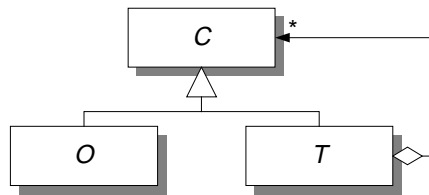


*Evaluation of COM Support in
Development Environments
COT/3-4-V1.1*



Centre for Object Technology

Revision history:	27-06-98	V0.5	Initial draft.
	06-08-98	V0.6	Included Delphi and VJ++ eval. summary.
	10-08-98	V0.7	VB eval. summary included.
	10-08-98	V0.8	VC++ eval. summary included.
	11-08-98	V0.9	Format changes.
	20-08-98	V1.0	Conclusion included.
	08-09-98	V1.1	General revisions.

Author(s): Peter Petersen, Aarhus University
Kåre Kjelstrøm, Aarhus University
Morten Grouleff, Aarhus University
Peter Andersen, Aarhus University
Henrik Lykke Nielsen, Danish Technological Institute
Kristian Lippert, Danish Technological Institute
Steen D. Olsen, Danish Technological Institute
Kim Vestergaard, Danish Technological Institute
René Elmstrøm, Danish Technological Institute

Status: Final

Publication: Public

Summary:

This document summarizes a comparative study of four development environments support for COM based development. The development environments included are: Borland Delphi, Microsoft Visual J++, Microsoft Visual Basic and Microsoft Visual C++/ATL.

© Copyright 1998 Aarhus University, Danish Technological Institute.

Contents

1. INTRODUCTION.....	4
2. THE PROCESS AND RESULTS OF THE EVALUATION.....	4
3. PROPOSED USE OF THE EVALUATION.....	5
4. SUMMARY OF COM SUPPORT IN FOUR DEVELOPMENT ENVIRONMENTS	5
4.1 SUMMARY OF COM SUPPORT IN BORLAND DELPHI.....	5
4.1.1 <i>General</i>	5
4.1.2 <i>Component Server</i>	6
4.1.3 <i>Compilation and Distribution</i>	7
4.1.4 <i>Component Client</i>	7
4.1.5 <i>Development Environment</i>	8
4.1.6 <i>Development Process</i>	8
4.1.7 <i>Auxiliary</i>	8
4.1.8 <i>Overall Impression/Summary</i>	8
4.2 SUMMARY OF COM SUPPORT IN MICROSOFT VISUAL J++.....	9
4.2.1 <i>General</i>	9
4.2.2 <i>Component server</i>	9
4.2.3 <i>Compilation and distribution</i>	9
4.2.4 <i>Component client</i>	9
4.2.5 <i>Development environment</i>	9
4.2.6 <i>Development process</i>	10
4.2.7 <i>Overall impression and summary</i>	10
4.3 SUMMARY OF COM SUPPORT IN MICROSOFT VISUAL BASIC.....	10
4.3.1 <i>General</i>	10
4.3.2 <i>Component Server</i>	11
4.3.3 <i>Compilation and Distribution</i>	12
4.3.4 <i>Component Client</i>	12
4.3.5 <i>Development Environment</i>	12
4.3.6 <i>Development Process</i>	13
4.3.7 <i>Auxiliary</i>	13
4.3.8 <i>Overall Impression and Summary</i>	13
4.4 SUMMARY OF COM SUPPORT IN MICROSOFT VISUAL C++ USING THE ATL LIBRARY	13
4.4.1 <i>General</i>	13
4.4.2 <i>Component Server</i>	14
4.4.3 <i>Compilation and Distribution</i>	15
4.4.4 <i>Component Client</i>	15
4.4.5 <i>Development Environment</i>	15
4.4.6 <i>Development Process</i>	15
4.4.7 <i>Overall Impression and Summary</i>	16
5. CONCLUSION.....	17
5.1 <i>DIRECTNESS OF COM SUPPORT</i>	17
5.2 <i>COM INTEGRATION AND SUPPORT IN IDE</i>	17
5.3 <i>SIZE OF COMPONENT INSTALLATIONS</i>	18
5.4 <i>EFFICIENCY OF USE</i>	19
6. APPENDIX A	20
6.1 <i>RATING ON DIRECTNESS OF COM SUPPORT</i>	20
6.2 <i>RATING ON SUPPORT AND INTEGRATION IN IDE</i>	20
6.3 <i>RATING ON SIZE OF GENERATED COMPONENTS</i>	21

1. Introduction

Component based development has shown to be a promising technology to support the development of flexible software applications and to promote the reuse of software. The component technology of Microsoft Windows has until this moment proven to find the most widespread use, partly due to success of Microsoft Windows itself. The component technology has been marketed under a number of different names: OLE (Object Linking and Embedding), ActiveX and COM/DCOM (Distributed Component Object Model). (D)COM is the name of the underlying technology and it is an embedded service in the Microsoft Windows family of operating systems.

In this document a summary of the evaluation of COM based development in four different development environments is presented. The environments evaluated are:

- Borland Delphi version 3.01
- Microsoft Visual J++ version 1.1
- Microsoft Visual Basic version 5.0 and
- Microsoft Visual C++ version 5.0 using ATL version 2.1

2. The Process and Results of the Evaluation

The evaluation of development environments has been performed in a number of steps:

1. A so called "Pilot" defining a small example application using different aspects of COM of was defined. In the definition an application to be developed and a number of incremental changes to this application were defined.

The definition of the Pilot can be found in:

COT/3-15: The COM Pilot Application

2. The "pilot" application was developed in four development environments. The results of this activity is reported in:

COT/3-10: Development of the Pilot Application in Delphi

COT/3-11: Development of the Pilot Application in Visual Basic

COT/3-12: Development of the Pilot Application in Visual C++

COT/3-13: Development of the Pilot Application in Visual J++

3. The Pilot application defines a COM client project and a COM server project. COM technology supports the integration of components developed in different programming languages and development environments. This ability was tested in the next activity where the COM servers developed in the different environments were tested from COM clients developed in all four development environments. The result of this integration highlighted a number of differences in the strategy for developing and using COM components. The result of this activity was described in the taxonomy (see point 4 below).

4. To further compare the support for COM development in the four environments investigated, a list of questions (called a taxonomy) were answered for each of the

four environments. The questions and answers for each environment can be found in:

COT/3-5: Evaluation of COM Support in Delphi

COT/3-6: Evaluation of COM Support in Visual Basic

COT/3-7: Evaluation of COM Support in Visual C++ using ATL

COT/3-8: Evaluation of COM Support in Visual J++

5. Finally a summary of the evaluation of the support for COM based development in each of the four development environments were written. This summary can be found in chapter 4 of this document.

To fully understand and appreciate the different parts of the evaluation a basic understanding of the COM technology is required. We recommend *Inside COM* by Dale Rogerson (Microsoft Press 1997) to be consulted for a basic understanding of COM.

3. Proposed use of the evaluation

We expect the evaluation presented in this document and in the related documentation to be useful for software developers and project managers in a number of situations:

1. The evaluation can be used to select the most suited development environment/mix of development environments to be used in a specific situation. For this purpose the summary characteristic of COM support in each environment presented in the next chapter would be the starting point. The pilot application and especially the COM taxonomy for the environments should be consulted for details in the COM support of the environments.
2. The evaluation can be used to evaluate the consequences of moving to component based development using a specific development environment. In that case the summary evaluation, the taxonomy and the pilot application for the specific development environment should be consulted.

4. Summary of COM Support in Four Development Environments

4.1 Summary of COM Support in Borland Delphi

4.1.1 General

Delphi is an IDE for Windows95/NT, which is developed by former Borland Inc., now Inprise Inc. Based on the original Turbo Pascal compiler by Anders Hejlsberg, Delphi allows the development of 32-bit windows programs in Object Pascal. The COM support described in this section relates to Delphi 3.01, and not to older versions of the IDE. From version 2.0 to 3.0, the language saw dramatic changes in that interface

facilities like those present in the Java language were added. This was probably done in order to make the COM language mapping prettier and more handy.

4.1.2 Component Server

The development of a COM component in Borland's Delphi begins with the programmer deciding upon the type of server it should reside in. Delphi allows both flavours, in-process and out-of-process servers, to be built.

The transition from one kind of server to another is possible, but not trivial. As Delphi forces the developer to make the choice of server at the beginning of the project's lifetime, a new project is needed to make the transition, and old code copied and pasted. Adding new components to the object server, on the other hand, is easy, conveniently supported by the different wizards. Any kind of COM object may be created, and Delphi even has support for ActiveX and property pages as well.

The content of the COM server is created and modified through the Type Library Editor, which will eventually spit out skeleton code. The generated code only lacks the specification of procedure and property bodies, and Delphi creates a type library wrapper file, which maps IDL entities to Pascal.

If the type library definition of the object server should be changed after some of the method or property bodies have been implemented, the Type Library Editor will nicely leave the implementations alone, while changing the definitions. This means that reengineering code is not a problem.

A problem, on the other hand, is using other people's type libraries or implementing predefined interfaces. The Type Library Editor will let the developer open existing type libraries, but not connect them to the project. Connections, i.e. references, can be added to libraries in the Windows Registry, but doing so will not allow the use of the types in those libraries in the definition of interfaces and methods. According to the rules of type libraries, it should be possible to use all types in the transitive closure of referenced type libraries. By cleverly renaming an existing type library that was not made with Delphi to be called the same as the object server project file, but with the .TLB extension instead, any type library can be used, though.

Since any kind of servers can be built in Delphi, including ones that use predefined type libraries, standard marshalling is an issue, the tool should deal with. Unfortunately it does not do that, and the developer is forced to use MIDL and a C compiler to get the proxy/stub DLL. That DLL will, however, work fully with Delphi components. Type library marshalled components, on the other hand, are not a problem.

The language mapping from IDL types to Pascal types is completely specified, and that means that Delphi will handle any types including SafeArrays and Variants as well. User defined types such as structures and unions will also be converted to Pascal equivalents, and thus be usable from the code. This is true even if these types cannot be defined in the Type Library Editor.

The Delphi COM framework provides all the API functions needed for working with components, but also wraps up the details that a developer is basically not interested in: QueryInterface can be called directly on all interfaces, but it is also possible to type-cast CoClass variables to the desired interface, which will have the same effect as creating the CoClass and then query for the desired interface.

When using Delphi's wrapper to create the CoClass, it is not necessary to call AddRef and Release explicitly, since the runtime engine will keep track of all interface variables and simply do an implicit AddRef, when assigning an interface to a new variable. Similarly Release will be invoked when such a variable goes out of scope or is assigned a new value, e.g. "nil".

The same philosophy applies to interfaces that inherit from IDispatch: Delphi hides the details of Invoke, but lets the developer do all the hard work too, if he has masochistic tendencies. The HRESULT and exception error handling strategies are likewise wrapped up in Delphi equivalents, but again the details can be manually handled if desired.

4.1.3 Compilation and Distribution

Once a component server has been defined, the programmer will be interested in packing up the files that belong to it. The Install Shield program, a Delphi specific version of which ships with the Client/Server release of the IDE, can be used for this purpose, but it will only handle projects that contain an executable file, and can thus not be used with in-process servers. For that reason, an installation package might as well be built by hand.

Fortunately all object servers will contain either an implementation of the API functions DllRegisterServer and DllUnregisterServer or will be self registering, when run as ordinary programs. It is worth to note that in-process servers may be registered and unregistered from Delphi's menus as well.

The target types of the final build may be either thin servers that do not link the Delphi runtime engine or thick ones that do. When planning to install many components that were made with Delphi, the first choice is the best, otherwise the second will take up the smallest amount of bytes.

4.1.4 Component Client

Using and reusing Delphi COM components can be done in any of the three possible ways, depending on the kinds that are legal with that particular server. Components that only expose a V-Table, as well as ones that provide dual interfaces, may thus be created. The clients may bind very early through the V-Table, early through the dispIDs obtained at compile time, or late at runtime through dispIDs.

When it comes to building components from components, Delphi objects are readily aggregateable from birth, which means that they can be the inner of an aggregate. To create the outer, though, a little work is needed, since the virtual methods of the

interface “ObjQueryInterface” must be redefined in order to take the aggregates into consideration.

In a manner quite analogous to aggregation, Delphi exposes no direct support for containment, but there is no stopping the developer from creating the contained component himself and manually adding the statements that delegate calls. This process is in fact so simple that wrapping it up in the language seems superfluous.

4.1.5 Development Environment

Components are integrated into the Delphi IDE in such a manner that they can be manipulated as any other Delphi program. This means that the debugger will work; for in-process servers only the specification of a client executable’s name is needed, and out-of-process servers will just have to be run from within the Delphi environment to get this functionality. It is worth to note that by starting multiple instances of Delphi it is in fact possible to debug from client into server and back again. This is also true if the client or the server is being developed in another IDE, e.g. J++ or Visual C++.

4.1.6 Development Process

The Delphi IDE allows the developer to develop COM components as he would any other program. This means that it is possible to insert break-points, single-step through the code and add watches on variables.

The Delphi IDE does not impose any restrictions on the GUIDs and IIDs for the components. It is the responsibility of the programmer to change these to reflect version evolution.

4.1.7 Auxiliary

The Delphi IDE has a strong support for many aspects of Microsoft’s component technology, including OLE and ActiveX. Recently Borland fusioned with the CORBA vendor Visigenic to form Inprise Inc. This means that the next version of the Delphi IDE will see an ORB and CORBA support included.

4.1.8 Overall Impression/Summary

All in all the Delphi IDE appears strong and flexible. It has a number of quirks that need to be taken care of in the Type Library Editor, and there is no good support for standard interfaces. These problems are not crucial though, work-arounds can help overcome the limitations.

4.2 Summary of COM Support in Microsoft Visual J++

4.2.1 General

The Java programming language is a reasonably simple object oriented language in the Simula tradition. It has a syntax very much like C++ but fortunately lacks most of the features that make C++ so complex. Java has single inheritance and not multiple inheritance. Java classes can implement multiple interfaces. This means that describing COM interfaces and COM components with multiple interfaces is straightforward in plain Java. Java has a very strict type system. This can cause problems in a COM context, where other components may require specific types that Java cannot map. An example is using the C type void*.

4.2.2 Component server

A component server in Java is a public Java class. It is not possible to have two public classes in a class file. All class files are loaded using msjava.dll, which is distributed with Internet Explorer. Java classes using COM will not work with any other Java Virtual Machine (JVM). Java components are always INPROC. The user does not have to worry about details like the Class Factory, implementing QueryInterface or reference counting. Standard HRESULTs are mapped into exceptions and standard exceptions are returned to the callee using HRESULTs. Java components are aggregateable. A Java class can aggregate a single component as aggregation is implemented as inheritance.

4.2.3 Compilation and distribution

Compiling the component consists of selecting "Build" from the menus. The Java ActiveX wizard does the registration the first time. Distribution of the class files is not very well supported. You have to write a registration script using a BAT file or the like, and you have to find out yourself what other class files your component might depend on. There is no automatic installation program creator either.

4.2.4 Component client

It is possible to interface to any COM server provided the types used in the function arguments are of acceptable types. The first step in getting access to a component is to compile the type library describing the component into a special Java class using the Java type library wizard. The component may now be instantiated using the Java keyword new. This encapsulates the call to CoCreateInstance. To get hold of another interface pointer on a component is done by using a regular Java cast of pointer.

4.2.5 Development environment

Visual J++ is integrated in the Microsoft Developer Studio, which is an integrated development environment. It may also be used with Microsoft compilers for FORTRAN and C++. This has the advantage that developers switching from one of these will

quickly feel at home. On the other hand there is not much explicit support for Java, and the overall impression is that this is a quick hack to get a Java environment to the market.

4.2.6 Development process

Creating a simple server is done by writing a public Java class with the functionality you wish to export. This is then exported as a COM component using the Wizard for that purpose. It essentially just generates a line of code setting the CLSID in the source file and directs the user to change a few build options. An IDL and a type library file describing the component are generated. Building a more complex server with multiple interfaces or having control over the IIDs used requires the user to write an IDL file describing the component first. This is not that difficult, as the generated IDL file may be edited and compiled from within the IDE.

4.2.7 Overall impression and summary

Java could be the perfect programming language for developing COM components. The built-in notion of interface closely matches that in COM.

Future versions with stronger support for COM should include a graphical type library editor or component designer, a complete implementation of all the standard components, from which new ones could easily be constructed, and an interface builder that supports designing and using COM for visual components. This seems to be the aim in version 6.0 judging from the beta version downloadable from Microsoft. But there is a long way to go, as most of it is still not included.

4.3 Summary of COM Support in Microsoft Visual Basic

4.3.1 General

Visual Basic is both a development environment as well as a programming language.

The programming language is a standard third generation programming language with a syntax strongly influenced by Basic, Comal 80 and Pascal. Execution of a Visual Basic program is based on an events model, where - primarily - user interface elements "generates" events which the application programmer can fill out with application specific code. The language is very similar to the language in Borlands Delphi development environment.

Visual Basic programs are structured in modules (one file per module). Examples of modules could be a class, a code module, a usercontrol, a form etc.

The major structuring mechanism which underlies both Visual Basic as a tool as well as the programs constructed using Visual Basic is encapsulation.

The development environment is based on three major facilities:

1. A very direct and interactive GUI designer in which the user interface is "drawn" using visual controls. This type of GUI designer is standard in most Windows development tools of today.
2. An interactive debugger which enables (see COT/3-6 4.1.5 and 5.1):
 - immediate execution of programs without explicit compilation
 - single-stepping and change of order-of-execution, use of breakpoints, monitoring and change of the values of variables, and editing of the program code while existing variables are instantiated and the program is "running".
3. Extensibility using binary components in the form of e.g. ActiveX Automation Servers and visual ActiveX Controls. Visual Basic has very good integration of type-library information (syntax info etc.) into the development environment (see COT/3-6 4.1.2).

4.3.2 Component Server

Using standard Visual Basic it is not possible to create arbitrary COM components. All components created using Visual Basic will have dual interfaces (see COT/3-6 1.4.1.3 and the COT/3-11 section: "Pilot Application Implementation Process").

The only disadvantage that this may represent though, is the added memory used by supporting both IDispatch and custom interfaces. In most cases this will be negligible.

Visual Basic's standard packages are all based on the ActiveX technology (specific collections of interfaces) such as ActiveX Code Components (that is Automation Servers, both EXE and DLL), ActiveX Controls and ActiveX Documents.

An interface in Visual Basic is specified as the collection of members in a class module. It is not possible to create a component directly using an IDL-file, but it is possible to implement the interfaces of a component defined in a type-library (which could e.g. be constructed using the MIDL-compiler) (see COT/3-6 1.4.4.3 and COT/3-11 phase 1.1 subsection "Inheriting from a type-library"). Visual Basic will always include a type-library in a server, but it is also possible to create independent .tlb-files using Visual Basic (see COT/3-6 1.10 and COT/3-11 phase 1.1 subsection "A single default interface").

As a result of Visual Basic's general encapsulation approach the COM features of Visual Basic come in standard packages of functionality (see COT/3-6 1.3). This is in most cases a significant advantage: It makes it very simple and fast for even the novice to create and use components. But in some circumstances it would be nice to have low-level access to COM features in the same way as in e.g. Visual C++. An area in which Visual Basic is weak is in the use of GUIDs. It is not possible to specify the GUID for a component which is being created. And it is not possible for a client to specify components and interfaces using GUIDs (see COT/3-11 phase 1.1 subsection "Inheriting from a type-library") but only ProgIDs (in Visual Basic this would be as *ProjectName.ClassName*). For further limitations in how clients can specify interface names see COT/3-11 phase 1.1 subsection "Building a client for Deep_Thought1_1a.dll".

These limitations are not very often of great importance and the limitations themselves protects the developer from making difficult-to-spot errors. But as well as encapsulation is one of the major benefits of Visual Basic it is also a possible source of irritation because of the limitations it imposes on the use of COM.

4.3.3 Compilation and Distribution

Visual Basic's *Binary Compatibility* option helps the developer to comply with a formerly defined interface (e.g. in a distributed component) when compiling a server. Warnings are issued - and restrictions are to a certain degree enforced - if the interface is violated, so that existing - that are possibly already distributed - clients will continue to work also against the new version of the server.

Visual Basic comes with a tool - the Setup Wizard - for creating setup programs for components as well as end user applications. It handles the details of determining which dependent files to distribute; and the installation as well as registration of the included components at the client machine. Setup Wizard - which is a Visual Basic specific tool - is able to create installations for both disk-based and internet-based installations.

Most of the technicalities involved in compiling and distributing components are handled by the Visual Basic development environment and Setup Wizard in union.

4.3.4 Component Client

ActiveX technology and Visual Basic go hand in hand perfectly. It can be (almost) transparent to the client-application developer whether a specific functionality is inherent in the Visual Basic language or part of an external component. In fact many Visual Basic developers use COM components without being aware of it at all.

Visual Basic is able to use all kinds of components (both IDispatch and IUnknown based), automatically choosing the most efficient approach (see COT/3-6 3.1.4.1) for binding members.

Components are instantiated based on their ProgIDs not GUIDs. This is great for readability, but it may in certain situations be limiting for the client developer.

Visual Basic certainly prefers if the datatypes are limited to Automation types. Clients which communicate other types than those defined as Automation types can in some cases be communicated if necessary although in that case: There goes the *nice* integration!

Information available in the server's type-library is (automatically) used very intensely both for syntax checking, optimisation purposes and for providing help to the developer.

4.3.5 Development Environment

See section 4.3.1.

4.3.6 Development Process

Visual Basic's most prominent characteristic with respect to COM development is its simplicity. A lot of the features of COM - and especially its ActiveX incarnation - is designed so that they are easy to use from Visual Basic and so that they integrate nicely into Visual Basic.

It is trivial for a Visual Basic programmer to create a server. Most of the work involved is a simple matter of adding class modules and setting a few properties on the classes as well as on the server itself. If functionality in the server is wanted more work will of course be involved (!). But adding members to a class (and thus creating interfaces) is no different from writing standard Visual Basic procedures.

As the use of COM in Visual Basic is based on encapsulation and not on code-generation (see COT/3-6 1.3.1.2) it is simple to change created components (within the limitations imposed by Visual Basic) (see COT/3-6 1.4.5.1). E.g. changing an EXE server to be a DLL server is just a question of setting a simple property in Visual Basic's development environment. A setting which can literally be made in seconds (see COT/3-6 1.2.3). Furthermore Visual Basic checks that the changes are legal. i.e. an existing interface isn't changed when a new one should be created etc. (see COT/3-11 phase 1.2).

When making COM based abstractions both delegation and aggregation are possible implementation strategies, but it is not possible in Visual Basic to create servers which uses COM based aggregation (see COT/3-6 3.2.1), instead you have to use delegation.

4.3.7 Auxiliary

4.3.8 Overall Impression and Summary

All in all Visual Basic is probably one of the simplest tools to use for doing COM development. The process is easy and the COM approach integrates nice and consistently into standard Visual Basic programs. The weak points lies in the limitations when it comes to low-level COM technicalities.

Future versions of Visual Basic will no doubt open up for more low-level features of COM. Such new features of Visual Basic might be more complex to use. But they will probably be implemented as options only the experienced user has to worry about.

4.4 Summary of COM Support in Microsoft Visual C++ using the ATL Library

4.4.1 General

The programming language is C++. This language is known from many other platforms. The language is pretty good for OO modelling and the definition of COM interfaces is implemented conveniently as abstract superclasses in C++. The purpose of ATL is to be able to create efficient (on size and performance) and lightweight COM components. The components are highly configurable and flexible without having to implement the tedious low-level C++ COM code. The advantages and disadvantages can be seen in the following.

4.4.2 Component Server

The advantages can be listed as:

The configurability can be done on the following topics:

- Registration in the Registry
- Implementation of multiple interfaces created by either the programmer or other vendors.
- Many components in the same server.
- Support for all known Win32 threading models.
- Many target types (debug, Unicode, minimal dependency, minimal size).
- Framework for many different predefined components.

With ATL it is easy to create components that supports either custom- (vtables), dual-, and disp-interfaces. Automation is well supported both through wizards and manually programming. Many of the difficult COM types (like BSTR) can be encapsulated in native easy-to-use types. The server can support COM exceptions through the predefined `ISupportErrorInfo` interface. COM collections can easily be created.

When creating the out-proc servers VC++ automatically creates a project for a proxy/stub DLL. During creation you can choose whether or not to include the DLL in the code. The singleton pattern is supported through the `DECLARE_CLASSFACTORY_SINGLETON()` macro. Creating components that support bi-directional communication through Connection Points can be done if the syntax of IDL is not a problem for the programmer. There is a high degree of reuse of components by the concept of COM aggregation. This is highly integrated into ATL by some macros. There are also some interface optimisations, like tear-off interfaces, which is closely coupled to the COM aggregation concept.

The disadvantages can be listed as:

C++, multiple inheritance, virtual methods, and templates can be hard to learn.

ATL is a framework for creating COM components not for easy programming, meaning that there is no sandbox to play in as with Visual Basic and Delphi. You are pretty much left on your own or to adapt old libraries. There is support for MFC in server residing in DLL's.

The knowledge of IDL is a must. The programmer should know the mappings between COM types and native C++. Click-and-Create is almost only a forward matter. If you have done something you regret the only way to undo it is by changing the source code. Many of the settings for a component or a server have to be manipulated manually in the source code editor, but it is usual by adding or deleting a couple of macros. If a programmer in the development process wants to change from in-proc to out-proc server

implementation (or the opposite way), he has to start all over and copy the code from the old server to the new. When working with free threading the user is more or less left on his own when creating, destroying, communicating between, and synchronising the threads.

4.4.3 Compilation and Distribution

The targets for compilation is very configurable. You can optimise for speed and size. You can also choose debug and Unicode targets.

As default ATL assumes you don't use the C-Runtime library for the release versions of the components. This can for an inexperienced ATL programmer that is about to ship his component, give a lot of headache. If you use the C-Runtime library (i.e. by using the `#import` directive) you should delete the symbol `_ATL_MIN_CRT` from the defines-list. When creating a visual ActiveX control it is pretty hard to embed other controls in the new control.

The generated components is by default self-registrating in the Registry. The self registration can be manipulated as wished.

4.4.4 Component Client

When working as a client of a component there is a mechanism called COM smartpointers that ease the problems concerning COM reference count compared to the traditional pure C++ pointers. The new `#import` directive increases the usage of COM servers. If a typelibrary for a server changes during development the `#import` will discover the change and regenerate the files that describes the server for the client. The usual HRESULTs can be mapped into real C++ exceptions if this programming style is preferred.

4.4.5 Development Environment

ATL is integrated into the Visual C++ Developers Studio. The components in projects belonging to a workspace can be reached as normal C++ classes. The Class View can display normal classes and COM interfaces using special symbols. The environment contains many tools for viewing, debugging, browsing, and testing the projects.

4.4.6 Development Process

The Visual C++ IDE (Developers Studio) combined with the ATL library ease the C++ programmer's efforts in implementing COM objects. The Developers Studio has a lot of wizards that can generate many different kinds of COM objects. The generated code is ATL specific and very flexible. You can use the wizards to manipulate the GUIDs, CLSIDs, ProgIDs and many other aspects. Methods can be added on the fly (before compile time) by using the method wizard that is very well integrated into the Developers Studio. After using the code-generating wizard it is up to the programmer to manipulate all the aspects by hand. The generated C++ code is normal C++ code and because of that it can be used with many of the other tools that is integrated with

Developers Studio: Class View, Debugger, and Browser. Other developer's COM interfaces can fairly easily be reengineered by using a tool called **OLE-COM Object Viewer** that comes with Visual C++ 5.0.

4.4.7 Overall Impression and Summary

When using VC++ 5.0 and ATL it is possible to create very fast, very small, and very flexible COM components. The stuff that is possible in COM you can do with ATL. The few exceptions for this can be reached through API and/or plain C++. The problem about ATL is that it is only COM support so some auxiliary class libraries should implement other aspects as databases or done through plain API programming. The older MFC libraries can be used from In-Proc ATL COM components but increases the size of the components and some extra work has to be done.

5. Conclusion

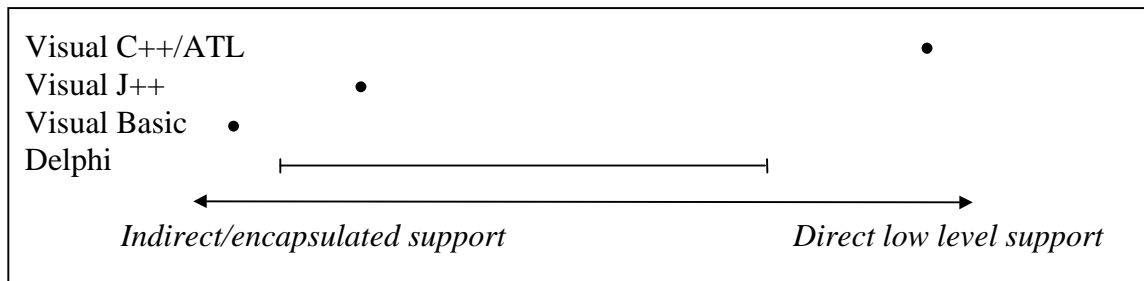
When dealing with COM development many criteria can be applied to compare different tools. We have in the following applied three criteria:

- directness of COM support
- COM integration and support in IDE
- size of generated components

After these criteria have been applied to the evaluation of the four tools the efficiency of each tool is evaluated as a function of the COM-complexity of the components to be generated.

5.1 Directness of COM support

This criteria specifies whether the support for COM-functionality in the tool is direct in the sense that it corresponds to the (low) level of the COM API functions.



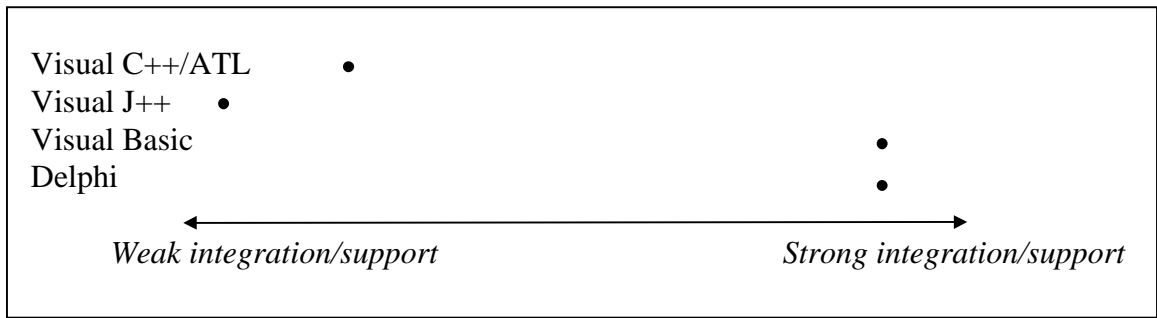
In general terms the more indirect/encapsulated the support for COM is; the easier the access for the COM-functionality is for the programmer - especially for the inexperienced programmer. The more direct low level support in the tool, the more flexible and complete is the access for the COM-support on the cost of higher complexity in the access.

As can be seen on the graph Delphi is capable of providing different levels of support: From rather encapsulated to pretty direct support.

Appendix A Section 6.1 explains the rating given the four tools with respect to directness of COM-support.

5.2 COM Integration and Support in IDE

The programming environment is very important for the development and maintenance of COM servers and clients. The following graph rates the ease of use of each IDE (Interactive Development Environment) when working with COM servers and clients.

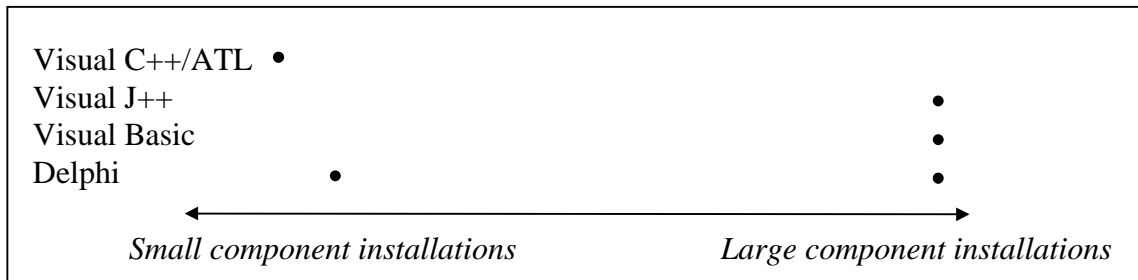


The stronger the integration and support for COM-development is in the IDE the more efficient (i.e. it takes shorter time to develop a COM component) the tool is when developing COM clients and servers.

Appendix A Section 6.2 explains the rating given the four tools with respect to strength of COM integration/support in the four tools.

5.3 Size of Component Installations

The size of COM component installations generated in the four tools vary and this is important for the space and time complexity of applications using these components.



Visual J++ and Visual Basic generally produce larger component installations because of the run-time DLLs *required* by these components. On the other hand these run-time DLLs include a lot of functionality which the programmer otherwise would have to produce/include in the component if needed. Delphi can produce components with or without run-time DLLs (hence the two dots in the evaluation of Delphi). The most compact component installations can be produced using Visual C++/ATL.

When discussing the effect of using run-time DLLs one also has to consider that a run-time DLL only has to be installed once in a multiple component installation, so the above figure is valid only for single component installations.

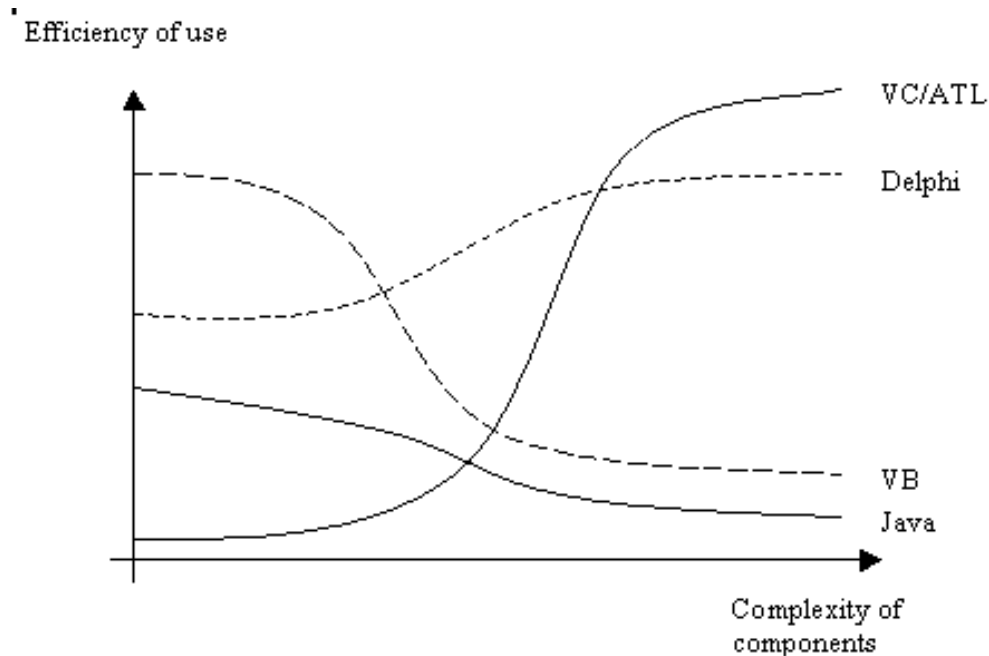
Furthermore a component's dependency on other components (e.g. the run-time DLLs) can both be an advantage (splitting applications into independently updateable bites) as well as source to problems (components getting out of sync etc.).

Appendix A Section 6.3 explains in more detail the rating given the four tools with respect to size of generated components in the four tools.

5.4 Efficiency of Use

Finally what in most cases decide which development environment to choose for COM development (given that you do not already have a preference for one of the programming languages) is how efficient the tool will do the job (develop the applications/components that is).

In the next graph we have summarised the efficiency of using the four tools as a function of the complexity of the components that must be produced.



As can be seen Visual Basic is the most efficient tool when working with components with low COM-complexity because of the encapsulation of COM-support (which hides many of the technicalities of COM) and the good integrated support in the tool for COM development. But when creating objects which are very specialised with respect to COM functionality it fails. Visual Basic is for the inexperienced COM programmer as Velcro is for kids: very easy to use. Visual C++/ATL is difficult - especially at the beginning - because lots of COM knowledge is assumed and a substantial amount of work is needed to produce even simple components. Visual C++/ATL wins when the components get highly complex because the high level of flexibility supports this complexity. For Delphi the support for low complexity component development is almost as Visual Basic, but the flexibility of the language lets it produce very specialised objects too. It has almost all the advantages from both VC/ATL and Visual Basic. The Java components can be made pretty easy but the lack of flexibility in the tool with respect to COM development does not leave Java many chances at the current moment for being chosen as the tool for COM development.

6. Appendix A

6.1 Rating on Directness of COM Support

COM spans lots of functionality and services. The following table lists the basic functionality that can be required for a programming language. The answers in the table show how well and precise the programming language is able of performing the task.

The tasks are:

1. Is it possible to manipulate the IDL directly? This is important for optimisation purposes.
2. Can you create components that support pure custom interfaces (vtable only) ?
3. Can you make a Custom Marshaller DLL (proxy/stub DLL) for the custom interface for an Out-Proc server.
4. Is it possible to create a server component that support only the IDispatch interface (no custom interface!) ?
5. Is it possible to create a server component that support both the IDispatch interface and a custom interface (a dual interface)?
6. Is the special COM surrogate inheritance mechanism COM Aggregation supported when developing servers
7. Is it possible to control all the aspects about GUIDs (both IIDs and CLSIDs), ProgIDs (version dependant and independent), Threading Models, and so on.

Precision of Min. COM support	Delphi	Java	VB 5.0	VC 5.0/ATL
Direct manipulation of IDL	√	(√)	÷	√
Pure custom interfaces (vtable only)	√	÷	÷	√
Custom marshalling of Custom Interfaces	(√)	÷	÷	√
Pure Dispatch Interfaces	√	?	÷	√
Dual (Dispatch + Custom) Interfaces	√	?	√	√
COM Aggregation	(√)	(√)	÷	√
Flexibility in self-registration	÷(?)	÷	÷	√

6.2 Rating on Support and Integration in IDE

The programming environment is very important for the development and maintenance of COM servers and clients. The following questions reveal the ease of use when working with COM servers and clients.

1. Does the tool support the concept similar to IntelliSense as defined by Microsoft. IntelliSense give visual clues of types of methods and parameters in the editor of the IDE

2. Does the tool support Binary Compatibility, meaning it automatically checks the interfaces the programmer is currently working with to see if it is similar to already predefined interfaces
3. Are the sourcecode debugging of the components fully integrated in the IDE for both servers and clients (both in-proc and out-proc)
4. When distributing the components does the IDE give you direct support for making installation packages?

Programming Environment	Delphi	Java	VB 5.0	VC 5.0/ATL
IntelliSense	√	÷	√	÷
Binary Compatibility	÷	÷	√	÷
Debugging of Component	√	÷	√	√
Support for installation	(√)	÷	√	÷

6.3 Rating on Size of Generated Components

The generated component can vary in size because of different strategies for deployment. Some tools support only self-contained objects (Self), some support components that has to distribute a big runtime system (RS) too, and some support both (Both). The self-contained objects can vary in size too depending on the tool

Binary Components	Delphi	Java	VB 5.0	VC 5.0/ATL
Strategy for deployment	Both	RS	RS	Self

Both Java and VB create very small components. The drawback is that these components have to be executed in the context of a runtime system. If the runtime system is installed once new components will gain from this. Delphi supports both mechanisms and the self-contained components are pretty small. ATL creates only very small self-contained components (except for a registrar that can be shared).