

**SofCheck™**

*Software Analysis and Verification*

ISO/IEC JTC1/SC22/WG9 N420

# Ada 200Y -- What and Why

*SIGAda '02*

*December 2002*

*S. Tucker Taft*

*President*

*SofCheck, Inc.*

# Ada is Alive and Evolving



**Ada 83 Mantra: “No Subsets, No Supersets”**

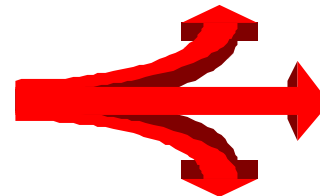


**Ada 95 Mantra: “Portable Power to the Programmer”**

- **Internet, especially Comp.Lang.Ada, Team-Ada fosters...**
  - Active interplay between users, vendors, and language lawyers
  - Open discussion of new ideas and possible language enhancements
- **Availability of open-source GNAT fosters...**
  - Grass roots interest in Ada
  - Additional open-source contributions to compiler and library
  - Experiments with new syntax and semantics

# ISO WG9 and Ada Rapporteur Group

- **Stewards of Ada's Standardization and Evolution**
- **Includes users, vendors, and language lawyers**
- **First "Official" Corrigendum Released 9/2000**
- **Now Focusing on Language "Amendments"**
- **So Which Way do we Go?**



# Overall Goals for Language Evolution

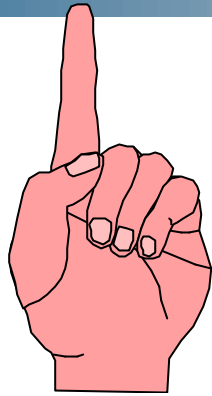
- Enhance Ada's Position as a:
  - Safe
  - High Performance
  - Flexible
  - Portable
  - Accessible
  - Distributed, Concurrent, Real-Time, Object-Oriented Programming Language
- Finish the job of integrating object-oriented concepts into Ada





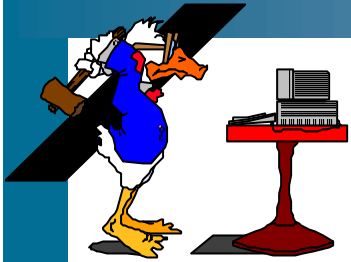
# Safety Is Our Most Important Product

- **Ada is the premier language for safety critical software**
- **Ada's safety features are critical to making Ada such a high-productivity language in all domains**
- **Amendments to Ada should not open any new safety holes**
- **Amendments should provide even more safety, more opportunities for catching mistakes at compile-time.**



# Disclaimer!

- Not all of these proposals are going to make it into 200Y
- Users need to get involved to set priorities, help refine proposals
  - Please participate actively today and in Thursday workshop
- ISO WG9/ARG will be publicizing efforts more during the coming year
  - Starting with this conference!
- Big issues:
  - Who are the real/important users and what do they need/want?
  - How can we keep complexity of understanding and implementation manageable?
  - Upward Compatibility? Upward Consistency? What is “obscure”?



# Possible Safety Amendments

- **Pragma to prevent unintentional overriding or non-overriding of primitive operations**
  - Catch spelling errors, parameter profile mismatches, maintenance confusion (ARG Approved)
- **Standardized Assert Pragma**
  - plus other Pre\_Assert/Post\_Assert/Invariant Pragmas associated with Subprogs, Pkgs, or Types (work item)
- **Pragma/Attributes for specifying physical units associated with particular subtypes**
  - Catch unit inconsistencies in complex computations
- **Configuration Pragma to require initialization of local variables on all paths prior to a use**
  - Match requirements of Java Virtual Machine byte-code verifier; catch a common cause of errors



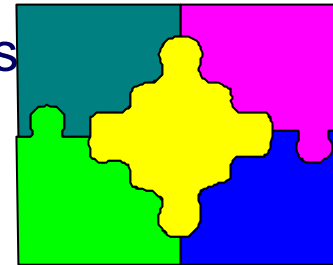
# Why use Pragmas for Safety checks?

- **Pragmas are a natural way to add safety checks**
- **The only effect of an additional safety check is to reject an otherwise legal program**
- **No effect on semantics of programs that survive the check**
- **Pragmas can be associated with:**
  - A single declaration
  - A point in the execution sequence
  - A declarative region
  - A source file or an entire library (configuration pragma)



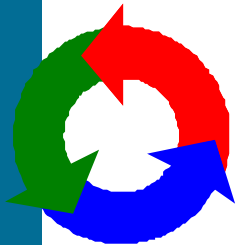
# Dealing with Today's Reality

- **Today's Reality:**
  - The rise in importance of the Java Virtual machine and .Net common runtime
  - Increasingly complex APIs; API Wars
  - Component based systems
  - Multilingual Systems
  - Dynamically Bound Systems
- **Cyclic Dependence between types is the norm in complex O-O systems**
- **Emergence of Notion of “Interface” that can have multiple implementations (CORBA, Java, C#, COM)**
- **Amendments to Ada may help address this reality**



# Enhancing Interoperability with Today's Reality

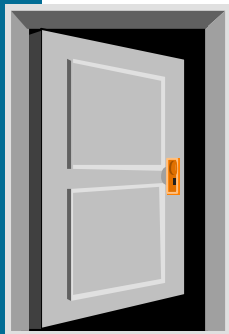
- **Support Cyclic Dependence Between Types in Different Packages**



- Various alternatives considered
- Current approach: “type T is [tagged] separate in P;”
- Also related to anonymous access type proposal

- **Support Notion of “Interface” as used in Java, CORBA, C#, etc.**

- Already supported by Ada->JVM compilers somehow
  - E.g. Pragma Convention(Java\_Interface, T);
  - Plus some magic Compiler-provided bodies for primitives that call same-named op of encloser
- Proposal for “abstract interface” types




# Example based on type “stub” Proposal



```
package Employees is
  type Employee is private;
  type Department is separate in Departments;
  procedure Assign_Employee(E : access Employee;
    D : access Department);
  ...
  type Dept_Ptr is access all Department;
  function Current_Department(D : access constant Employee) return
    Dept_Ptr;
end Employees;
```

```
package Departments is
  type Department is private;
  type Employee is separate in Employees;
  procedure Choose_Manager(D : access Department;
    Manager : access Employee);
  ...
end Departments;
```

# Proposed “Abstract Interface” Amendment

- **Type NT is new T  
with Int1 and Int2 and  
record ... end record;**
  - **Int1 and Int2 are “Interfaces”**
    - Declared as: **Type Int1 is interface;**
    - Similar to **abstract tagged null record** (no data)
    - All primitives must be **abstract** or **null**
  - **NT must provide primitives that match all  
primitives of Int1 and Int2**
    - In other words, NT *implements* Int1 and Int2.
  - **NT is implicitly convertible to Int1’Class and  
Int2’Class, and explicitly convertible back**
    - and as part of dispatching, of course
  - **Membership test can be used to check  
before converting back (narrowing)**
- 

# Example of Abstract Interface

```
package Observers is -- "Observer" pattern
  type Observer is interface;
  type Observer_Ptr is access all Observer'Class;

  type Observed_Obj is tagged separate in Observed_Objects;

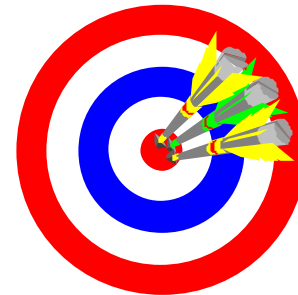
  procedure Notify(O : in out Observer;
    Obj : access Observed_Obj'Class)
    is abstract;

  procedure Set_Next(O : in out Observer; Next : Observer_Ptr)
    is abstract;
  function Next(O : Observer) return Observer_Ptr is abstract;

  type Observer_List is private;
  procedure Add_Observer(List : in out Observer_List;
    O : Observer_Ptr);
  procedure Remove_Observer(List : in out Observer_List;
    O : Observer_Ptr);
  function First_Observer(List : in Observer_List)
    return Observer_Ptr;
```

# Portability Enhancements

- **Ada provides excellent support for building portable code**
- **Ada library still relatively slim; Amendments to define additional standard libraries could enhance portability**
- **Focus should particularly be on ensuring portability for server-side Ada, E. g.:**
  - Files and Directories
  - Sockets
  - HTTP/CGI Servlet interfaces
  - Timezones
  - Environment variables
  - ODBC/JDBC equivalent
- **Based on Posix or Win32, but simplified and made O/S independent**

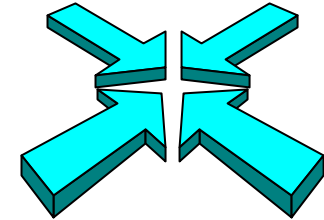


# Enhancing Accessibility to Ada

- **Address Ease of Transition to Ada**
- **No Mandate from Top anymore =>**
  - Ada must be able to infiltrate from bottom or side of organization
  - Need to look at increasingly popular paradigms and frameworks
    - JVM, J2EE, EJB
    - Microsoft COM and .Net
    - XML/XSL
    - ODBC/JDBC
    - HTTP/Servlet
- **UML-ish Modeling Increasingly Popular**
  - Needs to be easy to go between UML and Ada
- **Full integration of Object Oriented concepts**



# Possible Accessibility Amendments



- **Cyclic dependence (type stub) amendment**
- **Multiple “Interface” concept**
- **Object.Operation(...) syntax for calling user-defined primitives; e.g.:**

```
package P is
    type T is tagged private;
    procedure Update(
        X : in out T;
        Y : Whatever);
end P;
A : P.T;
...
P.Update(A, What); => A.Update(What);
```
- **Generalized use of anonymous access types**
- **Extensible Protected types**



# Object.Operation syntax (cont'd)

- More familiar to users of other object-oriented languages
- Reduces need for extensive use of “use” clause
- Allows for uniform reference to dispatching operations and class-wide operations, on ptrs or objects; e.g.:

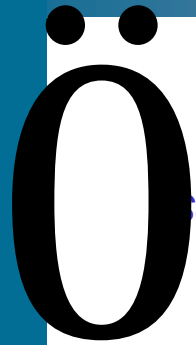
```
package Windows is
  type Root_Window is tagged private;
  procedure Notify_Observers(Win : Root_Window'Class);
  procedure Display(Win : Root_Window) is abstract;
  ...
end Windows;
package Borders is
  type Bordered_Window is new Windows.Root_Window with private;
  procedure Display(Win : Bordered_Window) is abstract;
  ...
```

```
BW: access Bordered_Window'Class;
BW.Display; BW.Notify_Observers; -- both of these “work”
```

# Generalized use of Anonymous Access types



- Two kinds of generalization
  - Allow access “parameters” for access-to-constant and access-to-subprogram cases
  - Allow use of anonymous access types in components and stand-alone variables
- Should help reduce “noise” associated with unnecessary explicit conversions of access values
- Also allow optional specification of “not null” constraint on access subtypes, and anonymous access type specifications
  - E.g.: type String\_Ref is access all String not null;
  - Improves safety, efficiency, and documentation by pushing check for null to caller or assigner rather than ultimate point of use.



***SofCheck***<sup>TM</sup>

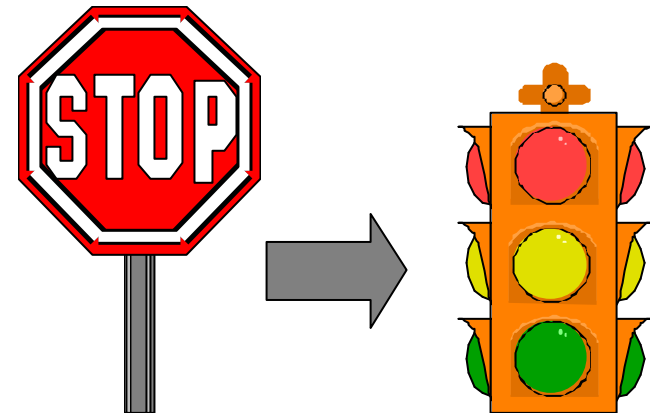
*Software Analysis and Öerification*

## **Other Ada 200Y Amendments Under Consideration**

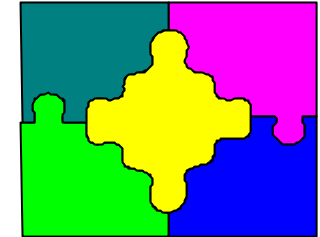
# Extensible Protected types

- This was considered during Ada 9X
  - Felt to be too risky given that both tagged types and protected types were new concepts
- Time may be right to integrate the two capabilities, e.g.:

```
protected type Sem_With_Caution_Period is
new Semaphore with
  function Is_In_Caution_Period
  return Boolean;
  procedure Release_With_Caution;
private
  In_Caution_period : Boolean := False;
end Sem_With_Caution_Period;
```



# Generalize Formal Package Parameters



- Allow partial specification of actual parameters
  - Currently it is all or nothing
  - Important when there are two formal package parameters that need to be “linked” partially through their actual parameters
- Example

**generic**

**with package I1 is new G1(<>);**

**with package I2 is new G2(**

**Element => I1.Element, others => <>);**

**package New\_Abstraction is ...**

# Make Limited Types Less Limited



- Easier: Allow use of explicitly initialized limited objects, where initial value is an aggregate.
  - Aggregate is built in place (as it is now for controlled types)
  - Define new syntax to represent “implement by default”
    - Use “<>” for this, corresponds to notion of “unspecified”
  - Still no copying allowed, and no assignment statements
  - Aggregates can be used as initial expression for declaration, as expression for initialized allocator, and as actual parameter value
- Harder: Allow functions to return limited objects
  - Return statement must return an aggregate or function call
  - Function call can be used where aggregate is proposed to be allowed above
  - Must give up on return-by-reference of Ada 95?

# Other Possible Goodies...

- Pragma Pure\_Function (from GNAT)
- Nonreserved Keywords (e.g. “Interface”)
- Controlling ‘Read/’Write of Tags
- Additional Standard Restrictions and a Standard “Profile” for Ravenscar
- “private with A.B;” -- A.B only visible in private part
- Downward closures -- local subprograms can be passed as parameters to global subprograms
  - Uses anonymous access-to-subprogram types for parameters.
- Task termination handlers
  - especially for termination due to unhandled exceptions

# Which Way Do We Want to Go?



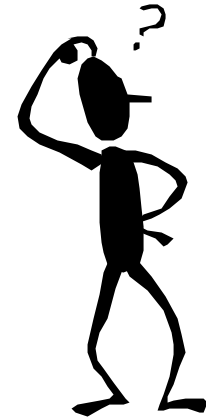
- **Should learn from new languages and other programming paradigm developments**
  - No good model for multiple inheritance during Ada 9x process, but now multiple interface inheritance has emerged as good compromise
  - UML establishing OO at design-time as well as at code time
  - Useful Concurrent and Distributed OO models beginning to emerge
- **Should not ignore marketing and transition issues**
  - E.g. Object.Operation(...) syntax might help preserve OO view
- **Should keep our core “values” in mind**
  - Safety, High Performance, Portability



# What can we afford?

- From an implementation point of view
  - Minimize syntax changes
    - Standardize packages, attributes, and pragmas
  - Keep semantics “straightforward”
  - Do trial implementations
    - E.g. 127 lines to support Object.Op in GNAT for tagged types (according to Martin Carlisle)
- From a language complexity point of view
  - Try to enhance by simplifying
  - Remove unnecessary restrictions
  - Support “natural” extensions
  - Use paradigms familiar from and well tested in other languages

# ARG is looking for well-formed proposals



- Packages worth standardizing
  - Two groups already working on this => join in
- Pragmas/Attributes worth standardizing
  - Identify existing compiler-specific features that should be more portable
- Elimination of unnecessary restrictions, implementation dependencies, and inappropriate “erroneous” or “bounded error” situations, etc.
- Write to [Ada-Comment@ada-auth.org](mailto:Ada-Comment@ada-auth.org)
- Participate in Thursday workshop.
- Speak up now!

# Two Discussion Groups

- ALIOOOP Group
    - Ada Linguists Interested Only in OOP
  - Type Stub
  - Interfaces
  - Object.Operation
  - Anonymous Access Types; not null access types
  - Limited Less Limited
  - Relaxing Freezing in Generics
  - Partially Parameterized Formal Packages
  - Asserts/Pre/Post/Invariants
- IRONMASCC Task
    - I Really Only Need Mission And Safety Critical Computing
  - Returning to our roots; MASC issues
  - Ravenscar Profile and associated Restrictions and Policies
  - Task Termination Handling
  - Extensible Protected Types
  - Future of Distribution Annex
  - Other High Integ/Real-Time
  - Asserts/Pre/Post/Invariants