Creating Web Services for Legacy COBOL

Craig A. VanLengen
craig.vanlengen@nau.edu

John D. Haney
john.haney@nau.edu
College of Business Administration, Northern Arizona University
Flagstaff, AZ 86011-5066

Abstract

Billions of lines of COBOL code are executed on a daily basis, primarily in financial transactions. With the presence of newer development environments these programs must be rewritten or accessed in a viable way. Placing legacy programs into web services is one way to interface existing functional programs with contemporary interfaces, whether Windows or Web. This study presents how a COBOL legacy program can be placed into a web service and accessed from a COBOL client, a Windows client, and a web client. The inclusion of this type of skill into the curriculum is highly desirable in order to address the skill loss due to the retirement of COBOL programmers.

Keywords: web service, client, COBOL, .NET, legacy, Windows, web

1. INTRODUCTION

Business organizations have a large investment in software written in COBOL. Evelyn (2002) estimates the investment in COBOL "to be $3 trillion." It is estimated that 70 percent of the world's data is processed by COBOL, that nine out of 10 ATM transactions are done using COBOL with thirty billion online COBOL transactions processed daily. This volume of COBOL based software is significant considering that the programs were written 10, 20, 30, or more years ago and the programmers that wrote and maintained them are retiring (Evelyn, 2002).

The loss of experienced COBOL programmers to retirement is compounded by the fact that only a few colleges offer courses and programs with COBOL skills (Mitchell, 2006b). Mitchell (2006b) estimates that the COBOL software base will grow "3% to 5% annually through 2010," but mostly from maintenance of the COBOL code.

Since the prognosis is a significant reduction in COBOL proficient programmers, there must be alternatives to COBOL development. One alternative would be to replace existing COBOL programs with programs written in more current languages. This is not feasible in the short-term given the amount of existing COBOL code. Instead of replacing this large software base, alternatives to extending the life of the code should be considered (Coyle, 2001) (Evelyn, 2002). Provision should be provided for enhancements without touching the mostly stable COBOL code (Mitchell, 2006b). Haney (2005) presented a way to wrap the legacy COBOL code with an objected-oriented COBOL proxy that could be called from a C# program or any other programming language written in a Microsoft .NET environment. Another way is to minimally modify the legacy COBOL by making it a web service that can be called from current environments including Windows and web browser based software.

The advantage of converting the legacy COBOL to web services is that standards for web services are provided (Evelyn, 2002) which allow components to be developed that can communicate over the Internet without worrying about the operating system or programming language (Micro, 2001). Other technologies such as, Common Object Request Broker Architecture (CORBA), Enterprise Java Beans (EJB), or the Distributed Component Object Module (DCOM), are code-centric and in some cases operating system dependent (Coyle, 2001). XML-based web services allow a web service on one server to be executed from an application on the same or a different server by sending an XML message using standard HTTP (Coyle, 2001). The SOAP messages, which are XML documents, are carried as HTTP requests and responses and can be used across organization firewalls (Micro, 2001). Another XML document is the Web Service Description Language (WSDL) file. This document lays out the contract between the web service and the client, by exposing the methods as services and defining data parameters (Micro, 2001).

Converting the legacy COBOL into web services can also be part of a service-oriented architecture (SOA) strategy for an organization where the interface is exposed and new developers are insulated from the COBOL code (Mitchell, 2006a). Kanter and Muscarello (2005) conducted a study on effective ways to web-enable mission-critical legacy systems. They compared the adaptation of legacy COBOL systems using Fujitsu Software migration tool and a full rewrite of the legacy system using the Java programming language. The migration tool solution required "less than 3% of the time needed to rewrite the application in JAVA" (Kanter & Muscarello, 2005).

Figure 1 presents an overview of an example COBOL web service. The COBOL web service contains the legacy program, which performs an update of an indexed sequential master file using a sequential transaction data file. An object-oriented COBOL proxy program, within the web service, acts as an interface between the legacy program and the client program (Haney, 2005). The transference of data is through the Data Object class. The client references the web service.

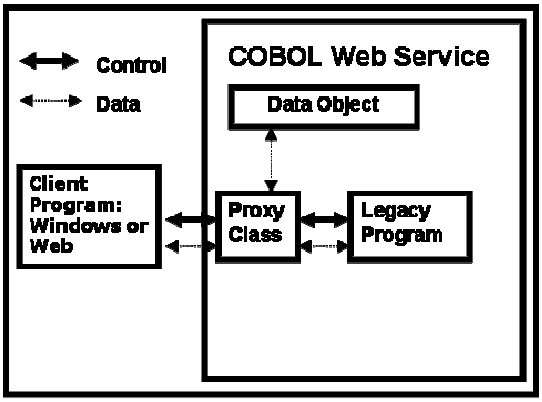

Figure 1. System Overview

This example demonstrates how to create a web service using a legacy COBOL program, and then consume the web service using a COBOL, Windows, and web application client. Modifications of the legacy program, other than changes to enable interfacing for the web service, are outside the scope of this study.

2. The Legacy Program

Some modifications must be made to the legacy program in order to interface properly within the web service. A linkage section must be added to communicate between the legacy program and the proxy class within the web service. The Procedure Division statement must be modified to reference the linage section. The Stop Run statement is replaced with an Exit Program statement. At the beginning of the program values being sent to the legacy program must be moved from the linkage section. At the end of the program logic, values being passed from the legacy program are placed into the linkage section.

```
Linkage Section.
01 lnk-FileName          Pic X(80).
01 lnk-addCount          Pic 9(6).
01 lnk-chgCount          Pic 9(6).
01 lnk-delCount          Pic 9(6).
01 lnk-txtMessage        Pic x(50).

Procedure Division using lnk-FileName
        lnk-addcount
        lnk-chgCount
        lnk-delCount
        lnk-txtMessage.
```

Figure 2. Legacy Program

3. DEVELOPMENT ENVIRONMENT

The following tools were used in this project: Microsoft Visual Studio .NET 2003, ASP.NET, IIS, and Micro Focus Net Express®. Net Express was used to develop the web service containing the legacy COBOL program and also to generate the COBOL client. The legacy COBOL code can be found in the reference for Haney (2005).

Creating the Web Service Using Micro Focus Net Express

The first step is to create a Net Express project with the legacy COBOL program, cblUpdate.cbl. Once the project is created, an interface is created using the Service Interface to map the legacy COBOL as a web service using the current project and specifying the name of the COBOL source file (cblUpdate.cbl). Using the Default Mapping option of the Interface Mapper map the COBOL data types from the legacy COBOL program into XML. Figure 3 shows that the Interface Mapper generates an input and an output for each of the data fields from the COBOL program. The COBOL data fields are shown on the left side of the dialog and the operation CBLUPDATE along with the "Interface Fields" on the right side of the dialog.

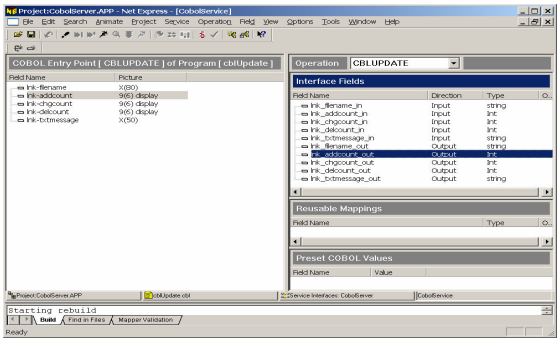

Figure 3. Interface Mapper

From the Interface Mapper it is possible to modify the default mappings. Operations can be added, changed, or deleted along with interface fields, Reusable Mappings, and Preset COBOL Values.

In this example, the name of the transaction file is sent to the web service and it returns to the client the counts of records added, changed, and deleted along with a message. The "Input" fields for addcount, chgcount, and delcount and the "Output" field for filename are deleted. Once editing of the interface is completed the service is ready for deployment.

The web service is deployed using the Micro Focus enterprise server. The first step is to configure and start the Enterprise Server Administration tool. Figure 4 illustrates that the server administration tool runs in a web browser so that the administration functions can be performed from any location of an enterprise.

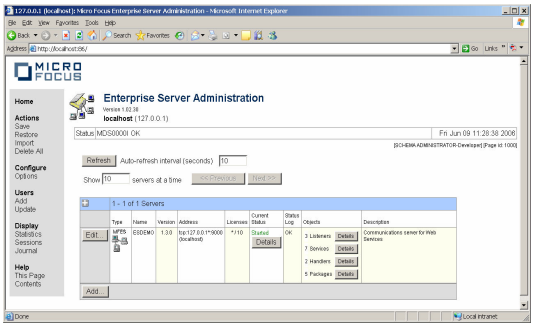

Figure 4. Enterprise Server Administration Tool

In this example the default configuration named ESDEMO is used. In Figure 4, under current status, started is indicated directly above the Details button. If the server is not currently running the button will be labeled Start instead of Details. By clicking on the "Start" button the server is started and the browser will appear similar to that shown above.

Next, the deployment settings of the service are set. This is accomplished using Net Express Service Interfaces and selecting the mapping file that is generated when performing the mapping, CobolServer.mpr. During this process the enterprise server (ESDEMO) is selected, and the enterprise server run-time environment is enabled. The application files needed for deployment are identified, along with two other files that are generated as part of the mapping, cblUpdate.idy and cblUpdate.int. cblUpdate.int is the executable file and cblUpdate.idy is required for debugging.

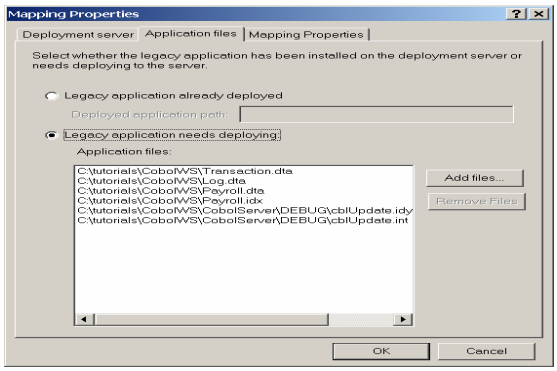

Figure 5. Deployment Settings

The web service is now ready for deployment by making sure the service, CobolServer.mpr, is selected and then clicking Service > Deploy. Several files are created during this process. CobolService.wsdl is an important file when creating clients. This file, which provides for interaction with the web service, is what defines the web service. The top part of the file is shown below. Notice the definitions of CBLUPDATEInput and CBLUPDATEOutput. These are created based on the interface mapping. The WSDL is the file that is used when the client is created.

```
 <!--
Micro Focus NetExpress 4.0 auto-generated
  WSDL document
-->
ypes>
chema elementFormDefault="qualified"
  targetName-
  space="http://tempuri.org/cblUpdateSe
  rv"
  xmlns="http://www.w3.org/2001/XML
  Schema" xmlns:SOAP-
  ENC="http://schemas.xmlsoap.org/soa
  p/encoding/"
  xmlns:tns="http://tempuri.org/cblUpdat
  eServ" />
</types>
nessage name="CBLUPDATEInput">
art name="lnk_filename_in"
  type="xs:string" />
</message>
nessage name="CBLUPDATEOutput">
```

Figure 6. CobolService.WSDL file

4. The COBOL Client

The next step creates the COBOL client. Again using Net Express, a simple COBOL client is created to interact with the web service. Net Express allows the generation of a client directly from the mapping or from the WSDL file. Either way it generates the same files. In this example the WSDL file is used.

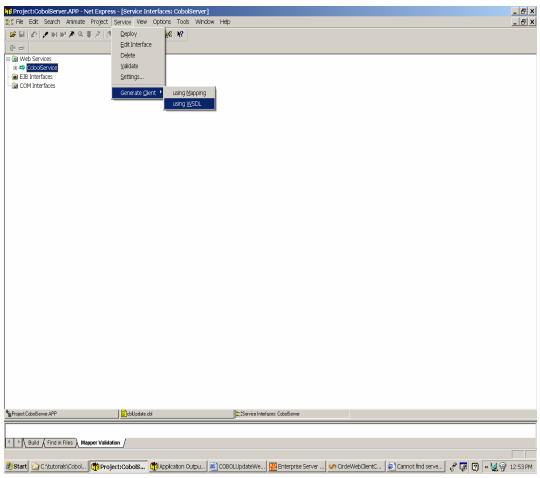

Figure 7. Create COBOL Client

A wizard window opens where the WSDL file is selected from the web service deployment. Net Express generates a simple client interface program and executes it as shown below.

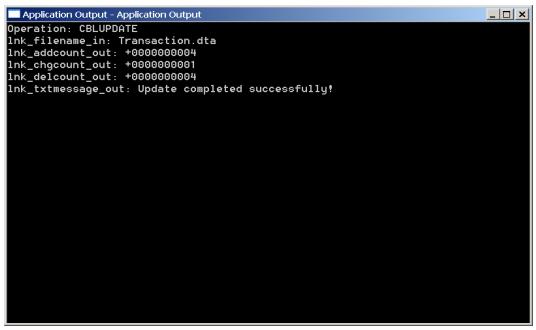

Figure 8. Client generated using Net Express and executed with the Micro Focus Enterprise Server

5. The Windows Client written in C#

Next, using Visual Studio .NET, ASP.NET, and IIS a Windows client written in C# is created. Also a web browser based client written in C# can be written. The IIS administration tool is used to create a virtual directory for the web server pointing to the physical directory where the .wsdl file for the web service was located. In this way

the .wsdl file can be located from within Visual Studio .NET.

Using Visual Studio .NET a new project named COBOLWSClient is created as a Visual C# Windows application. The user interface allows the user to click a button to display the basic Windows file dialog to select the transaction file. Once the transaction file is selected the "Update" button to call the legacy web service is clicked. This calls the web service and passes the name of the transaction file to the web service. After the update is competed, counts for records added, changed, and deleted along with a completion message are returned from the web service to the client program.

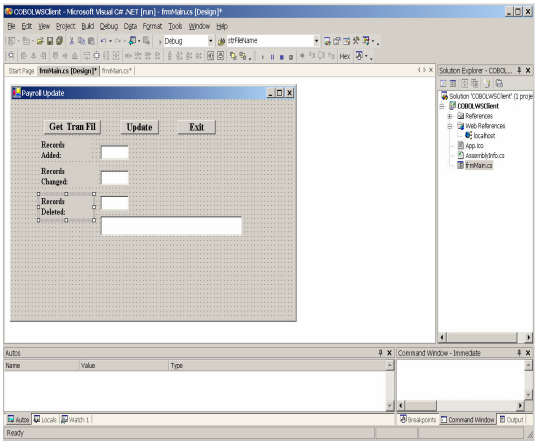

Figure 9. Visual Studio Dialog with the Windows UI

Before completing the code, to call the web service, a web reference must be added. This is accomplished by right-clicking on the project name in the Solution explorer, and then clicking on the Add Web Reference option. This brings up the following dialog showing the available web services. In this example, the Web Services on the Local Machine is selected.

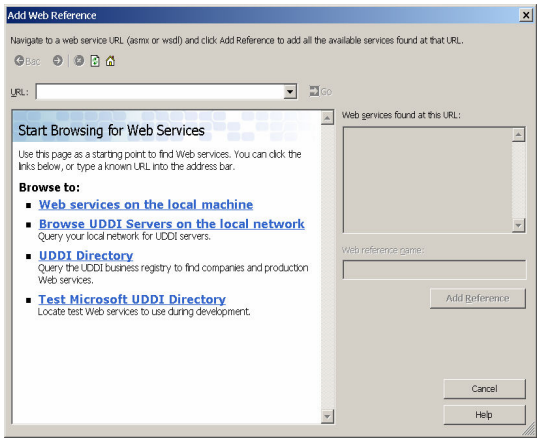

Figure 10. Add Web Reference Dialog

From the list of services presented the web service is selected, which brings up the following dialog. Notice the URL is for the .wsdl file. The web service is added into the project by clicking on the Add Reference button.

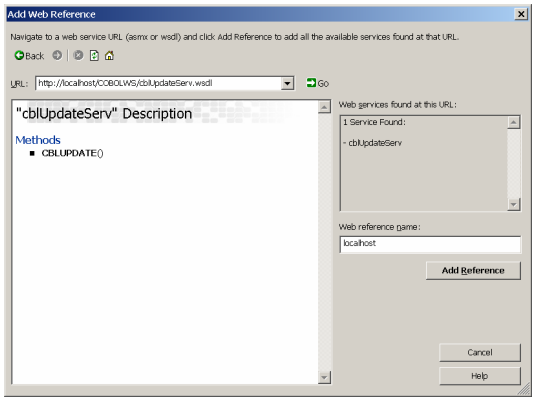

Figure 11. Add Web Reference Showing the Web Service

With the web reference included in the Visual Studio .NET project the IntelliSense feature of Visual Studio .NET provides access to the properties, methods, and data parameters of the web service class, to assist in writing the code. Figure 12 illustrates the code that executes when the user clicks on the Update button. cblUpdateServ is the name of the Web service and CBLUPDATE is the method called. The file name is passed from the client to the web service and the client receives back the counts for added, changed, and deleted along with an update message.

```
private void
btnUpdate_Click(object sender,
System.EventArgs e)
{
COBOLWSClient.localhost.cblUpdat
eServ myUpdate = new
COBOLWSClient.localhost.cblUpdat
eServ();
updateAdded =
Convert.ToInt32(myUpdate.CBLUP
DATE(strFileName, out
updateChanged, out
updateDeleted, out retMessage));
                    txtAdded.Text =
Convert.ToString(updateAdded);
                    txtChanged.Text
=
Convert.ToString(updateChanged)
;
                    txtDeleted.Text
=
Convert.ToString(updateDeleted);
                    txtMessage.Text
= Convert.ToString(retMessage);
                    btnUpdate.Enabl
ed = false;
}
```

Figure 12. C# Client Code

To execute and test the Windows C# client, the Enterprise Server must be running. If the server is not running it can be started as shown under the COBOL client example shown previously.

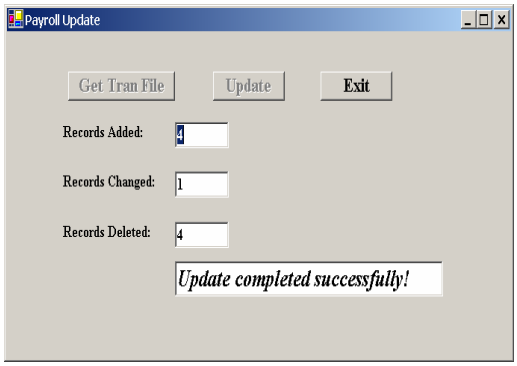

Figure 13. Results from Execution of the Windows Client

6. The Web Client written in C#

For developing a client for a web application a new Visual C# project is created using the ASP.NET Web Application Template. In this example the project is named WebClientCBLUpdate. In this example the default WebForm1.aspx page is used for the client. Normally this page would be renamed or removed and a new page added with a more meaningful name. A user interface is created that is similar to the Windows interface presented previously. Modifications must be made for the selection of the transaction file since the file commands for Windows are different from those used in the web environment. A web reference is added the same way as for the Windows client. The C# code that is executed when the "Update" button is clicked is basically the same as that used for the Windows client.

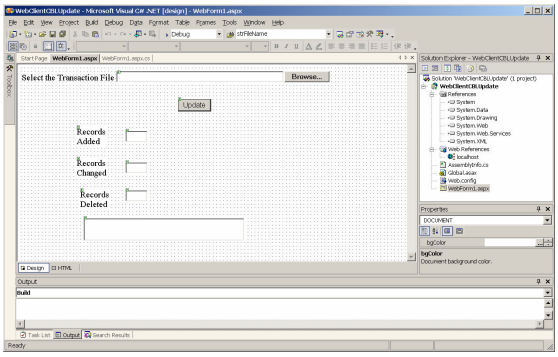

Figure 14. Visual Studio showing the Web Application Client

As in the case of the Windows Client, prior to executing and testing the web client, it must be verified that the Micro Focus Enterprise Server is running.

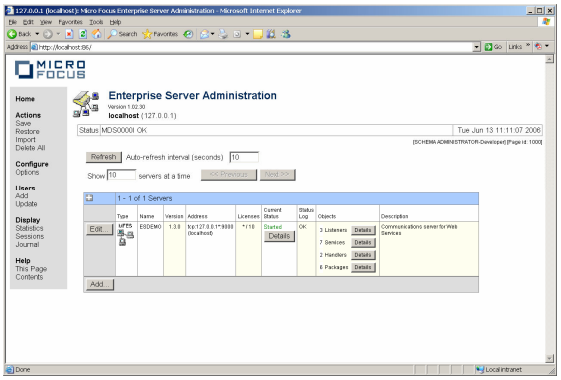

Figure 15. Verification that the Enterprise Server is running.

The web project is built and the web application is then viewed in a web browser. This is accomplished by clicking on the "Browse" button to locate the file. This process utilized the file dialog box.

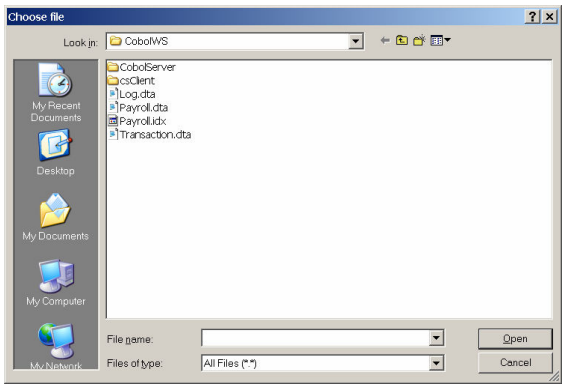

Figure 16. File Dialog for selecting the transaction file

Once the file is selected and the "Open" button is selected the window shown in Figure 17 appears.

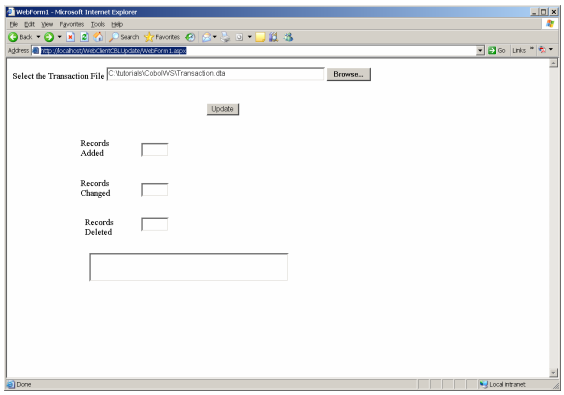

Figure 17. Web Browser Interface showing the Update Interface

Clicking on the "Update" button executes the Client program and produces the following results in the web browser.

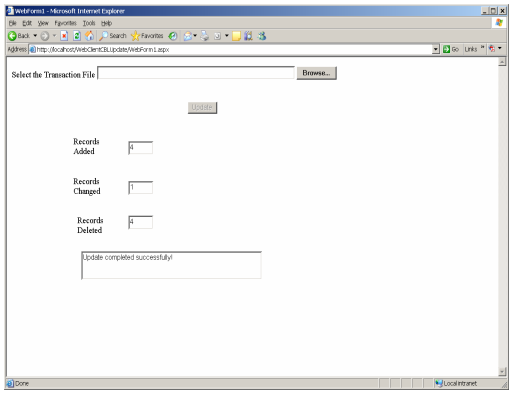

Figure 18. Web Browser Interface showing the Results

This example has demonstrated the process of creating a web service written in COBOL for the .NET environment. The functionality of the legacy COBOL program, within the web service, updates an Indexed Sequential master file. The consumer of the web service is a client written in COBOL, and replicated in both a Windows and a web application. Modification of the functionality of the legacy program is not addressed in this study.

7. CONCLUSION

By creating web services, organizations will be able to extend the life of their COBOL software investment while providing a contemporary interface for the legacy software. The "exposed" web services will then be available to be "consumed" by various application clients.

The billions of lines of COBOL code that exist in these legacy programs, primarily in financial institutions, will remain for some time. However, the retirement of COBOL programmers will generate a skill vacuum of legacy COBOL programmers. This study has addressed this concern in one area, that of placing legacy COBOL programs into web services. The ongoing maintenance is another concern that this study did not address specifically. However, the placement of legacy programs into web services has an implied reference to the need for current functionality of the legacy programs.

The combination of skills to maintain legacy programs, develop web services from COBOL programs, and develop Client programs to use the web services is necessary. This study has focused on one aspect of the overall skill package. A focus on the development of building web services and the client programs that reference them can be part of the curriculum. This can be accomplished by either a required or elective course that also provides some exposure to the COBOL code that exists in the legacy programs.

8. REFERENCES

Coyle, F. P. (2001) "Breathing Life into Legacy," September/October 2001, retrieved April 25, 2006, http://www.cobolportal.com/resources/articles/20010910_001.asp?bhcp=1.

Evelyn, R. (2002) "COBOL's Revenge: When Programs Outlive the Programmers," retrieved May 19, 2006, http://www.devx.com/devx/editorial/16357 .

Haney, J. D. (2005) "Running Legacy COBOL Programs by Proxy with COBOL.NET." Information Systems Education Journal. Volume 4, Number 28. July 2006. ISSN: 1545-679x. http://isedj.org/4/28/index.html

Kanter, H. A. & Muscarello, T. J. (2005) "Reuse versus Rewrite: An Empirical Study of Alternative Software Development Methods for Web-enabling Mission-critical COBOL/CICS Legacy Applications," retrieved May 30, 2006, http://www.adtools.com/info/whitepaper/Reuse-vs-Rewrite_final.pdf.

Micro Focus (2003) "MICRO FOCUS NET EXPRESS® GETTING STARTED," Issue 1, October 2003.

Micro Focus (2001) "Web Services and Micro Focus COBOL," retrieved April 25, 2006, http://www.microfocus.com/files/whitepapers/webservices2.pdf.

Mitchell, R. L. (2006a) "Rebuilding the legacy – modernizing mainframe code," retrieved April 25, 2006, http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=110717.

Mitchell, R. L. (2006b) "The Cobol brain drain," retrieved April 25, 2006, http://www.computerworld.com/softwaretopics/software/story/0,10801,110716,00.html?source=NLT_APP&nid=110716.