Language Issues for Ada’s Future

SIGAda 2000

S. Tucker Taft  stt@averstar.com
AverStar, Inc.
Burlington, MA
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
- 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
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.
Possible Safety Amendments

• Pragma to prevent unintentional overriding or non-overriding of primitive operations
  – Catch spelling errors, parameter profile mismatches, maintenance confusion

• Standardized Assert Pragma
  – plus other Precondition/Postcondition/Invariant Pragmas associated with Subprograms or Types

• 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 program semantics that survives 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
  – 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: “**with type** P.T [is tagged];”
  - Related to T’Class_Access class-wide access type proposal

- **Support Notion of “Interface” as used in Java, CORBA, C#, etc.**
  - No concrete proposals yet
  - 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
Example of "with type"
Proposal

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

with type Employees.Employee;
package Departments is
  type Department is private;
  procedure Choose_Manager(D : access Department;
      Manager : access Employees.Employee);
...
end Departments;
Conceivable “Interface” Amendment

- Type NT is new T with Int1 and Int2 and record … end record;
- Int1 and Int2 are “Interfaces”
  - Must be abstract tagged null record (no data)
  - All primitives must be abstract
  - May want new reserved word interface for these.
- 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)
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 probably be on ensuring portability for server-side Ada, E. g.:
  – Files and Directories
  – Sockets
  – HTTP/CGI Servlet interfaces
  – Timezones
  – Environment variables
  – ODBC equivalent
• Might be mostly a “blessed” subset of Posix
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, EJB
    - Microsoft COM and .Net
    - CORBA
    - ODBC/JDBC
    - HTTP/Servlet
- **UML-ish Modeling Increasingly Popular**
  - Needs to be easy to go back and forth between UML and Ada
Possible Accessibility Amendments

- Cyclic dependence (with type) 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);
    ```
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