C++, involving only one non-abstract ancestor can be mapped into Ada hierarchies.

Summary: GNAT and Ada 2005

Working Items: 43 issues

- Ranging from major to trivial
- A few might still be accepted into the corrigendum
- Several are major enhancements – heavy implementation issues
  - Returning limited objects without copying
  - Protected and task interfaces
  - Priority-specific dispatching
  - Support for deadlines and EDF scheduling
- Miscellanea:
  - Container library
  - Tag read by T´Class´Input

How much work to get to GNAT 2005?

- Need to review every entry in Corrigendum 2000
- Need to design and implement remaining Corrigendum 200Y (and a few or the working items)
- Need to develop minimal ACATS tests (44 tests submitted)
- Need to update ASIS

- No major redesign of core technology, but several person-years of work to complete all AI´s
- GNAT Pro can already claim bragging rights for most important AI´s