Multilanguage Programming with Ada in the .Net Environment

Jeffrey W. Humphries, Martin C. Carlisle and Terry A. Wilson
Department of Computer Science
2354 Fairchild Dr, Ste 6G101
USAFA, CO 80840-6234
1-719-333-3590
Jeffrey.Humphries,Martin.Carlisle@usafa.edu

ABSTRACT
This paper describes our experiences in using Ada with other programming languages in the .NET environment. This paper explains our approach and presents lessons learned during our development of a real-world software project using .NET. We compare and contrast the languages used, justify our language choices, and present details of our efforts.

Categories and Subject Descriptors
D.3.2 [**Programming Languages**]: Language Classifications – *object-oriented languages.*

D.1.5 [**Programming Techniques]:** Object-oriented Programming.

General Terms: Languages

Keywords: Microsoft .NET environment, Multilanguage programming, Ada 95, A#

1. INTRODUCTION
Microsoft's .NET environment provides a large set of object-oriented libraries for application development. [5,6] It is a relatively new framework for programming under the Windows operating system. One of the central goals of .NET was to provide language interoperability. This paper describes our use of Ada with other programming languages in the .NET environment and explains our approach in using multiple languages in the development of a real-world software project. We evaluate each language used, justify our language choices, and describe our overall experience.

2. MICROSOFT'S .NET ENVIRONMENT
Through its .NET environment, Microsoft provides language independent development coupled with the potential of platform independent execution. Their Common Language Runtime (CLR) is the central component of .NET in that it provides the environment in which programs are executed [3]. The CLR also provides developers with a variety of several different programming languages such as C++, C#, Jscript, Visual Basic, and Perl [4,5]. Other languages are continually being added, such as support for Ada (also known as A#) [1].

Each language that runs under .NET must be compiled into the Microsoft Intermediate Language (MSIL) in order to execute on different platforms. The MSIL files produced by one language are identical to the MSIL files produced by other .NET languages – the CLR does not differentiate between them [3]. The MSIL is then compiled using the Just-In-Time (JIT) compiler specific to each runtime platform [4]. The resulting machine code is then executed by the machine's processor.

The main advantage of using the .NET is language independence. Microsoft also seeks to provide platform independence with .NET, but this has not fully developed. In addition, .NET supports language integration – meaning that classes, exceptions, and polymorphism, for example, function correctly across different languages [3]. Because of this language integration, any language that supports the CLR can support the same set of features [3,4,5]. These advantages make the .NET platform an appealing target for many applications where multiple languages are used.

3. DESCRIPTION OF PROJECT: RAPTOR
3.1 What is RAPTOR?
RAPTOR is a simple-to-use problem solving tool that enables the user to generate executable flowcharts (see Figure 1). RAPTOR was written for students being introduced to the computing discipline in order to develop problem solving skills and improve algorithmic thinking. RAPTOR introduces students to programming concepts and constructs without the burden of learning a detailed language syntax and development environment.

RAPTOR presents the student with a graphical user interface with four major areas. The Symbols area in the upper left presents the six primary graphical symbols that can be used when building a flowchart. The area immediately below the Symbols area is the Watch Window. This area allows the user to view the current contents of any variables and arrays as the flowchart is executing. The large, white area to the right is the primary Workspace. Users can build their flowcharts in this area and watch them update as they execute. The final area is the menu and toolbar. This area allows the user to change settings and control the view and execution of individual flowcharts.

[1] Work performed while this author was a member of the faculty at the US Air Force Academy

SIGAda'03, December 7–11, 2003, San Diego, California, USA.

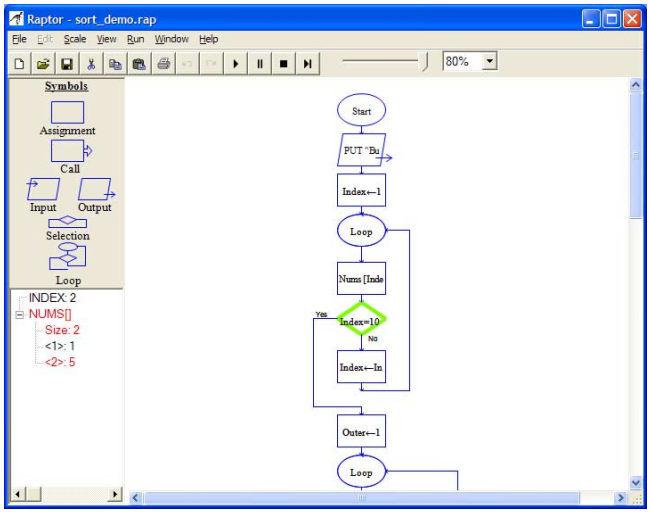

Figure 1 – RAPTOR Main Window

The GUI provides the full functionality expected of any graphical application, including context sensitive menus, tooltips, and user customization. In addition, the GUI provides feedback to the user as their flowcharts execute, allowing them to watch, slow, pause, and reset their "programs".

3.2 Our Approach

We used three different languages in the development of RAPTOR. Ada was used primarily for the lexer, parser, and interpreter of the simple syntax used in RAPTOR. C# was used for building the graphical user interface, using .NET's standard GUI elements, such as forms, dialog boxes, menus, etc. C# was also used for monitoring and executing the runtime system which allows flowcharts to execute. Finally, a legacy C++ library was used for providing a set of graphics routines that allows students to build flowcharts that perform graphical operations (e.g. draw circles, boxes, lines, etc). Figure 2 illustrates how the different languages interoperate. The interoperability DLL is automatically generated by Visual Studio .NET from the C++ COM DLL.

All development was done using Microsoft's Visual Studio .NET and AdaGIDE [2]. Visual Studio provides a full-featured graphical user interface for program design. Particularly useful are the automatic packing of widgets (widgets grow and shrink automatically with the window) and the treeview widget (watch window). These features are not available in other Ada GUI developments environments, such as RAPID.

3.3 Comparison of .NET Languages Used

3.3.1 Ada in .NET (A#)

Ada was the first language to include mixed-language pragmas as part of its specification, allowing it to easily interface with other programming languages. A# is a modification of Ada that sought to create a fully-interoperable environment for an Ada programmer using .NET [1]. Ada programmers can use libraries written by other .NET programmers programming in other languages, and also share their libraries with programmers using other languages [1].

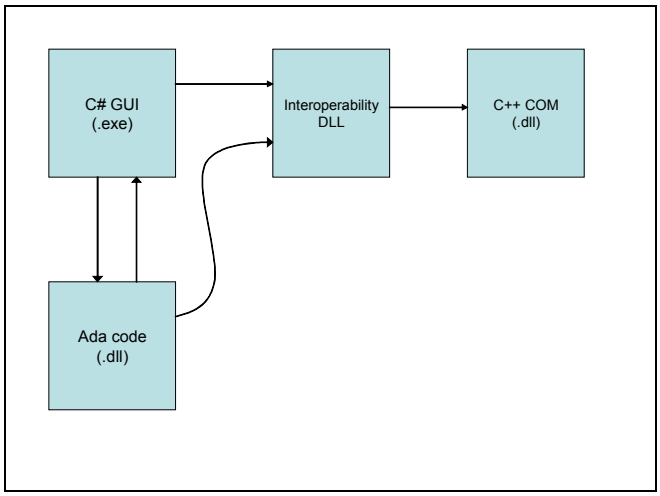

Figure 2 – Code Interoperability

Ada was the only language that allowed calling a routine in a separate executable (C# only allowed DLLs to be referenced). Ada has real enumeration types, which are compiled-time enforced for case statements. Because Ada offers real enumeration types, it made it easy to enumerate all of the tokens for the simplified RAPTOR language and catch missing tokens in each case statement at compile time. These strengths persuaded us to use Ada for all of the lexer, parser, and interpreter components of RAPTOR.

The primary weakness of Ada for this project was in the graphical user interface (GUI) design. The only .NET GUI builder tool for Ada, RAPID, is not full-featured and lacks many of the convenient components that are needed for RAPTOR. For example, RAPID does not include a complete set of design widgets, such as tree views and paneling that make the RAPTOR interface easier to use for students. Other features, such as panel resizing would have been very hard to do with current Ada GUI tools. Although this is not specifically a lacking in the design of the Ada language, language decisions must be made for projects based on not only the features of the language, but also available libraries and tools [8].

Another weakness of Ada was its awkward syntax for creating an object that implements an interface. This is particular was noticeable when attempting to create a child class of a .NET class. Each interface that the parent implements must be specified as a discriminant of the child tagged record. Other issues, such as type circularities, and mixing .NET and Ada strings, are handled more cleanly. In these cases, A# has inherited the design decisions from JGNAT [1,7].

A# does support the `object.method` syntax common to other object-oriented languages [1]. The authors strongly hope this proposal makes it into Ada 2005. Even though it is a syntactic sugar, it makes it much easier to call methods without requiring either long package names, or use clauses.

All Ada programming for RAPTOR was done using AdaGIDE. AdaGIDE has a simple target button for selecting a .NET target which makes it simple to create a DLL (instructions on the A# web site detail how to create a DLL for the .NET environment). AdaGIDE does not currently support autocompletion in the A#

`object.method` syntax. In addition, the A# compiler is somewhat immature and does not cover the complete Ada language (e.g. representation clauses, controlled objects as components of other objects). While some of these limitations come from the A# implementation, many are inherited from its ancestor, JGNAT [7].

3.3.2 C# in .NET

C# is an excellent language for program development under .NET because it is robust, versatile, and well-designed [3]. Currently, it is the language most often used to develop for .NET. Although it is a relatively new language, its designers used lessons learned from previous languages such as Java, C++, and others, to build a simple, safe, object-oriented language that is ideal for .NET programming [3].

C# works well under Visual Studio .NET because it is so well integrated into the development environment. One strength of C# was the extensive GUI building capability available in Visual Studio .NET. Another strength of using C# in Visual Studio .NET for the RAPTOR project was that it provided autocompletion for both .NET libraries and also Ada methods in an AdaGIDE-produced DLL. This made it easy to "see inside" various DLLs to integrate their functionality into the rest of the project.

The major downside to C# was that it was very difficult to change a design decision in Visual Studio after making it. For example, it was extremely tedious to add a panel after the fact to contain a set of already existing GUI objects. It was also cumbersome to make what should be easy changes to the project and its files (e.g. change the name of the executable). This is a tool issue rather than a language issue, but as it is unlikely for a project to use C# without Visual Studio .NET, it is worth noting.

3.3.3 C++ in .NET

In order to integrate the ability to use graphics within RAPTOR, we decided to use a legacy COM object that provided the graphics routines. This COM object was written in C++ and compiled under Visual Studio 7.0. The advantage to using this COM object is that it allowed direct use of the Win32 API.

Working with C++ and COM objects in general presented several problems. First, the code is hard to read. In addition, COM has a steep learning curve and is difficult to integrate under .NET.

Passing strings to a COM object is awkward because it forced a fixed maximum size. It was difficult to debug this as originally passing the `char *` worked sometimes and sometimes not depending on whether the system decided to marshal the arguments. This was resolved by using fixed length strings which has obvious disadvantages.

3.4 Mixed-Language Issues

Adding the Ada DLL to the Visual Studio .NET project was very simple, involving only adding a reference to the DLL. There were two other issues that needed to be addressed.

First, A# generates elaboration code for packages. If the main program is also compiled with A#, then calls to these routines are automatically inserted. Since the main program in RAPTOR was written in C#, it was necessary to explicitly call the Ada initialization routine, adainit.

Second, the default string type in C# more closely resembles the Ada data type Unbounded_String (from the package Ada.Strings.Unbounded) instead of the Ada string type. When the Ada DLL returned a string to the C# code, it was necessary to add an explicit conversion. A# provides a convenient "+" operator to convert back and forth between Ada and .NET strings.

4. CONCLUSION

This paper described our experiences in using Ada with other programming languages in the .NET environment and explained our approach in using multiple languages in the development of a real-world software project. Through its .NET environment, Microsoft provides language independent development coupled with the future promise of platform independent execution.

The main advantage of using the .NET environment for this project was the ability to easily combine the strengths of multiple programming languages and leverage legacy code. Also, the availability of a .NET runtime for Linux provides some amount of immediate platform independence.

Ada fit well into this environment of language independence, and allowed us to gain the advantages of strong-typing for portions of our project, while also leveraging the extensive GUI capabilities of Visual Studio .NET. If broader tool support for Ada in the .NET Framework becomes available, and certain OO extensions (resolving circular types, supporting interfaces and object.method notation) make it into the Ada 2005 standard, then Ada will be a strong contender for projects wishing to use the .NET Framework.

5. REFERENCES

[1] Carlisle, Martin C., Sward, Ricky E., Humphries, Jeffrey W., *Weaving Ada 95 into the .Net Environment*, SIGAda 2002, December 8-12, 2002, Houston, TX.

[2] Carlisle, Martin, et al, AdaGIDE. See http://www.usafa.af.mil/dfcs/bios/mcc_html/adagide.html.

[3] Liberty, Jesse, *Programming C#, 2nd Edition*, O'Reilly & Associates, 2002.

[4] Platt, David S. *Introducing Microsoft .NET*, c2001, Microsoft Press, Redmond, WA 98052-6399.

[5] The .NET Runtime Environment. See http://www.microsoft.com/net/

[6] Weiss, Aaron, *Microsoft's .NET: Platform in the Clouds*, ACM White Paper, Dec 2001.

[7] Comar, Cyrille, Gary Dismukes, and Franco Gasperoni, *Targeting GNAT to the Java Virtual Machine*, Proceedings of the Tri-Ada 97 Conference, St Louis MO, Nov 9-13, 1997, pp. 149-161.

[8] Lawlis, Patricia, *Is the Answer Always Ada?*, Proceedings of the Tri-Ada 97 Conference, St Louis MO, Nov 9-13, 1997, pp. 297-301