Experience with integrating Java with new technologies: C#, XML and web services

Judith Bishop[1], R Nigel Horspool[2] Basil Worrall[1]
[1]Computer Science Department, of Pretoria 0002, South Africa jbishop,bworrall@cs.up.ac.za
[2]Computer Science Department, University of Victoria, Victoria BC, Canada V8W 3P6, nigelh@uvic.ca

ABSTRACT

Java programmers cannot but be aware of the new initiative from Microsoft of a complete language, C#, network environment, .NET, and host of supporting technologies such as web services. Before moving all development to a Microsoft environment, programmers will want to know what are the additional advantages of C# as a language over Java, and whether the new and interesting features of C# and .NET be incorporated into existing Java software. This paper surveys the advantages of C# and then presents experience with connecting it to Java in a variety of ways. The first is by providing a common XML-based class for the development of programmer controlled GUIs, which can run independently of the expensive (resource-intensive) Visual Studio development environment that would otherwise be needed for using C#. The second provides evidence that C# can be linked to Java at the source code level, albeit through C++ wrappers. The third is a means for retaining the useful applet feature of Java in a server-side architecture of .NET's web services.

The paper concludes that there are many common shared technologies that bring Java and C# close together, and that innovative ways of using others can open up opportunities not hitherto imagined.

Keywords:

Java, C#, XML, JNI, UIML, SOAP, web services.

1. INTRODUCTION

Java has been with us for seven years now and has made phenomenal inroads into the world of system, business, internet and educational programming. As witnessed by conferences such as JavaGrande, its influence extends also into scientific and high performance computing, specifically in parallel and distributed applications [5]. The reason for Java being used by these latter communities is that it has *something to offer* over and above the languages currently in use – chiefly Fortran, Visual Basic and C/C++.

Specifically, object-oriented programming, increased security both within a program and between programs, parallelism facilities, applets and access to new resources through class libraries are cited as features which could be profitably used by scientific programmers [12].

The move towards Java in distributed computing has not been without its problems [10] [9], however, and it is to be expected that programmers will be loathe to embark upon another change of language so soon. Yet, the advent of Microsoft's new language C# cannot go unnoticed, and the questions to be asked are:

- What are the additional advantages of C# as a language over Java?
- Can the new and interesting features of C# be incorporated into existing Java software?
- What is the performance profile of C#?

Like Java, C# is not just a programming language, but co-exists with a particular runtime enironment (like Java's JVM), a means of communicating on the network (like Java's RMI but unlike Java's applets) and several independent technologies which are used by both languages (such as XML).

The purpose of this paper is to present experience with C# co-existing with Java in numerous ways, and to indicate which avenues of approach are likely to be fruitful in the near and medium future. The paper serves as a survey of possibilities, some of which are written up in more depth in other work [6], [14].

Whereas Lobosco *et al* [9] survey some fourteen specialised projects for adapting Java for high-performance computing, we concentrate on exploiting freely available (if not always free) application independent technologies in this area.

The paper is organised as follows: Section 2 gives a brief overview of C# from the Java programmer's viewpoint. Section 3 takes three of the main technologies for C# to Java interoperatbility, namely XML, JNI and .Web services and shows how they can be used to enhance Java programs. Section 4 outlines continuing and future work; Section 5 gives our conclusions.

2. ADVANTAGES OF C# FOR JAVA PROGRAMMERS

C# is an object-oriented language developed by Microsoft to act as a worthy successor to C++ and Visual Basic. Like its close cousin, Java, it runs within a specialized environment, but a notable difference is that C# code is not interpreted. It is compiled first to an intermediate code (CLI) and then before execution, it is JIT compiled to native code. Thus C# only ever runs in native code, which is a plus for high performance computing.

Much of C# is the same as Java. At the language level, its object-oriented model is very similar in that it defines classes and interfaces, does not have multiple inheritance, and relies on garbage collection at runtime (known as 'managed code').

C# has several new features which make it interesting for Java programmers. Those which we have identified (not an exhaustive list) are:

1. Better control of synchronized objects with the volatile modifier.
2. Operator overloading as in C++ is allowed.
3. The switch statement can switch on strings (see section 3.1 for an excellent example of this feature).
4. The indexing operator [] is overloaded for all collections so that one can move seamlessly between arrays and more complex data structures such as hash tables as the need arises.
5. Input-output and file input-output is simpler than in Java, and also more powerful, with the introduction of format statements.
6. Objects can be serialized in binary and in XML. The XML format would typically be used across the network or between programs.
7. there is a Dispose method for deterministic control of releasing resources held by classes.
8. Structs (light weight objects) are available. They are allocated on the stack and are not subject to garbage collection.
9. Values are automatically converted to objects and back as required.
10. The foreach statement gives increased power to iteration over collections of objects (similar, but much simpler than, the iterators in Java).
11. Properties are available for all data members of a class, giving a much cleaner and neater syntax for the definition of get and set.
12. Multidimensional arrays represent a contiguous block of memory and are therefore susceptible to the compiler loop optimizations expected for high performance computing. These are distinct from the normal jagged arrays of multiple objects which C# (and Java) has as default.
13. Verbatim strings avoid the use of escape characters, and allow multi-line strings, a feature which was essential for the development of the XML based GUI class described in section 3.1
14. Overflow can be detected or ignored in expressions and type conversions.
15. Delegates provide callback functions, akin to functors in C++.

On the debit side, C# does not have inner classes, dynamic class loading, strictfp (for enforcing IEEE 754 floating point) and of course cross-platform runnability. At present, C# is available only on Windows at industrial strength, although a FreeBSD Unix version exists for research purposes [13].

We have been programming small and large programs in C# for six months and have found it reliable, fast and easy to use. For a Java programmer, it isa remarkably familiar language base. What is different is its supporting technologies, and of course all of its APIs (called anmespaces). In what follows, we look at the some of the major technologies that enable Java and C# to co-exist.

At this stage, there are several professional books on C#, such as [16] and also several independent online resources such as [3]. In addition, there are some independent comparisons of C# and Java, the best being those by [11] and [8]. To date there has not been a similar comparison of supporting technologies, but [14] aims to fill that gap.

3. SUPPORTING TECHNOLOGIES FOR JAVA AND C#

There is an almost overwhelming range of supporting technologies for Java, and an equal number for C#. Some of them, such as http, XML and SOAP, are independent and shared by both languages. Others are specific to Java or to C# and do not have counterparts in the other camp, such as JavaBeans on the one hand and Web Forms on the other. It is clearly important to know which technologies are shared and which are not, and it is not always easy to make the distinction, since Sun and Microsoft, being vendors, have a vested interest in presenting technology in a propriety manner. For example, Sun calls its XML messaging capability JAXM and Microsoft's distributed and reusable components are called ActiveX controls.

A programmer trained in Java, say two or three years ago, is faced with two challenges:

- sifting out the new technologies that are relevant, and would enhance productivity
- learning how to integrate these into existing software successfully.

We shall go through three high-profile technologies, and show how they can be used to great effect to bring C# into Java programs. These are XML, JNI and .NET Web Services.

3.1 Incorporating XML into Java and C#

XML is a universal format for data that provides structure to information so that it can be easily parsed [4]. Scientific programmers often have complex data sets to input or transfer. By using XML to describe the structure of the data, integrity can be ensured across programs, and between runs of a program. XML can be enhanced by XML-schemas that can specify meta-structure, and by XSL, which describes stylesheets, used for the presentation of a class of XML documents.

We illustrate the power and ease of use of XML by describing a tool which considerably eases the burden of writing visually appealing user-interfaces in both C# and Java. The tool is called XGui [6] [7]. It was originally written in C#, but can be called from Java, as described in the next section. It has also been recently successfully translated into Java.

C# has two antecedents – the Windows world where people are used to building programs with point and click forms, and Java, where the awt and Swing class libraries make hand-written GUI code exceptionally easy. Emulating Java's ease of GUI creation in C# without the support of Visual Studio .NET is quite a challenge.

Approaches to creating C# GUIs

We have investigated four options for C# GUIs:

Option 1: Use Visual Studio as is. Creating a GUI using Visual Studio is easy to do, especially for those who have been used to this paradigm. Components (or controls) are clicked onto the screen in the correct position, and then properties are filled in to connect the actions associated with them to handler code. However, the code generated by the environment to implement the form is intimidatingly long and verbose. Visual Studio is also a large system, not suitable for running on all computers. Moreover, the open source version of C# that runs on Free BSD Unix obviously does not have this environmental support [13].

Option 2: Use Visual Studio but create a strict calling interface to user code. The idea here would be to have the programmer write the handlers, rather than use the property sections to do so. The problem is that it is not possible to separate out dumped code from programmer written code in any viable form without editing the dumped code, which spoils the purpose. The dumped code is lengthy and overwhelming. If it is hidden using Visual Studio's region feature then key features are also hidden, such as the names of the components, text that is displayed and so on. Such code is all mixed up with non-key information such as the pixel position of the component, the text alignment and the tab index.

Option 3: Create the form without Visual Studio. Avoiding the environment helps very little. We programmed a simple "ButtonTest" example through Visual Studio first and it came to some two pages of dumped code and 10 lines of an `actionPerformed` event. The length of the dumped code is due to Windows Forms controls only having one, no-parameter, constructor, and there being no flow managers. So even a simple instantiation of `Button` has to be followed by several (usually six) assignment statements. For example:

```
this.waitButton.BackColor =
  System.Drawing.SystemColors.Control;
this.waitButton.Location =
  new System.Drawing.Point(7, 128);
this.waitButton.Name = "waitButton";
this.waitButton.TabIndex = 2;
this.waitButton.Text = "Wait";
this.waitButton.Click +=
  new System.EventHandler(
  this.actionPerformed);
```

We then tried to massage the code down into its bare bones as it would be written by a human, but we could not achieve the desired effect. Windows Forms are clearly not meant to be programmed by hand.

Option 4: Design a customised class. The final possibility is to hide the complexity in a special multi-

purpose class which uses ordinary method calls to operate on a reasonably complex GUI. The setting up of the GUI could be done in one of two ways:

- Have a small set of methods to call to create simple buttons and textboxes
- Use an XML description of the form to drive the setup.

The first alternative was used with the Display class discussed in [1]. We support the second alternative because it is modern and far more extensible. It is analogous to the embedding of SQL in JDBC calls. All of the above programming shown in Option 2 would be replaced by the XML specification

```
<button text='Wait'/>
```

which would be used by the constructor of the GUI class to create a button control.

The idea of using XML notation to specify the layout of a GUI is not new; the User Interface Markup Language (UIML) is an XML tagging scheme which was invented for exactly that purpose [2]. However, UIML is a complicated tagging scheme, while our XML tags are greatly simplified by making them have the same names and attributes as the Winforms controls that they generate. The purpose of UIML is to be platform independent, while XGui is very much oriented towards Windows.

A simple GUI class for C#

Each instance of the XGui class creates a single Windows *form* (called a Frame in Java terminology) on the screen. Each form contains some simple *controls* (components in Java), such as labels, buttons and text boxes that the calling program can use for GUI input and output.

The controls displayed on the form and their layout are specified by an XML string passed to the constructor. The user can subsequently set and get information that appears within the controls, e.g., the user can call the `GetText` method to obtain the text available inside a textbox control. Figure 1 shows an XGui Windows form for a simple scientific program to calculate capacitance charge using Simpson's Rule.

Figure 2 shows the XML specification needed to define that form. The <horizontal> and <vertical> XML tags simply group controls in horizontal or vertical lists. Otherwise, we have singleton tags for each control that is supported.

The XML notation is infinitely extensible. We can add attributes to specify the size of a control, its colour, its alignment relative to other controls, its position in the form, and so on.

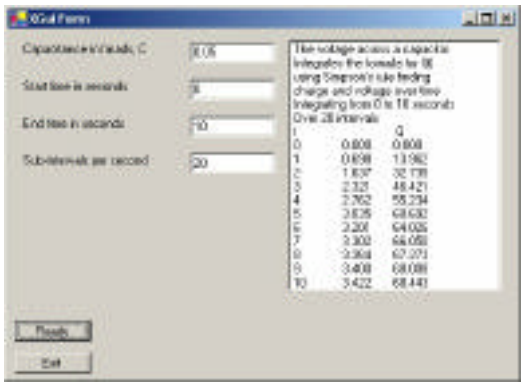

Figure 1 A user interface created by XGui

```
XGui xgui = new XGui(@"<gui name
     ='Capacitance Charge'>
  <vertical>
    <horizontal>
      <vertical>
        <label text =
          'Capacitance in farads, C'
           width=150/>
        <label text = 'Start
          time in seconds' width=150/>
        <label text =
       'End time in seconds'
       width=150/>
        <label text =
       'Sub-intervals per second'
       width=150/>
       </vertical>
       <vertical>
       <textbox name='capacitor'
          text='0.05' width=80/>
       <textbox name='starttime'
          text='0' width=80/>
       <textbox name='endtime'
          text='10' width=80/>
       <textbox name='intervals'
          text='20' width=80/>
      </vertical>
      <vertical>
      <listbox name = 'list' width=200/>
      </vertical>
    </horizontal>
    <button name='Ready'/>
    <button name='Exit'/>
  </vertical>
</gui>");
```

Figure 2. XML specification for a GUI

Where there is no confusion, tag names are the same as the names of the corresponding components in the C# Windows Forms class library and the attribute names match the names of members of those classes. Case is ignored for the tag names and attribute names.

Everything has a sensible default. Layout and sizing are automatic unless overridden by explicit attributes.

The XGui object created by the program maintains the controls specified in the XML. To interact with them, we use a series of methods, such as:

`string GetButton()` – returns the id of the button that was pushed; by default, the id is the same as the text the user sees displayed on the button.

`string GetText(string id)` – returns the text that is currently contained in the textbox whose id is specified.

`void PutText(string id, string text)` – writes the string text into the textbox or the scrollable text window whose id is specified.

There are several others which control the colour of the individual controls. Figure 3 shows some sample C# code that interacts with the controls in the form. There is just one constructor:

`XGui(string xmlSpec)` – uses the XML string to construct the layout.

Controls implemented in the XGui class include: Label, Button, TextBox, and ListBox, OpenFileDialog and SaveFileDialog.

```
while(xgui != null) {
  string b = xgui.GetButton();
  switch (b) {
    case "Ready":
      C = double.Parse(xgui.
        GetText(capacitor"));
      // … and other values
      if (intervals%2==1) intervals++;
      xgui.PutText("list",
        "Integrating from "
        +start+" to "+end+" seconds");
      // … and other values
      // Perform integration
      break;
    case "Exit":
      xgui.CloseGUI();
      xgui = null;
      break;
    }
  }
```

Figure 3 The C# Handler for the XGui object

Assessment

The XML notation is very powerful and extensible, and is revolutionising the way we write programs. For example, supposing we wished to add a button to cause the program to draw a graph of the process. All we need to add is

```
<button name = 'Draw Graph'/>
```

wherever we would like it to appear in the GUI, and then add the appropriate case to the switch statement (C#) or equivalent if-else statement (Java). While XML has been used in many contexts, we do not know of any attempts to integrate it as thoroughly into coding as we have done. It should be noted that the implementation of XGui (described in [7]) is non-trivial. Its current implementation uses regular expressions to normalize the XML notation and reflection to make access to controls truly dynamic.

3.2 Using JNI to integrate Java and C#

Instead of retooling a system in C# and retraining programmers to do so, let us just ensure that we can call C# from Java. Then, as and when needed, new parts of a system can be written in the new language. From Java's point of view, C# should be Native Code in the same way as C or C++ is. If so, then it can be called from Java using JNI, the Java Native Interface, which comes standard with the JDK. JNI supports the portability of code across all platforms and allows code written in C/C++/Assembler to run alongside the JVM (Java Virtual Machine). The interface works at the level of method calls, so that native methods can create, update and inspect Java objects and call their methods.

Summary of JNI for C

The steps to use a C function via JNI are described in [15], and involve the following steps. The Java side includes a method stub along the lines of

```
public native void displayHelloWorld();
```

and then uses the javah tool to generate a C header file, which includes the declaration:

```
JNIEXPORT void JNICALL
   Java_HelloWorld_displayHelloWorld(
   JNIEnv *, jobject);
```

The method we started out with, displayHelloWorld, is now prefixed with Java and the name of the class, making it `Java_HelloWorld_displayHelloWorld`. We implement this method in C, say, as follows:

```
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>
JNIEXPORT void JNICALL
   Java_HelloWorld_displayHelloWorld(
     JNIEnv*env,jobject obj) {
       printf("Hello world!\n");
     return;
}
```

Then the Java to activate everything is:

```
class HelloWorld {
  public native void displayHelloWorld();
  static {
    System.loadLibrary ("HelloWorld");
  }
  public static void main(String[]args) {
     new HelloWorld().displayHelloWorld();
  }
}
```

When the `HelloWorld` class is instantiated, it starts up JNI and the C method, by loading the library which includes the HellowWorld.cpp file. If all goes well, Hello world!, is printed out.

JNI for C#

Now suppose we want to implement the same procedure, but this time for C#, i.e. using the C# method:

```
using System;
public class CSharpHelloWorld {
   public CSharpHelloWorld () {}
   public void displayHelloWorld() {
     Console.WriteLine(
      "Hello, World From C#!");
   }
}
```

The difficulty is that JNI at present only works for C and C++, not yet for C#. To call a C# method, we will need to wrap it in C++. But C# is managed (i.e. garbage collected) and C++ is not. Fortunately, we can use C++ with what are called Managed Extensions, creating a managed wrapper, which interacts with an unmanaged wrapper, which interacts with Java. This is summarised in Figure 4. The managed wrapper includes a class called HellowWorldC with a function called “method” as follows:

```
public:
  HelloWorldC() {
  t = HelloWorld();
}
void method() {
  t -> displayHelloWorld();
}
```

The original C++ wrapper is then altered to interface with the Java method as:

```
#include "HelloWorld.h"
#include "HelloWorld.cpp"
#include <string.h>

JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld
  (JNIEnv *jn, jobject) {
        TestLibC* t = new
        TestLibC();
     t->method();
  }
```

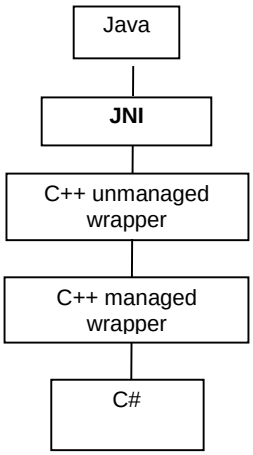

Figure 4. Using JNI for Java to C# communication

Assessment

The question is whether this technique can be used generally, and with heavy-duty C#. The answer is yes, with a bit more work. The problem is that pointers in managed C++ may change their values during a garbage collection. One could prohibit the type-casting of managed pointers to unmanaged pointers. A more satisfactory solution is to create a static reference to a managed pointer. Using this technique, we successfully called the large C# class, XGui, described above from Java. For details, see [17].

3.3 .NET WEB SERVICES AND JAVA

The .NET framework is the implementation of the standardized Common Language Infrastructure (CLI). .NET consists of many languages that are built on the implementation of the CLI, called the Common Language Runtime (CLR). Application execution is not interpreted, as in Java, but all supported languages are compiled to an intermediate representation, which is then compiled to the operation system's native code. The framework itself is the programming model of the .NET platform and is used for the construction and deployment of both local and distributed services and applications. see Figure 5.

Unlike Java, .NET relies on server side processing. There is no concept of an applet, where the runtime machine is held in the browser. Client-side applets are relatively simple, with access to data kept on the server either by sockets, JDBC or JSP and servlets. Although intrinsically more complex than the client-side program model, the .NET environment in reality simplifies the server side processing structure. .

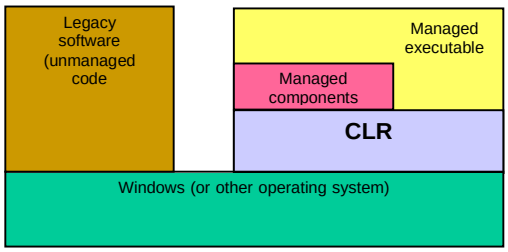

Figure 5. The .NET framework

However, there is an interesting alternative to the standard .NET approach. We can create and maintain a Java applet on the client side that is able to interact with the server as a .NET web service. A *web service* is a software component that exposes useful functionality to web users through a standard web protocol such as SOAP (Simple Object Access Protocol). This link enables a client to do a remote procedure call to all the functions of the web service that are exposed to the web via a SOAP message. The response from the web service will also be in the form of a SOAP message.

Setting up a Web Service with Microsoft Visual Studio .NET is easy. Then each method can be exposed to the web by adding a [WebMethod] tag, e.g.

```
[WebMethod]
public String testString() {
  return "Hello World!";
}
```

For the client to access this service it will have to generate a valid SOAP request and send it to the web service. SOAP requests are generated by a Visual Studio wizard and are themselves phrased in XML, some of which is:

```
<soap:Envelope xmlns:xsi
   ="http://www.w3.org/2001/
   XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/
          2001/XMLSchema"
   xmlns:soap="http://schemas.
   xmlsoap.org/soap/envelope/">
  <soap:Body>
   <testString xmlns=
   "http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Although we are once again using XML, as in 3.1, note that this XML is machine generated. The web service will in turn generate a response to the above request that will also be in the form of a SOAP message.

Accessing the web service from Java

To access a web service written in C#, we construct a SOAP message and send it to the web service. The Java XML package JAX is used for generating SOAP messages. The appropriate JAX classes for a SOAP message are:

```
SOAPMessage
  SOAPPart
    SOAPEnvelope
   SOAPHeader
   SOAPBody
  AttachmentPart
```

The AttachmentPart can contain any data that is not in XML format. After creating objects for each part of the message, parts of the body are filled in with the XML string that is relevant to our application, for instance:

```
Name bodyName =
    SE.createName("testString",
    "L","http://tempuri.org/");
SOAPBodyElement SBE =
    SB.addBodyElement(bodyName);
```

This will generate the following XML code and add it to the SOAPBody:

```
<L:testString xmlns:L=
    ”http://tempuri.org”>

</L:testString>
```

To add more XML tags to the document create SOAPElements and add them to the SOAPBodyElement. To generate a SOAP message we must get a connection to the web service and send the message. This is also done through further JAX classes:

```
SOAPConnectionFactory SCF =
    SOAPConnectionFactory.
    newInstance();
SOAPConnection con =
    SCF.createConnection();

URLEndpoint endpoint = new
  URLEndpoint("http://localhost/
    WebService1/Service1.asmx/
    testString?");
SOAPMessage response =
  con.call(message,endpoint);
```

When executing `con.call(message,endpoint)` the request will be send to the URLEndpoint and it will wait for the response from the web service and store it in a new SOAPMessage object. To use the response that is received from the web service is also very easy because we have a new SOAPMessage object, and we can access the SOAPBody as we did before.

Assessment

SOAP messaging might seem like a complex and esoteric technology, but in fact, it is quite straightforward and follows set patterns. A Java programmer should have no problem in setting up the first connection, and thereafter additional messages just require resetting the SOAPBody.

With an increasing number of web services coming on line, the ability to access them from existing applets will considerably enhance their value and prolonged their lifetime.

4. FUTURE WORK

Work continues in two areas:

- implementing more examples of the use of the technologies mentioned here, and assessing their performance;
- investigating other technologies for interaction, such as ODBC.

Within the high performance community, it is clearly important to assess the impact of layered approaches to software (as in the JNI solution). With SOAP messaging from applets, an important consideration would be the speed of graphics dispatch. The XGui class is now being ported to Rotor, so that it can be run on Free BSD Unix. Here the interactions between Java and C# can be tested all over again.

5. CONCLUSIONS

The relative merits of C# and Java will doubtlessly fuel arguments for many years. As a language, C# has corrected some deficiencies of Java and added a few new features. More importantly, C# is integrated into the .NET environment which provides access to web services and operating system services for the Windows platform. C# will join C++ and Visual Basic as a major programming language for developing Windows applications and web applications that run on Windows servers.

Java is also intended for general application development and web services development, on both the client side and server side. Java has the advantage of being platform independent, which C# may never become. However, there are certainly situations where C# will be preferable to Java. If applications are being developed for Windows, code written in C# will normally execute more efficiently, will have direct access to operating system services, and will more easily inter-operate with programs written in other languages. The ‘unmanaged code’ feature of C# allows exactly that. This paper provides an example of C# calling C++ which, in turn, calls Java. The approach is entirely object-oriented, with class instances in C# communicating with class instances in Java via method calls.

While it is not easy to combine C# and Java in the same application program, it is possible. The example application in Java invokes an instance of a C# class, XGui, for access to a GUI on Windows. There are potential difficulties working with pointers to objects, but these can easily be avoided.

The C# language provides direct access to operating system services and web system services in the Windows environment. This paper shows how Java code can be linked with C# code to obtain similar access, albeit yielding a program which is no longer platform independent.

ACKNOWLEDGEMENTS

This work was supported by NRF grant 2969. We are frateful for the technical input and advice of Johnny Lo, Cobus Smit, John Muller, Kathrin Berg, Theo Danzfuss and Theo Crous.

REFERENCES

[1] J M Bishop and N T Bishop, Object-orientation in Java for scientific programmers, Proc. 22nd International SIGCSE Technical Symposium on Computer Science, Austin, pp 205-216, March 2000.

[2] R. Cover, User Interface Markup Language, http://www.oasis-open.org/cover/uiml.html, last visited June 2002.

[3] C# Links and Resources, http://www.webreference .com /programming/csharp/, several articles on C#, last visited June 2002.

[4] Extensible Markup Language (XML), http://www.w3c.org/XML/, last visited June 2002.

[5] V Getov, G von Laszewski, M. Philippsen and I. Foster, Multiparadigm communi-cations in Java for grid computing, Communications of ACM 44 (1) 118-125, October 2001.

[6] R N Horspool and J M Bishop, XGui, http://www.cs.up.ac.za/~jbishop/rotor, Last visited June 2002.

[7] R N Horspool and J M Bishop, XGui: the design and implementation of an XML GUI generator for C# and Java, in preparation, June 2002.

[8] M Johnson, C#: a language alternative or just J--?, *JavaWorld*, November 2000, at

http://www.javaworld.com/javaworld/jw-11-2000/jw-1122-csharp1p2.html, last visited May 2002

[9] M. Lobosco, C Amorim and O Loques, Java for high performance network based computing: a survey, Concurrency and Computation - Practice and Experience, **14** (1) 1-32, January 2002.

[10] J Moreira, S Midkiff, M Gupta, P Artigas, P Wu and G Almasi, The Ninja Project, Communications of ACM 44 (1) 102-109, October 2001.

[11] D. Obasanjo, A comparison of Microsoft's C# programming language to Sun Microsystem's Java programming language, http://www.25hoursaday.com/CsharpVsJava.html, 72 pages, last visited June 2002.

[12] C M Pancake and C Lengauer, High-performance Java, Communications of ACM 44 (1) 99-101, October 2001

[13] Rotor, http://research.microsoft.com/programs/europe/rotor/default.asp, last visited June 2002

[14] C Smit and J Muller, J2EE platforms and Microsoft .NET: a techological comparison, Technical Report, Department of Computer Science, Univeristy of Pretoria, May 2002.

[15] B. Stearns, Trail: Java Native Interface, http://java.sun.com/docs/books/tutorial/native1.1/index.html, last visited June 2002.

[16] A Troelsen, C# and the .NET platform, APress, 2001.

[17] B. Worrall and J. Lo, Integrating C# and Java, http://www.cs.up.ac.za/polelo/ interests.html, last visited June 2002.