Ada Seminar

An Introduction
The History of Ada

What?  
Who?  
When?  
Why?  
How?
The World Software Crisis

- US DoD, world largest computer user
- Complexity - Tri-forces
  - Embedded systems
  - Databases
  - Processing
  - Real-time
  - Scientific
• Variety of programming languages and variant used:
  • TACPOL, CMS-2Y, CMS-2M, SPL-1, Jovial J3, Jovial J73, FORTRAN, COBOL, CORAL 66, Pascal, PL/I, PL/M, ALGOL 60, ALGOL 68, Simula, Modula, CHILL, Mesa, CSP, PEARL, APL, Linda, and etc...

• Transportability
• Complexity - led to
  • escalating costs both development and maintenance

• Reliability - Mission / Safety Critical

• Other part of the world having same experience
Identified software crisis symptoms:

- Responsiveness
- Reliability
- Cost
- Modifiability
- Timeliness
- Transportability
- Maintainability
- Efficiency
Solutions?

- Single standard is needed
  - General computing
  - Embedded and Real-Time applications
  - Scientific engineering
Solution!

- 1975 - HOLWG
  - Straw man
  - Wooden man

- 1976 - Tin man
  - Iron man
    - 17 proposals, 4 selected
      - Green (CII Honeywell Bull)
      - Red (Intermetrics)
      - Blue (Softech)
      - Yellow (SRI International)
Solution!

- 1978 - Steel man
- Red
- Green

- May 2, 1979 - **GREEN** team won! CII Honeywell Bull led by French engineer, Jean Ichbiah
Ada was born!

- Pascal-like syntax
- Concurrency
- Object-based
- Named after Augusta Ada Byron (1815 - 1852), Countess of Lovelace, daughter of Lord Byron
Ada, not ADA!
Ada and Standardization

- July 1980 - Definitive version proposed to ANSI
- December 12, 1980 - MIL-STD-1815
- January 1983 - Awarded ANSI standard - Ada 83
- 1987 - ISO Standard 8652
- 1988 - Ada 9X revision
- 1995 - Ada 95 (Object-Oriented) - ISO/IEC 8652:1995
- 2000 - Ada 200X revision
Ada 2005 New Features

- Language features
  - Character set - Support new 32-bit character type, Wide_Wide_Character in source code, e.g. identifiers and comments
  - \( \pi \) : constant := Pi;
- Interface
  - Abstract interfaces to provide multiple inheritance
  - Protected and task interfaces
- Union
  - Unchecked unions using variant records with no run-time discriminant
  - Annex B.3.3 Pragma Unchecked_Union
- With
  - Limited With Clauses
  - Access to private units in private part
- Access types
  - Not null access
  - Anonymous access
Ada 2005 New Features

- Language library
  - Containers
- Scan Filesystem Directories and Environment Variables
  - Directory Operations
  - Environment variables
- Numerics
  - Add Greek pi ($\pi$) to Ada.Numerics
  - Vector and matrix operations
  - Non-Preemptive Dispatching
- Dynamic priorities for protected objects
  - Dynamic ceiling priorities
Ada 2005 New Features

• Real-Time and High Integrity Systems
  • Timing events
  • Execution-Time Clocks
  • Group execution-time budgets
  • Task termination procedure
  • Further functions returning Time_Span values
• Ravenscar profile
  • Ravenscar profile for high-integrity systems
  • New pragma and additional restriction identifiers for real-time systems
  • Annex H
  • Partition Elaboration Policy for High-Integrity Systems
• New scheduling policies
  • Priority Specific Dispatching including Round Robin
  • Support for Deadlines and Earliest Deadlines First Scheduling
  • Non-Preemptive Dispatching
• Dynamic priorities for protected objects
  • Dynamic ceiling priorities
Why Ada?

When failure is not an option!
Ada Characteristics

- Expressive - Primitive objects and operations and must be extensible
- Reliability - Strong typing, enumerated data, capture errors during compilation
- Language implementation of tasking
- Language extension by annexes
- Portability, readability and maintainability
- Built on top of software engineering philosophy
- Validated
- Time-to-market
package Class_Stack is

   type Stack is tagged private;

   procedure reset (the : in out Stack);
   procedure push (the : in out Stack; item : in Integer);
   procedure pop (the : in out Stack; item : out Integer);

private
   type Stack_Index is new Integer range 0 .. 10;
   type Stack_Array is array (1 .. Stack_Index’Last) of Integer;

   type Stack is tagged record
      Element  : Stack_Array;
      Position : Stack_Index := 0;
   end record;

end Class_Stack;
Expressive

• Structured definitions of objects and operations using Package

• Package must be extensible using Genericity

generic
  type T is private;
  Max_Stack : in Positive := 10;
package Class_Stack is

  type Stack is tagged private;

  procedure reset (the : in out Stack);
  procedure push (the : in out Stack; item : in T);
  procedure pop (the : in out Stack; item : out T);

private
  type Stack_Index is new Integer range 0 .. Max_Stack;
  type Stack_Array is array (1 .. Max_Stack) of T;

  type Stack is tagged record
    Element   : Stack_Array;
    Position : Stack_Index := 0;
  end record;

end Class_Stack;
package Class_Stack is

  type Stack is tagged private;

  procedure reset (the : in out Stack);
  procedure push (the : in out Stack; item : in T);
  procedure pop (the : in out Stack; item : out T);

private

  type Stack_Index is new Integer range 0 .. Max_Stack;
  type Stack_Array is array (1 .. Max_Stack) of T;

  type Stack is tagged record
    Element  : Stack_Array;
    Position : Stack_Index := 0;
  end record;

end Class_Stack;

with Class_Stack;
...

package Integer_Stack is new Class_Stack (Integer);
...

package Float_Stack is new Class_Stack (Float, 50);
Reliability

- Strong typing
- Range constraints
- Enumerated data types
- Arrays begin at 1
- Array aggregates
  - A : array (1 .. 6) of Float := (1 .. 5 => 0.0, 6 => 1.0);
- Safe variables passing
  - procedure Set (A : in Integer; B : out Integer; C : in out Integer);
- Capture errors during compilation
Tasking

- Language implementation of *tasking* (process) rather than system primitive *threads*
- Rendezvous instead of semaphores
Language Extension

• Annex A - Predefined Language Environment
• Annex B - Interface to Other Languages
• Annex C - Systems Programming
• Annex D - Real-Time Systems
• Annex E - Distributed Systems
• Annex F - Information Systems
• Annex G - Numerics
• Annex H - High Integrity Systems
Readability

• Pascal-like syntax
• Natural
• Top-down
• Clean
• No cryptic symbols
• Clear definitions
• Easy maintenance
Time-To-Market

- Definition of Time-To-Market

- Timely delivery of reliable software vs. deliver first, patch later

- Force programmers/software engineers spend more time on design

- 10% time spent on coding

- Also allow rapid prototyping
Software Engineering Philosophy

• Built on top of strong software philosophy
  • Enforce strong disciplines in software engineering
    • On-The-Fly programming is also allowed (rapid prototyping)
  • Language for both implementation and design
    • Directly reflects structure of the problem space
• Efforts of software engineers around the world
  • Not a design of one or two individuals
  • Technology NOT proprietary to one organization
• Matured
  • MIL-STD
  • Full ANSI and ISO
Validation

- Fully validated
- Validation suite
Examples ...
Example 1 - Readability

Calculate (int *a, *b)
{
    int c, d;
    c=*a++;
    d=c*a/*b;
    a=d+(*a)-b;
}

procedure Calculate (a : in out Integer; b : in Integer) is
    c, d : Integer;
begin
    c := a;
    a := a + 1;
    d := c * a / b;
    a := d + a - b;
end Calculate;
**Example 2 - Reliability**

```c
int Matrix[10];
...
Max = 15;
...
do_something_dangerous (int *matrix, int max)
{
    for (i=0; i++; i<=Max)
    {
        safety_value = Matrix[i];
        ...
    }
}
```

Matrix : array (1..10) of Integer;
...
Max : Integer := 15;
...

```
for I in 1 .. Max loop
begin
    Safety_Value := Matrix (I);   -- Constraint error raised by compiler
    ...
end;
```
Example 3 - Reliability

```plaintext
green=1, red=2

... color traffic_light = green;
... do_something();
...
if (traffic_light == green)
    start_train();
else
    wait();
...

do_someting()
{
    traffic_light = 1; /* Red */
}

type Color is enum (Green, Red);
...
Traffic_Light : Color := Red;
...
Traffic_Light := 1; -- Compiler gives error
...
if Traffic_Light = Green then
    Start_Train;
else
    Wait;
end if;
```
Example 4 - Tasking

```c
void do_some_process()
{
    Tid1 = Create_Thread (&process_1);
    Tid2 = Create_Thread (&process_2);
    ...
    Start_Thread (Tid1);
    Start_Thread (Tid2);
    ...
    Kill_Thread (Tid2);
}

procedure Do_Some_Process is
    task Process_1 is
        begin
            ...
        end Process_1;
    task Process_2 is
        begin
            ...
        end Process_2;
    begin
        Do_Whatever_Here;
    end Do_Some_Process;
```
Who use Ada?
Military

- C4I
- Battle Theater Management System
- Mission Planning
- Weapon Control
- Missiles
- Warships
- Satellites
- Simulations / War Games
- more...
Aerospace Industries

- NASA
- Boeing
- Lockheed Martin
- British Aerospace
- Airbus
- Air Traffic Control
- GPS
Transports

• TGV Bullet Trains
• Railway Traffic Control (European Train Control)
• Paris Subway
• Cairo & Calcutta Metros
Financial

- NASDAQ, Dow Jones, DAX, etc...
- Swiss Bank (International Fund Transfer, SWIFT)
- Well Fargo Bank
Safety and Testing

- Nuclear Power Plants
- Pratt & Whitney
Medical

- X-ray, CT scan, MRI
- Heart pacer
Others

- Pollution Monitoring
- Video Security
- Radio Telescope
- Video Editing
- Fault-Tolerant System
- Sharp Zaurus-Mac OS X Sync Software
- Network Cooperative Software Development Daemon (CVS)
- Web Applications
Countries that use Ada

- US, Canada, Mexico, Brazil
- UK, France, Germany, Austria, Poland, Italy, Norway, Russia, Spain, Sweden, Switzerland, Belgium, Portugal, Finland
- Japan, China, India, Pakistan, South Korea, Taiwan, Hong Kong, Vietnam, Singapore, Australia, Malaysia(?)
References

- Steelman - Requirements for High Order Computer Languages, Department of Defense (http://www.xgc.com/manuals/steelman/t1.html)
Q & A

Thank you!

Adrian Hoe
adrianhoe.com
adastarinformatics.com