A#: Multilanguage Programming with Ada in .NET

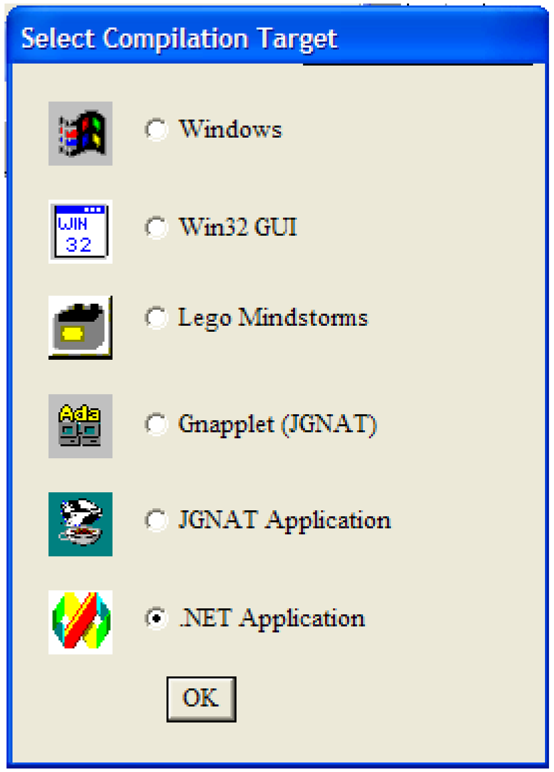
Prof. Martin C. Carlisle, PhD
Associate Professor of Computer Science
United States Air Force Academy

Getting Started - Downloads

- .NET Framework SDK (1.1)
  - Install Redistributable, then SDK
  - http://msdn.microsoft.com/netframework/downloads/
- MGNAT (Ada compiler for .NET)
- MSIL2Ada (creates Ada specs for .NET files)
  - http://www.usafa.af.mil/dfcs/bios/mcc_html/a_sharp.html

Getting Started – Ada IDE

- AdaGIDE fully integrated with MGNAT. Use target button to select .NET Framework
  - http://www.usafa.af.mil/dfcs/bios/mcc_html/adagide.html
- AdaGIDE depends on GNAT for Windows (even if you're only planning to compile to .NET)
  - ftp://ftp.cs.nyu.edu/pub/gnat/3.15p/winnt


Select Compilation Target

- Windows
- Win32 GUI
- Lego Mindstorms
- Gnapplet (JGNAT)
- JGNAT Application
- [✓ .NET Application]

Getting Started – C# IDE

- For MS GUI Designer or mixed-language proramming
  - Visual Studio ($$)
    - http://msdn.microsoft.com/vstudio/
  - #develop (free)
    - http://www.icsharpcode.net/OpenSource/SD/
- Autocompletion is better in VS .NET
  - Finds user defined classes in addition to .NET Framework classes

RAPID – Ada GUI Designer

- Very simple.
- Multi-implementation (JVM, .NET, Gtk, Tcl/Tk), multi-platform.
- Multi-platform.
- Download
  - ftp://ftp.usafa.af.mil/pub/dfcs/carlisle/usafa/rapid/index.html
- .NET executable in rapid.net subfolder

Getting Started – Setup A#

1. Unzip MGNAT
2. Add mgnat\bin to PATH
3. C:\Windows\Microsoft.NET\Framework\v1.1.4322 (location of ILASM.exe) – add to PATH
4. C:\Program Files\Microsoft.NET\SDK\v1.1\Bin

   (location of GACUTIL.EXE) – add to PATH
5. Add to Registry
   - HKEY_LOCAL_MACHINE\Software\ Ada Core Technologies\MGNAT\Root = "c:\mgnat" (or unzip location)
   - HKEY_LOCAL_MACHINE\Software\ Ada Core Technologies\MGNAT\Standard Libraries\DOTNET = "c:\mgnat\include"
6. Run register_mgnat.bat in mgnat\dll
7. Run compile.bat in mgnat\include

Hello World – Take 1

```
with Ada.Text_IO;
use Ada.Text_IO;
procedure Hello_Dotnet is
begin
   Put_Line(Item => “Hello .NET world!”);
end Hello_Dotnet;
```

Hello World – Take 1

- Pretty boring– hard to tell this program uses .NET at all!
- See mgnat\include for the Ada specs for the standard .NET libraries (lots of them).
- Find help by using SDK Documentation

Hello World – Take 2

```
with MSSyst.Windows.Forms;
use MSSyst.Windows.Forms;
with MSSyst.Windows.Forms.MessageBox;
with MSSyst.Windows.Forms.DialogResult;
procedure Hello_Dotnet2 is
   Result : DialogResult.ValueType;
begin
   Result := MessageBox.Show(“Hello .NET!”);
end Hello_Dotnet2;
```

.NET vs Ada Strings

- Error!: operator "+" not defined for type "Standard.String", use of MsSyst.String will fix
- .NET Strings (unicode) are different from Ada strings, but MGNAT will automatically convert
- Need to add:

```
with MSSyst.String;
use MSSyst.String;
```

.NET vs. Ada Strings

- Can convert a .NET String to Ada or vice-versa using "+"

```
procedure Bob(X : in MSSyst.String.Ref) is
   Y : String := +X & " was a .NET string";
   Z : MSSyst.String.Ref;
begin
   Z := +Y;
end Bob;
```

- Automatically done (Ada → .NET) in procedure calls

MGNAT .NET Type/Package Names

- System is replaced by MSSyst:
  - MSSyst.Windows.Forms
- Valuetype – .NET has pass-by-value types:
  - Look for
    - “public enum” (e.g. System.Windows.Forms.DialogResult => MSSyst.Windows.Forms.DialogResult.ValueType)
    - “public struct” (e.g. System.Drawing.Rectangle)
- Ref – used for .NET classes
  - Look for
    - “public class” (e.g. System.Windows.Forms.Form => MSSyst.Windows.Forms.Form.Ref)

.NET Enumerations – Part 1

- Look like Ada enumerations, but…
  - Can add them together to create unnamed values
- Mapped to Ada enumeration types

```
package MSSyst.Windows.Forms.DialogResult is
   type ValueType is (None, OK, Cancel,  Abort_k,
      Retry, Ignore, Yes, No);
   pragma Convention(MSIL,ValueType);
```

- Note use of Abort_k for Ada reserved word

.NET Enumerations – Part 2

```
for ValueType use (
        None => 16#00000000#,
        OK => 16#00000001#,
        Cancel => 16#00000002#,
        Abort_k => 16#00000003#,
        Retry => 16#00000004#,
        Ignore => 16#00000005#,
        Yes => 16#00000006#,
        No => 16#00000007#   );
    function "+" (L,R : Valuetype) return Valuetype;
    pragma Import (MSIL, "+", "+");
```

.NET Enumerations - Restrictions

- See previous for adding .NET enumerations
- Although mapped to Ada enumerations, the ‘Image and ‘Value functions don’t work for .NET enumeration types (use Enum.GetName method if needed)
- ‘Image does work for Ada enumerations

.NET structs

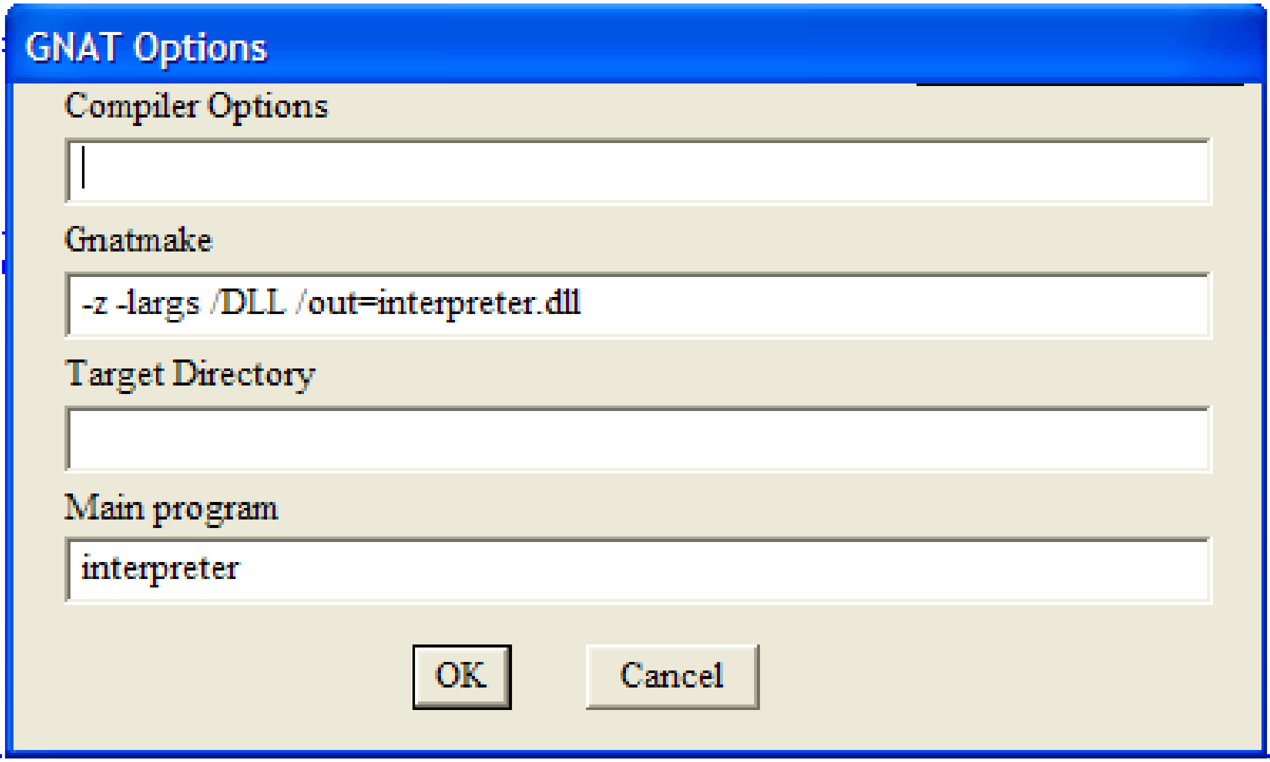
- A .NET Struct will map to Ada as a tagged null record
- .NET structs have properties, which in Ada map to Get functions and Set procedures
  - E.g. in Rectangle:

```
function Get_Left(This : Valuetype) return Integer;
procedure Set_Y(This : Valuetype; Value: Integer);
```

- You'd expect This to be "out" in Set_Y but…

Creating a .NET DLL using AdaGIDE

- Name DLL same as top-level package
- Use Tools/GNAT Options in Current Directory



Compiler Options

Gnatmake: -z -largs /DLL /out=interpreter.dll

Target Directory

Main program: interpreter

Creating a .NET DLL using AdaGIDE

- Sample in simple_net_dll, used in simple_counter
- simple_counter is a #develop project

Adding .NET DLL to C# project

- Right click on References and add reference
- Browse to DLL
- Call adainit in Main method: ada_packagename_pkg.adainit();

Calling back to C# from Ada

- Run msil2ada on C# executable
  - msil2ada simple_counter.il simple_counter
- Now can reference back into C# using generated .ads file
- See examples in simple_net_dll_callback and simple_counter_callback

Object.Method Syntax

- A# supports object.method syntax (proposed for Ada 2005)
- Must use –gnatX flag in compilation (AdaGIDE: Tools/GNAT options in current directory)
- See object_dot_method folder for example
- Requires dispatching parameter be first

Extending a .NET class in Ada- Interfaces

- Implementing Interfaces

type Typ(I_IContainerControl : IContainerControl.Ref) is new …

-- means that this type implements the IContainerControl interface

-- IContainerControl.Typ was defined with

-- pragma MSIL_Interface(Typ);

Extending a .NET class in Ada – pragmas, constructors

- Mark the type with convention MSIL

```
  type Typ(…) is new Form.Typ with null record;
  type Ref is access all Typ'Class;
  function New_Form(This : Ref := null) return
     Ref;
private
  pragma Convention(MSIL,Typ);
  pragma MSIL_Constructor(New_Form);
```

Extending a .NET class in Ada-constructors

- First thing a constructor must do is call a parent constructor– special syntax:

```
function New_Form(This : Ref :=null) return Ref is
   Super : Form.Ref := Form.New_Form(This));
begin
   return This;
end New_Form;
```

- Note that Super must be defined (first), but is never used. Returns “This”, which appears to be null.
- A bit of “compiler magic” here!

Extending a .NET class in Ada- warnings

- Compiler will issue a warning when:
  - Super is defined in constructor
    - Part of compiler that recognizes unused variables doesn’t recognize the special constructor syntax
  - You call a superclass method without a type cast
  - Using an unconstrained array as a parameter to a Convention MSIL type
    - This maps to three parameters and is awkward to call from another .NET language

European Mirror Sites

- AdaGIDE:
  - ftp://sunsite.informatik.rwth-aachen.de/pub/mirror/ftp.usafa.af.mil/pub/dfcs/carlisle/adagide/
- A#
  - ftp://sunsite.informatik.rwth-aachen.de/pub/mirror/ftp.usafa.af.mil/pub/dfcs/carlisle/asharp/
- RAPID
  - ftp://sunsite.informatik.rwth-aachen.de/pub/mirror/ftp.usafa.af.mil/pub/dfcs/carlisle/usafa/rapid