A Comparative Analysis of Programming Languages for GIS

Kurt Swendson
Department of Resource Analysis, Saint Mary's University of Minnesota, Minneapolis, MN 55404

Keywords: GIS, ArcMap, ArcView, ArcObjects, C#, Python, VB Script, VBA

Abstract

Many GIS departments in organizations throughout the world have developed customized tools using an Environmental Systems Research Institute (ESRI) proprietary language named Avenue. Avenue is reaching the end of its life cycle, and will soon be unsupported. ESRI has moved on to new versions of its software, which is not backward compatible with Avenue. This project explores the various options available to GIS professionals in order to bring their customized tools up to date using the latest software.

Introduction

GIS professionals may rewrite their customized GIS applications written in Avenue using many different languages and methods. Which of these is the best, in terms of development speed, execution speed, maintainability, and future scalability/ability to upgrade?

Although the GIS sector is made up of highly skilled and trained professionals, the subset of these people who also have computer programming skills is much smaller, and thus more expensive (Marble, 2005).

Careful planning and selection of a migration strategy is valuable in that it can save countless hours of research and development, and additionally save future development hours for inevitable further upgrades and migrations when Environmental Systems Research Institute (ESRI) and/or the rest of the computer industry moves to yet another software package.

There are countless computer languages and operating systems available in the market today, but it is safe to assume the ESRI products will run on Microsoft Windows, using programming languages available for this operating system. Although ESRI does support Unix operating systems, this study will focus on software running on a Microsoft platform.

The ESRI website shows the lifecycle chart for the various ArcView versions (Product Lifecycle Support Policy, n.d.). All versions but the latest version – ArcView 3.3, have been retired, and ESRI no longer supports them. ArcView 3.3 has been moved from general availability to extended support to mature support, which is still supported, but at a cost (Product Lifecycle Support Policy: ArcGIS 3.3, n.d.).

ESRI's new object oriented software, running under ArcGIS 9.x, is a COM-based model named ArcObjects.

Swendson, Kurt. 2007. A Comparative Analysis of Programming Languages in GIS. Volume 9, Papers in Resource Analysis. 13 pp. Saint Mary's University of Minnesota Central Services Press. Winona, MN. Retrieved (date) from http://www.gis.smumn.edu/

(Welcome to ArcObjects Online, n.d.). Therefore, languages that support COM are the only valid languages to review. Of those languages, the following will be studied: VBA (Visual Basic for Applications), VB Script, Python Script, Microsoft Visual Basic, Microsoft Visual C++, and C#.NET. This study will choose an example ArcView tool written in Avenue, the programming language used in ArcView, and duplicate all or part of its functionality using all of the above technologies, comparing and contrasting the different languages with regards to ease of development, ease of use, speed of execution, reliability, supportability, and integration into ArcToolbox and Model Builder.

Methods

Description of Original Tool

With cooperation of the Minnesota DNR, a sample Avenue application was provided. It performs a number of useful GIS functions of varying complexity. The tool, as stated in the user manual: "The Sampling Tool was designed to assist biologists in using ArcView to generate spatially explicit random or systematic sampling schemes to support resource monitoring, mapping, and research needs. The tool works either with polygons in a theme or with graphics that have been added to a view. Samples can be entirely within a single polygon or shape, or distributed among several disjoint polygons or shapes. A number of user defined constraints and settings are offered as input options" (Minnesota DNR, 2005).

The requirements of this project were to select a sample of existing tools from the given ArcView 3.3 application, implement the same functionality in different languages, and test for speed, reliability, integration into ArcToolbox, and ease of use.

From these requirements, the following course of action was chosen: examine functionality of the existing ArcView 3.3 tool, choose various functions from the tool to convert to VBA within ArcMap, then duplicate the same functionality in the other languages.

The functions chosen from the tool were as follows. 1. Random points: Within a selected group of polygons, generate *n* number of randomly spaced points. 2. Systematic square points: Within a selected group of polygons, generate a matrix of points, evenly spaced by *n* degrees. 3. Systematic triangle points: Within a selected group of polygons, generate a matrix of points, evenly spaced by *n* degrees. Every other row of points, offset by *n* * ½ degrees, resulting in points arranged in a triangular pattern. 4. Tiled hexagons: Within a selected group of polygons, generate a tiling of hexagons, having sides of *n* degrees long, completely covering the area. This was completed by first generating the triangle points, and connecting the points in a hexagonal pattern. The code for each tool could easily be changed to space the points by any unit of measure: meters, miles, feet, etc.

Implement the Tool in ArcMap

Keeping in mind the original project parameters, the true graphical user interface (GUI) – oriented design of the original tool was unacceptable. The original project parameters were to develop a tool that would work within ArcToolbox or Model Builder.

The ArcView 3.3 tool worked by selecting a polygon in ArcView, then processing was done on the selection. The ArcMap tool was developed to start with a shapefile containing polygons. The entire group of polygons served as the input for processing.

Therefore, porting the tool to a command line program that accepts parameters such as shapefile name, distance between points, and output file name could easily be done. From there, the tool could be integrated into ArcToolbox and Model Builder. Previous coursework at Saint Mary's University of Minnesota created a toolbox that contained examples of a command line script that received its parameters from ArcToolbox, thus proving it can be done.

During the development of the VBA code, there was difficulty with implementation of the relational operator function, so an alternate plan was devised: draw a "blanket" of points and hexagons larger than the target area, and use the ArcToolbox tools to clip the points and hexagons that fell within the target area.

Discussions with the DNR resulted in some new ideas and example code using the relational operator, eliminating the need to perform the clip operations. The example code provided many interesting ideas, but the code ended up not being used. Eventually, after experimentation inspired by the example code, the relational operator became workable. With the relational operator successfully implemented, as each point or hexagon is generated, the relational operator method asks: "Does it overlap the target area?" If it does, the feature is saved, otherwise it is ignored. The actual implementation asked "not disjoint" which covered all "true" operations: overlap, within, intersect, etc. With successful use of the relational operator function, the random point function was obtainable – draw random points until the exact desired number of points fall within the target area.

A brief outline of the original point/hexagon tool is stated here:

1. Get extent of the target area
2. Get user input on how far apart to put the points
3. Add a small amount to the extent, to make it bigger
4. Create blanket of points over adjusted extent
5. If hexagons are requested, draw hexagons over adjusted extent
6. Call clip tools to clip points
7. Call clip tools to clip hexagons exactly by target area [clipped hexagons]
8. Call clip tools to do spatial join of hexagons to target area [complete hexagons]

After implementing the relational operator, the logic changed slightly:

1. Create relational operator on target area
2. Get user input on how far apart to put the points
3. Create blanket of points over adjusted extent
4. If hexagons are requested, draw hexagons over adjusted extent
5. If relational operator is not disjoint, save feature

The triangle points are the vertices of the hexagons. Generating the points in this manner do not use the trigonometry functions which require much more processing time. Instead, alternate rows of points are offset by half the distance

between the points. The resulting triangular array of points can be easily connected to create regular hexagons.

Implement the Tool in VB Script

After successful implementation of the functionality in VBA, the same code was attempted in VB Script. VB Script has some drawbacks: no early bound COM objects are allowed. `IDispatch` is the interface that allows discovery of interfaces and methods at runtime – late bound objects. All of the tools shown in ArcToolbox are late-bound COM objects.

ESRI's ArcGIS Desktop Developer Guide, Appendix A provides an unparalleled introduction to using COM and ArcObjects. It states: "The object classes within the ESRI object libraries do not implement the IDispatch interface; this means that these object libraries cannot be used with late binding scripting languages such as Java Script and VBScript, since these languages require that all COM servers accessed support the IDispatch interface" (ESRI, 2004).

Developers can easily see all the methods available for an object by using IntelliSense in VB/VBA. If instantiating an object that uses `GPDispatch`, and the control-space keystrokes are keyed, IntelliSense shows nothing. This additionally supports experimental evidence, where VBScript could not instantiate `IEnvelope` to get extent. In addition, no examples could be found on the web.

Therefore, VB Script is unable to duplicate the functionality in ArcMap. It definitely can be used to script tools published in ArcToolbox because those COM objects implement `IDispatch`, but any early – binding COM objects who do not implement `IDispatch` cannot be called with VB Script.

This does not mean that a tool cannot be created in VB Script using a hybrid of the ArcToolbox tools and compiled VB or C++ components. If the developer would spend the time to develop the compiled components, he/she may as well develop all of the functionality in the compiled language.

Implement the Tool in Python

After termination of the VB Script portion of the project, Python was explored. Python is said to have good COM support, and probably should have the ability to support early binding COM methods.

In Python Programming on Win32, Hammond and Robinson state that Python does support early binding, but only for `IDispatch` objects (Hammond and Robinson 2000). The utility `makepy` will create a Python class from the COM interface for use in the Python program. Here is an example run of `makepy`:

```
D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to
d:\arcgis\python21\win32com\gen_py\2396BA16-B4C6-4D51-
86E7-B466720
6E86Fx0x1x0.py

D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py "dogserver 1.0 Type
Library"
Generating to
d:\arcgis\python21\win32com\gen_py\2396BA16-B4C6-4D51-
86E7-B466720
6E86Fx0x1x0.py
```

Having generated the wrapper Python class, it can be used in a Python script:

```
import win32com.client
w=win32com.client.Dispatch("Dogserver.Dog")
w.Bark()
```

The problem with `makepy` classes is that they do not support early binding COM objects. The discussion by Hammond and Robinson about early

binding actually is an early binding class of a late binding method.

Another COM library, `comtypes`, is available on the web. It does true early binding of COM objects.

`Comtypes` is dependent on another Python package named `ctypes`. `Ctypes` works exclusively with Python 2.3. After installing Python 2.4 from python.org, which is the most current version, the `ctypes` install presented an error requiring version 2.3 of Python to be installed.

Perhaps when porting `ctypes` from 2.3 to 2.4, it was declared problematic or a feature of questionable value, and thus doing early binding COM with Python will only be an anomaly of 2.3. It could also be possible that the package is being written for 2.4 and not yet complete. Perhaps the `ctypes` install is erroneously hard coded to version 2.3, and if the versioning were changed to versions 2.3 and later, it would work for all qualified versions.

Proof of Concept Using VB and C++

Upon experiencing such trouble, it was decided to prove the get extent call would work as a compiled VB and C++ program. First, make the `ienv` example program that minimally does the `IEnvelope` interface to determine the extent of a shapefile. Specific details about `ienv` will be discussed later. Project references are shown in Figure 1.

The resulting programs reported maximum and minimum X and Y coordinates of an input polygon shapefile. They also proved, using compiled VB and C++, the implementation is possible.

Having thus identified the COM objects required to implement some

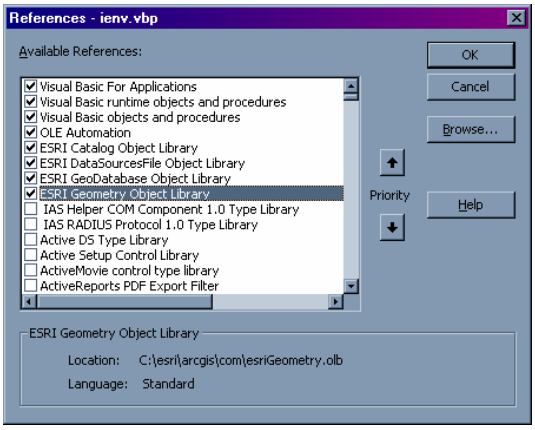

Figure 1. This figure shows the dependent ESRI libraries required to calculate the envelope of a polygon. Screenshot taken from the Microsoft Visual Basic IDE.

functionality, each of the ESRI libraries were run through `makepy` to create the Python wrapper classes:

```
D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to D:\arcgis\python23\lib\site-
packages\win32com\gen_py\2396BA16-B4C6
-4D51-86E7-B4667206E86Fx0x1x0.py

D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to D:\arcgis\python23\lib\site-
packages\win32com\gen_py\ADC7DE29-DC0B
-448E-BBF6-27E4E34CF2ECx0x1x0.py

D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to D:\arcgis\python23\lib\site-
packages\win32com\gen_py\1CE6AC65-43F5
-4529-8FC0-D7ED298E4F1Ax0x1x0.py

D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to D:\arcgis\python23\lib\site-
packages\win32com\gen_py\0475BDB1-E5B2
-4CA2-9127-B4B1683E70C2x0x1x0.py

D:\arcgis\python23\Lib\site-
packages\win32com\client>makepy.py
Generating to D:\arcgis\python23\lib\site-
packages\win32com\gen_py\C4B094C2-FF32
-4FA1-ABCB-7820F8D6FB68x0x1x0.py
```

The desired functionality was still out of reach by merely using the resulting classes. In order to prove the COM objects were able to be instantiated at all, the Python COM browser was used. The entire list of all ESRI objects was shown, which is of considerable size. Examples of a few objects, as seen in the object browser, are shown in Figure 2.

The file names for the objects are also reported. The ESRI `DataSourcesFile` object library filename is shown in Figure 3.

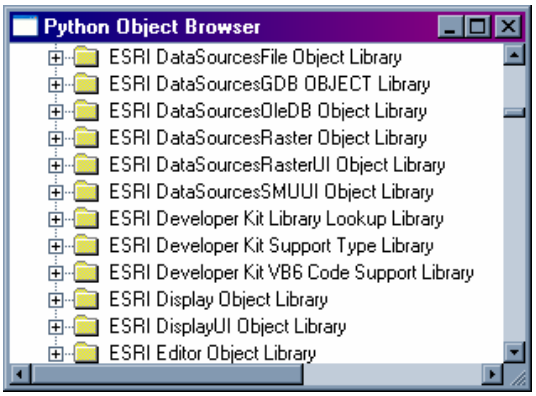

Figure 2. This figure shows an example of the Python Object Browser.

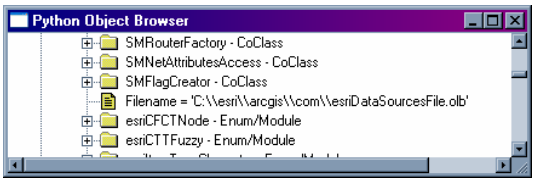

Figure 3. A fully qualified file name shown in the Python Object Browser.

Some properties of some of the objects examined in the object browser presented error messages. An application that uses the methods in the object shown in Figure 4 may not be able to do so using Python. Perhaps the errors shown here are a problem with the object browser, and not a shortcoming of the language specification.

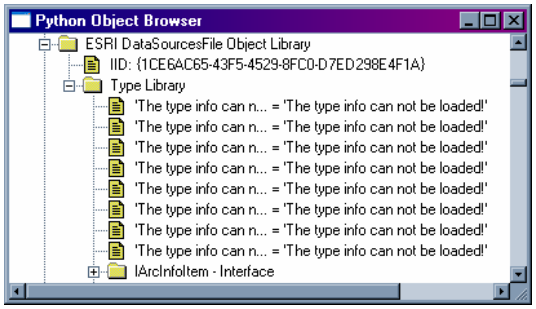

Figure 4. Example type library showing errors in the Python Object Browser.

In later experimentation, a test program used the `esricatalog.olb` module. This approach allowed the object to be instantiated. After the program ran, many files appeared in the `<python dir>\Lib\site-packages\comtypes\gen` folder that were not there before. Such behavior suggests that using the object browser or an application could instantiate and possibly internally `makepy` the objects.

To further prove it is the case, all the objects from the `<python dir>\Lib\site-packages\comtypes\gen` folder were deleted and the program was rerun. It generated the following output instead, which is promising, but also showed some unexpected output:

```
C:\temp\model>ienv.py
getmodule
# Generating
comtypes.gen._1CE6AC65_43F5_4529_8FC0_D7ED298E4F1A_0_1
_0
# Generating
comtypes.gen._00020430_0000_0000_C000_000000000046_0_2
_0
# Generating comtypes.gen.stdole
# Generating
comtypes.gen._0475BDB1_E5B2_4CA2_9127_B4B1683E70C2_0_1
_0
# Generating
comtypes.gen._5E1F7BC3_67C5_4AEE_8EC6_C4B73AAC42ED_0_1
_0
# Generating comtypes.gen.esriSystem
# Generating
comtypes.gen._C4B094C2_FF32_4FA1_ABCB_7820F8D6FB68_0_1
_0
# Generating comtypes.gen.esriGeometry
# Generating
comtypes.gen._59FCCD31_434C_4017_BDEF_DB4B7EDC9CE0_0_1
_0
# Generating
comtypes.gen._4ECCA6E2_B16B_4ACA_BD17_E74CAE4C150A_0_1
_0
# Generating comtypes.gen.esriSystemUI
# Generating comtypes.gen.esriDisplay
# Generating comtypes.gen.esriGeoDatabase
# Generating comtypes.gen.esriDataSourcesFile
done getmodule
do import wkspfact
done import wkspfact
inst obj
done inst obj
Check out spatial extension
done check out
add toolbox
done add toolbox
start dogserver
done dogserver
start shapefile

C:\temp\model>
```

Upon observing the `gen` folder, all of the deleted .py files were re-generated, along with many more.

The next experiment was to call a method that creates a typed return value, and call a method from that returned object. In short, implement the following VB code in Python:

```
Dim pWsFact As IWorkspaceFactory
```

```
Set pWsFact =
CreateObject("esriDataSourcesFile.
ShapefileWorkspaceFactory.1")

Dim pws As IWorkspace
Set pws = pWsFact.OpenFromFile("d:\bb",
0)
```

Step 1 was accomplished in previous experiments: create an `IWorkspaceFactory`. The earlier example [`ienv.py`] did just `CreateObject`. The next step is to call `QueryInterface` on `IWorkspace Factory`, and get an `IWorkspace`, and from that call the method `OpenFromFile`. `OpenFromFile` is shown in Figure 5 in the object browser.

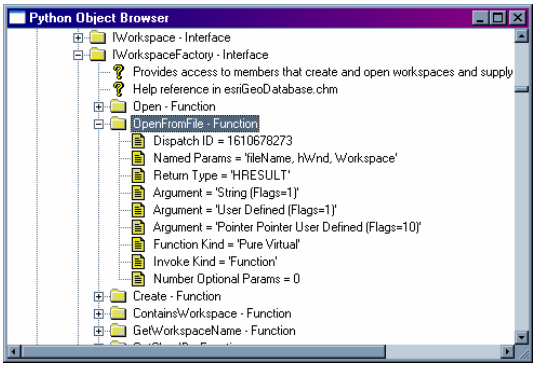

Figure 5. IWorkspaceFactory interface shown in the Python Object Browser.

The only successful way to determine the coclass that was used by `OpenFromFile` was to run the compiled VB application [`ienv.exe`] while concurrently running `regmon`. It showed that the app looked up a coclass named `esriGeoDatabase. AggregateDatasetEnumImpl`. The above code sample, which is the successful code, instantiates `IWorkspace`. Nowhere is `AggregateDatasetEnumImpl` shown. A web search of that coclass name turned up only one example, which did not do a `CreateInstance` of that coclass. In addition, the coclass is not documented anywhere in the ESRI website. The definition of aggregation could be a possible explanation why this coclass is never used explicitly.

The only way the coclass was revealed was to write the target program in another language, run `regmon` to see what COM objects it is using. Only then could the correct COM object be identified and used in the Python code. It seems excessive that such an effort is required to create an app in Python: first create a test app in another language.

Next experiment: After successful instantiation of `pWsFact = createobj (shapefilework spacefactory)`, call `QueryInter face` to transform it into an `IWork spaceFactory`. Here is an example of the generated files:

```
_1CE6AC65_43F5_4529_8FC0_D7ED298E4F1A_0_1_0.py.txt:554
:ShapefileWorkspaceFactory._com_interfaces_ =
[comtypes.gen._00020430_0000_0000_C000_000000000046_0_
2_0.IUnknown,
comtypes.gen._0475BDB1_E5B2_4CA2_9127_B4B1683E70C2_0_1
_0.IWorkspaceFactory,
comtypes.gen._0475BDB1_E5B2_4CA2_9127_B4B1683E70C2_0_1
_0.IWorkspaceFactory2]
```

Within the generated files, the section shown here defines the workspace factory:

```
class ShapefileWorkspaceFactory(CoClass):
    u'ESRI Shapefile Workspace Factory.'
    _reg_clsid_ = GUID('{A06ADB96-D95C-11D1-AA81-
00C04FA33A15}')
    _idlflags_ = []
    _reg_typelib_ = ('{1CE6AC65-43F5-4529-8FC0-
D7ED298E4F1A}', 1, 0)
ShapefileWorkspaceFactory._com_interfaces_ =
[comtypes.gen._00020430_0000_0000_C000_000000000046_0_
2_0.IUnknown,
comtypes.gen._0475BDB1_E5B2_4CA2_9127_B4B1683E70C2_0_1
_0.IWorkspaceFactory,
comtypes.gen._0475BDB1_E5B2_4CA2_9127_B4B1683E70C2_0_1
_0.IWorkspaceFactory2]
```

Here is the final implementation in the Python code:

```
pWsFact =
comtypes.client.CreateObject("esriDataSourcesFile.
ShapefileWorkspaceFactory")

o1 = pWsFact.QueryInterface(comtypes.IUnknown)
o2 =
o1.QueryInterface(comtypes.gen._0475BDB1_E5B2_4CA2_912
7_B4B1683E70C
2_0_1_0.IWorkspaceFactory)

print "now open files"
pws = o2.OpenFromFile("c:\\temp\\model",0)
print "done open files"
```

It can be seen that o2 is of the proper interface type to call the

`OpenFromFile` method. This is the desired COM object:

```
IWorkspace._methods_ = [
    COMMETHOD(['propget', helpstring(u'The connection
properties of the workspace.')], HRESULT,
'ConnectionProperties',
              ( ['retval', 'out'],
POINTER(POINTER(comtypes.gen._5E1F7BC3_67C5_4AEE_8EC6_
C4B73AAC42ED_0_1_0.IPropertySet)),
'ConnectionProperties' )),
    COMMETHOD(['propget', helpstring(u'The factory
that created the workspace.')], HRESULT,
'WorkspaceFactory',
              ( ['retval', 'out'],
POINTER(POINTER(IWorkspaceFactory)), 'Factory' )),
    COMMETHOD(['propget', helpstring(u'The datasets in
the workspace.')], HRESULT, 'Datasets',
              ( ['in'], esriDatasetType, 'DatasetType'
),
              ( ['retval', 'out'],
POINTER(POINTER(IEnumDataset)), 'Datasets' )),
    COMMETHOD(['propget', helpstring(u'The
DatasetNames in the workspace.')], HRESULT,
'DatasetNames',
              ( ['in'], esriDatasetType, 'DatasetType'
),
              ( ['retval', 'out'],
POINTER(POINTER(IEnumDatasetName)), 'DatasetNames' )),
    COMMETHOD(['propget', helpstring(u'The file system
full path of the workspace.')], HRESULT, 'PathName',
              ( ['retval', 'out'], POINTER(BSTR),
'PathName' )),
    COMMETHOD(['propget', helpstring(u'The Type of the
Workspace.')], HRESULT, 'Type',
              ( ['retval', 'out'],
POINTER(esriWorkspaceType), 'Type' )),
    COMMETHOD([helpstring(u'TRUE if the workspace is a
file system directory.')], HRESULT, 'IsDirectory',
              ( ['retval', 'out'],
POINTER(VARIANT_BOOL), 'isDir' )),
    COMMETHOD([helpstring(u'Checks if the workspace
exists.')], HRESULT, 'Exists',
              ( ['retval', 'out'],
POINTER(VARIANT_BOOL), 'Exists' )),
    COMMETHOD([helpstring(u'Executes the specified SQL
statement.')], HRESULT, 'ExecuteSQL',
              ( ['in'], BSTR, 'sqlStmt' )),]
```

After weeks of experimentation, the integrity of the Python development environment began to decay. When running `pythonwin`, Windows Explorer and Notepad freeze and become unusable until `pythonwin` is exited. Python documentation states `CoUnInitialize` is not working correctly, and experience confirms it. About 30 seconds after the script completes execution, Windows presents an error message.

Such behavior raised suspicion that critical Python system files were corrupt, so Python was completely uninstalled and re-installed. It unfortunately still showed the same undesirable behavior after reinstalling Python.

It is entirely possible that the author's inexperience with Python caused some of the observed behavior, but it is evident that many of the problems identified are still valid problems regardless of the experience of the developer.

Compiled VB and C++ Restated

In the process of trying to use Python, portions of the code were successfully written in compiled VB and C++. It is therefore possible to implement the code in the compiled languages. This statement is important: C++ can implement any functionality, especially COM implementations. Compiled VB code is syntactically identical to the VBA code written within ArcMap, and therefore is equally able to implement the desired functionality. There was no need to prove these two languages were viable by completely implementing the solution demonstrated in VBA. Small programs that demonstrated various parts of the complete solution, along with the whole solution within ArcMap, were sufficient.

Implement the Tool in C# .NET

ESRI provides a wealth of documentation and code samples from both ESRI documentation and from the user community in the ESRI user forums. As stated in the introduction, the VBA implementation of ArcObjects is mature and well documented, and the number of C# code samples is steadily increasing.

Therefore, it was relatively easy to continue the development paradigm using C#: find C# samples on the web, modify the samples to incorporate the desired functionality, and repeat.

The resulting application was completely command line driven; a command line program, using parameters such as shapefile name, distance between points, and output

shapefile name, can be typed from the command line or invoked from ArcToolbox.

Results

Original expectations were that all languages would be equal and an organization could choose to use any language that best suited its IT staff.

However, the manual stated that VB script was unable to support early binding COM objects. If a homogenous VB script solution is strongly desired, an IDispatch wrapper/shell for the required COM objects of the application at hand could be created. The drawback of this solution is that any future updates to the underlying objects could break the wrapper and need to be updated accordingly.

Python was also disqualified. It was better than VBScript in that it supports early binding COM objects, but not without problems, and some of the objects were not able to be supported (Figure 4). There was also the question of ctypes's availability only in Python 2.3.

VBA, integrated into ArcMap, was able to deliver all functionality without problem. Additionally, compiled VB and C++ demonstrated their abilities to support the functionality. C#, which implements the .NET language specification, was able to deliver the desired functionality. Because all .NET languages support the common language specification, all .NET languages may be considered identical and are thus able to deliver the desired functionality.

Development Time

Because not all languages were able to deliver the final product, only VBA and C# will be discussed.

Most development is an iterative process of "find an example," "modify the example to more closely match end result," and repeat.

It may not be a fair comparison to redo the same exact application in all languages to see which one is easier to develop. Lessons learned in writing the program in one language can be exploited in the other languages. Each language should be used to write a different end product, of comparable complexity, in order to eliminate such learning experiences. There is also the unfair advantage of a language the developer has more experience in.

All in all, the development time of VBA and C# were fairly equal. Part of the development time was learning the ArcObjects object model. The ArcObjects model is much larger and more complex than many developers encounter.

Execution Time

After the languages VBA and C# had been proven as viable languages, their execution times were compared.

The first test was conducted on the original computer the applications were developed on: an HP Pavilion dv5000 with an AMD Turion 64 bit 1.8 GHz with 1 GB RAM, running 32 bit Windows XP Professional.

The VBA application performed acceptably, taking less than a minute to create approximately 500 points and 200 hexagons over an area of 250,000 square miles. In comparison, the C# application took 37 times longer to perform the exact same thing.

This prompted a careful analysis of the source code, which revealed little room for improvement. The only questionable code, which followed the ESRI example of efficient creation of polygons, did the following within a function that was called repeatedly; once for each hexagon:

```
IPoint pPointsRing0 = new PointClass();
IPoint pPointsRing1 = new PointClass();
IPoint pPointsRing2 = new PointClass();
IPoint pPointsRing3 = new PointClass();
IPoint pPointsRing4 = new PointClass();
IPoint pPointsRing5 = new PointClass();
IPoint pPointsRing6 = new PointClass();
```

ESRI code samples provided with the ArcGIS install were described as the most efficient way to create polygons. Perhaps, in the case of this application, where many polygons were created, it is not the most efficient method. It may be efficient for the creation of a single polygon.

The code was changed to declare and instantiate the `PointClass` objects once, outside the function, as member variables to avoid duplicate construction/destruction. This modification improved the performance by 4 minutes. Figure 6 shows the improvement between "C# physical 1" and "STA physical."

Additional analysis of COM performance prompted comparison between apartment threading models to eliminate marshalling overhead. Again, Appendix A of the developer guide provides more information on COM, briefly outlining the meaning of apartment threading models (ESRI, 2004).

All subsequent experiments compare the performance of the C# application using both threading models. Experiments shown on the bar graphs are designated as running the STA or MTA threading models.

Additionally, the applications were compared on operating systems running on Microsoft Virtual PC and Microsoft Virtual Server 2003. Experiments shown on the bar graphs are designated as running physical or virtual.

Knowing these facts about the experiments and environments, the names of the experiments on the graphs may be interpreted as follows:

1) VB physical: VBA app run on physical computer.
2) VB virtual: VBA app run on virtual image, run on that physical computer.
3) C# physical: C# app, default threading model, run on physical computer.
4) STA physical: C# app, explicitly coded to use STA threading model, run on physical computer.
5) STA virtual: C# app, explicitly coded to use STA threading model, run on virtual image, run on that physical computer.
6) MTA physical: C# app, explicitly coded to use MTA threading model, run on physical computer.
7) MTA virtual: C# app, explicitly coded to use MTA threading model, run on virtual image, run on that physical computer.

The virtual system running on the HP dv5000 computer is Microsoft Virtual PC, running Microsoft XP Professional, using 660 MB RAM.

To prove the performance issues are not an anomaly of a single machine, the same application was installed and run on other computers.

The next experimental machine was a newly built Dell Dimension 4400 with a Pentium 4 1.33 GHz with 750 MB RAM, running Windows XP

Professional. The OS was installed from scratch, ArcGIS 9.1 was installed, and the application was copied to the computer and run. This clean install removes any risk of system corruption from continued development, installation

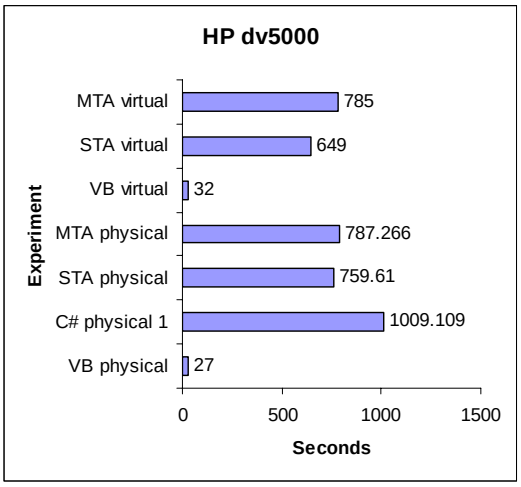

Figure 6. HP dv5000 experimental results.

and uninstallation of various products. All subsequent experiments were completed in this way: clean install of Windows, ArcGIS, and the application.

The virtual computer running on this system is Microsoft Virtual PC running an instance of Microsoft XP Professional with 555 MB RAM.

Results of this machine are outlined in Figure 7.

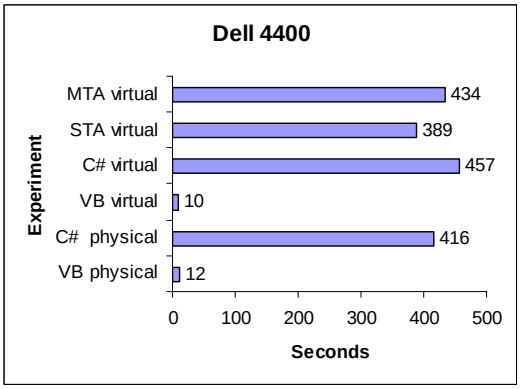

Figure 7. Dell 4400 experimental results.

Computer time was acquired on an instance of Microsoft Windows Server 2003 Enterprise Edition, running on Microsoft Virtual Server 2003 at 1.33 GHz and 1.95 GB RAM, on a Dell 2950 host with dual Intel Xeon 5150 processors at 2.66 GHz with 16 GB RAM running Microsoft Server 2003 R2 Enterprise x64 Edition. Figure 8 shows the results on this virtual machine.

It was observed that the applications run consistently faster on the virtual machines.

It was observed that memory usage was low and the majority of delay in the C# app was because it was I/O bound, repeatedly accessing the registry.

While running the slow C# app, regmon results were monitored. It could be seen that the app was continually re-creating a COM object because it repeatedly got the same registry key in `HKEY_CLASSES_ROOT`.

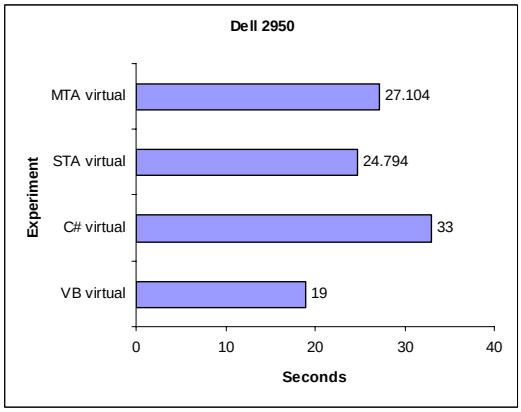

Figure 8. Dell 2950 experimental results.

This behavior draws the conclusion that the .NET classes are just COM interop wrapper classes. They are not doing anything more than adding extra overhead, as the following pseudo code outlines:

Create instance of class
Call class method

Class creates COM object
Class calls COM object
Class releases COM object
Class goes out of scope; it is destructed.

ESRI's ArcObjects is the single most extensive COM object model in the world. ESRI cannot "simply" rewrite everything overnight. They can, however, create several thousand wrapper classes with one line of code:

```
for %i in (*.olb) do tlbimp %i
/out:%i.dnet.olb
```

In time, ESRI will rewrite all objects in pure C#. The COM interop overhead will then be eliminated.

To demonstrate, a simple COM object [`dogserver`], that appends a single line containing date and time to a text file, was written in C++, and called with a C++ caller.

Afterwards, a COM interop wrapper was created with tlbimp as demonstrated above, and called from a C# caller. It was understandably slower because of the added overhead of the wrapper class created by tlbimp.

Next a pure C# implementation [`catserver`], which also appends date and time to a text file, saved as a callable C# class in a DLL assembly, was called from a C# caller. It turned out to be faster than the original COM program written in C++.

Figure 9 shows the results. `Caller` is the C++ program that calls `dogserver`. `CSCaller` is a C# program that calls `dogserver` using COM interop. `Catcaller` is the C# program that calls `catserver`. All three programs call the method to write to the text file 5000 times.

This gives strong evidence that although code written with today's C# class wrappers to ArcObjects may run slow, in time ESRI will rewrite them in pure C# and vastly improve their performance.

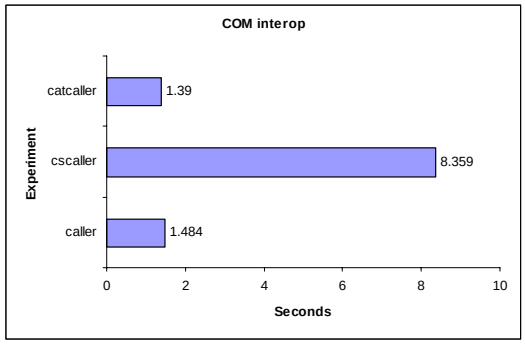

Figure 9. COM interop experimental results.

Hopefully in the interim, ESRI will rewrite the most heavily used COM objects first, so the benefits demonstated here can be reaped as soon as possible.

Discussion

It was proven that some languages are unable to deliver complete functionality. It was also found that the development time was roughly equal.

The reason for this research topic was such that many organizations are faced with maintaining an obsolete, unsupported product. They must move forward to a more current technology. The agonizing question: which language is best for the future of the organization?

With development time, maintainability, online support, and industry acceptance of VB, VBA, C++, VB.NET, and C# being equal, it appears that the longevity of the chosen language should be the deciding factor. Microsoft has announced the transition of Visual Studio 6 (VB, C++) from full support to end of lifetime support (Microsoft, 2006). Microsoft is also strongly recommending that organizations migrate their code to the new languages that support .NET. ESRI

and Microsoft both show examples of web services, VB.NET, and C# to leverage the latest languages.

Knowing this, it is quite evident that programs written in the more recent languages, such as Visual Basic 6, will soon need to be migrated to the newer .NET compliant languages.

Therefore, the strategic solution would be migration of Avenue directly to .NET if possible, and thus prevent another costly round of refactoring. Some organizations convert their code in a reactionary way with little thought to consider the future. This study has shown what languages are viable, and provides evidence to help make a prudent choice regarding the future.

Applications needing quick response time may need to be written in VBA in the interim until ESRI completes its C# rewrite of all ArcObjects.

Acknowledgments

I would like to thank Tim Loesch of the Minnesota DNR and the Resource Analysis staff at Saint Mary's University of Minnesota for their help during this project.

References

Environmental Systems Research Institute, Inc. 2004. *ArcGIS Desktop Developer Guide, 139*, Online manual provided with install of ArcGIS 9 software.

Environmental Systems Research Institute, Inc., *Product Lifecycle Support Policy: ArcView GIS (3.3),* Retrieved June 2006 from http://downloads.esri.com/support/product%20life%20cycle/other_/589arcviewgis.pdf.

Environmental Systems Research Institute, Inc., *Welcome to ArcObjects Online,* Retrieved June 2006 from http://edndoc.esri.com/arcobjects/8.3/default.asp.

Hammond, M. and Robinson, A. 2000. *Python Programming on Win32,* O'Reilly & Associates, Sebastopol, CA.

Marble, D. F. 2005. Defining the Components of the Geospatial Workforce – Who Are We? *Arc News Vol. 27 No. 4, 1.*

Microsoft Corporation, *Microsoft Support Lifecycle,* Retrieved June 2006 from http://support.microsoft.com/lifecycle/?LN=en-us&p1=3003&x=15&y=10.

Minnesota DNR. 2005. *DNR Sampling Tool (V2.8 Nov 09,2005),* Minnesota DNR, Saint Paul, MN.