AdaCore
The GNAT Pro Company

Towards Ada2012 : an interim report from the Ada Rapporteur Group

Ed Schonberg
Adacore Inc

Ada-Europe 2010

Valencia, Spain

The ARG in the greater scheme of things

- **The ARG is in charge of Ada language maintenance and design. Its technical decisions are examined and ratified by**
  - **WG9**,which has national representations, and is Working group nine of
    - **SC22**, the International Standardization Committee for Programming Languages, which is part of
      - **JTC1**: the joint (ISO/IEC) technical committee for Information Technology
      - which answers to two galactic entities:
      - **ISO** (International Standards Organization) and
      - **IEC** (International Electrotechnical Committee)

Activities and decisions of the ARG

- **Driven by user queries/comments and by internal design activities : Ada Issues (AI's)**
- **Ada Comment (a gloss on a technical point)**
- **No Action (live with it)**
- **Confirmation (the RM is correct and clear)**
- **Ramification (the RM is correct but obscure)**
- **Binding Interpretation (The RM has a gap or is wrong)**
- **Amendment : for the next standard of the language**
- **Ongoing: editing the RM and the Annotated RM**

Current scope of activities

- **regular AI's : processed on a rolling basis**
- **Review of ASIS 2005 standard : complete**
- **New amendment: time-bound.**
  - Need to demonstrate language evolution
  - Need to limit size of amendment (workload, ISO issues)
- **WG9 guidelines:**
  - All amendment AI's received by June 2009
  - All corrective AI's received by June 2010
  - Preliminary standard distributed by Nov. 2010
  - Then draft to WG9, vote, SC22, etc.
  - Tentative publication date of new document : Q2 2012.

Amendment highlights

- **Certain to be adopted**
  - Pre- and Post-conditions for subprograms
  - In-out parameters for functions
  - Bounded containers, proper concurrent queues, holder container
  - Better accessibility rules for anonymous access types
  - New concurrency constructs for multicores
  - Conditional and case expressions
  - Quantified expressions

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Program Correctness

- Need a general mechanism to introduce new checkable properties of entities : subprograms, types, subtypes.

- Checking may be static (compiler) or dynamic (assertions)

- AI05-0145 Pre- and Postconditions
- AI05-0146 Type Invariants
- AI05-0153 Subtype predicates
- AI05-0183 Aspect Specifications

Pre- and Postconditions

```
generic
   type Item is private;
package Stack_Interfaces is
    type Stack is interface;
    procedure Push (S : in out Stack; I : in Item) is abstract
      with Pre'Class => not Is_Full(S),
           Post'Class => not Is_Empty(S);
   ...
    function Is_Empty (S : Stack) return Boolean is abstract;
    function Is_Full (S : Stack) return Boolean is abstract;
end Stack_Interfaces;
```

Pre- and Postconditions (2)

- Unified syntax for aspect specifications
- Semantic analysis of conditions is done at end of package (freeze point of subprogram)
- Conditions can be verified dynamically like assertions, or statically by analysis tools and/or clever compilers
- Can specify ***classwide*** conditions and ***type-specific*** conditions. Classwide conditions are inherited by the corresponding primitive of each descendant type.
- Dynamic condition checking is controlled by assertion mode
- Check can be in caller or in callee.

Pre- and postconditions (3)

- **New Attributes, mostly for use in postconditions:**
- X'Old denotes the value of X before subprogram starts execution (X can be an arbitrary expression)
- F'Result denotes the result of the current function call.

Type invariants

```
package Q is
   type T(...) is private
     with Invariant => Is_Valid (T);
   type T2(...) is abstract tagged private
     with Invariant'Class => Is_Valid (T2);
   function Is_Valid (X : T)  return Boolean;
   function Is_Valid (X2 : T2) return Boolean is abstract;
end Q;
```

Type Invariants (2)

- For private types and type extensions.

- ***Classwide*** invariants and ***type-specific*** invariants
- Inheritance follows Liskov's rules

- Invariants are checked:
  - On object initialization
  - On conversion to the type
  - On return from function that creates object of the type
  - On return from subprogram that has (in)- out parameter of the type
- **Not bullet-proof:**
  - Still possible to modify object through access values
  - If invariant for private extension depends on visible inherited component, invariant is at risk.

Subtype Predicates

- Under discussion

```
type Rec is record
   A : Natural;
end record;
subtype Decimal_Rec is Rec
   with Predicate => Rec.A mod 10 = 0;
```

- A predicate can be specified for any subtype
- A predicate is not a constraint (akin to a null exclusion)
- Most common use: non-contiguous enumeration types.

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-times**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Containers

- The Ada2005 library is sparse, compared with those of other languages, and with the state of the art in data-structure design.

- AI05-0001 Bounded containers
- AI05-0069 Holder container
- AI05-0136 Multiway tree container
- AI05-0159 Queue containers
- AI05-0212 Accessors and Iterators for Ada.Containers

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Programming expressiveness

- More powerful functions
- Better iterators on all containers

- AI05-0139 Syntactic sugar for accessors, containers, and iterators
- AI05-0142 Explicitly aliased parameters
- AI05-0143 In Out parameters for functions
- AI05-0144 Detecting dangerous order dependences
- AI05-0177 Renaming expressions as functions

Syntactic sugar for iterators

- **Interface with implicit dereference on access discriminant:**

```
package Ada.References is
   type Reference is limited interface;
end Ada.References;
```

- **Allows indexing over containers:**

```
for Cursor in Iterate (Container) loop
   Container (Cursor) := Container (Cursor) + 1;
end loop;
```

Detecting dangerous order dependences

- In-out parameters for functions and unspecified order of evaluation are a bad combination!
- F (Obj) + G (Obj)
- Is problematic if F and / or G have side-effects on their actuals

- AI provides a precise statically checkable definition of identity and overlap between objects. Compiler can then verify that:

- in a complex expression involving a function call with a modifiable parameter, there is no other component of the expression that denotes the same object or a portion of it.

Renaming expressions as functions

- **Under discussion**

- To simplify the writing of pre/postconditions and predicates, allow parametrized expressions (aka function bodies in package specs):

- **function** Cube (X : integer) **is** (X ** 3) ;

Programming expressiveness (2)

- Flexible syntactic forms for predicates in contracts and elsewhere

- AI05-0147 Conditional expressions
- AI05-0158 Generalizing membership tests
- AI05-0176 Quantified expressions
- AI05-0177 Parametrized expressions
- AI05-0188 Case expressions
- AI05-0191 Aliasing predicates

Conditional Expressions

```
Value:= (if X > Y then F (X) else G (Y));
```

- If result type is Boolean, *else_part* can be omitted.
- Generally parenthesized
- Must work with classwide types and anonymous access types.

Extending membership operations

- **The argument of a membership test can be a set of values:**

- **If** (C **not in** 'A' | 'B' | 'O') **then**
- Put_Line ("invalid blood type");
- **else** …

Quantified expressions

A is sorted:

```
(for all I in A'First .. T'Pred(A'Last) | A (I) <= A (T'Succ (I)))
```

N is composite:

```
(for some X in 2 .. N / 2 | N mod X /= 0)
```

Computation is short-circuited.

some is not a reserved word

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Visibility mechanisms

- Need more flexible ways to name entities in the rather complex environment in which a unit is compiled.
- Incomplete types can be useful in additional contexts

- AI05-0135 "Integrated" nested packages
- AI05-0150 Use all type clause
- AI05-0151 Allow incomplete types as parameter and result types
- AI05-0162 Allow incomplete types to be completed by partial views

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Concurrency and real-time features

- Need to address the multicore revolution
- Better scheduling tools justify more elaborate constructs

- AI05-0117 Memory barriers and Volatile objects
- AI05-0167 Managing affinities on multiprocessors
- AI05-0169 Defining group budgets for multiprocessors
- AI05-0171 Ravenscar Profile for Multiprocessor Systems
- AI05-0166 Yield for non-preemptive dispatching
- AI05-0168 Extended suspension objects
- AI05-0170 Monitoring the time spent in Interrupt Handlers
- AI05-0174 Implement Task barriers in Ada

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Anonymous access types and storage management**
- **Syntactic sweeteners**

Anonymous access types and storage management

- Need to simplify accessibility rules for anonymous types
- Need more flexible storage reclamation mechanisms

- AI05-0148 Accessibility of anonymous access stand-alone objects
- AI05-0149 Access types conversion and membership
- AI05-0152 Restriction No_Anonymous_Allocators
- AI05-0189 Restriction No_Allocators_After_Elaboration
- AI05-0190 Global storage pool controls
- AI05-0193 Alignment of allocators

Major topics

- **Program correctness**
- **Containers**
- **Constructs for expressiveness**
- **Visibility mechanisms**
- **Concurrency and real-time**
- **Syntactic sweeteners**

Syntactic sweeteners

- What, no “continue” statement?
- Where are pragmas legal?

- AI05-0100 Placement of pragmas
- AI05-0163 Pragmas instead of null
- AI05-0179 Labels at end of a sequence_of_statements

The language design imperative

- **"You boil it in sawdust: you salt it in glue:**
  You condense it with locusts and tape:
  Still keeping one principal object in view—
  To preserve its symmetrical shape."

The Hunting of the snark

Discards

- AI05-0074-2 Allowing an explicit "end private;" in a package spec
- AI05-0074-3 Deferred instance freezing
- AI05-0140-1 Identity functions
- AI05-0175-1 Cyclic fixed point types
- AI05-0187-1 Shorthand for assignments with expressions naming target (a += 1)

TO BE CONTINUED!

- Details at
- http://www.ada-auth.org/AI05-SUMMARY.HTML

- Implementors: start your compilers !