# USING MICROSOFT'S COMPONENT OBJECT MODEL (COM) TO IMPROVE REAL-TIME DISPLAY DEVELOPMENT FOR THE ADVANCED DATA ACQUISITION AND PROCESSING SYSTEM (ADAPS)

**Katherine Rodittis**
**Air Force**
**Flight Test Center**
**Range Division**
**Edwards Air Force Base**

**Patrick Mattingly**
**Air Force Flight Test Center**
**Symvionics, Inc.**
**Edwards Air Force Base**

## ABSTRACT

Microsoft's Component Object Model (COM) allows us to rapidly develop display and analysis features for the Advanced Data Acquisition and Processing System (ADAPS).

## KEY WORDS

Flight Test, Real-time, Display, Microsoft, COM, Analysis, Software Development, ADAPS, IADS, Word, Excel

## INTRODUCTION

The landscape of real-time display requirements changes so rapidly that traditional software development practices cannot keep pace with the demands and as a consequence the flight test engineer often has to either sacrifice requirements or compromise the test schedule. Many real-time systems depend upon a custom-built or off-the-shelf display package that requires extensive work by an experienced programmer to integrate new display objects and features into the existing system. Oftentimes even if the display object's code is taken from another package or application, a programmer must entirely re-write the new object in order to integrate it into the existing system.

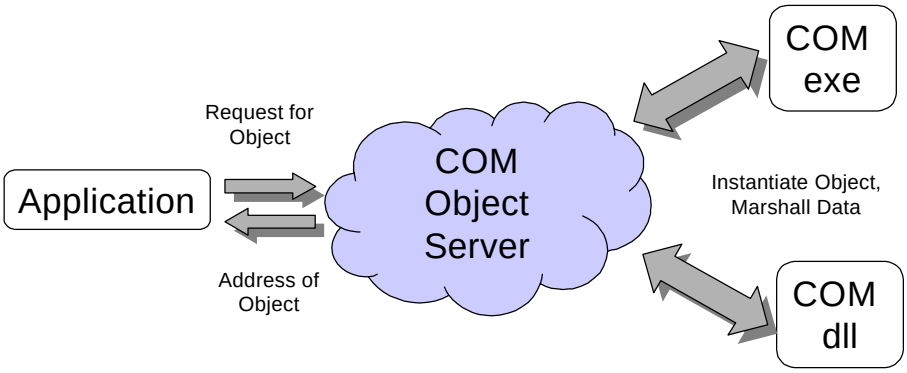
By using Microsoft's Component Object Model (COM) we hope to eliminate much of this work. COM allows the developer to integrate new objects wholesale into existing applications without re-compiling or re-linking the objects or the applications. We hope to exploit this proven technology to establish a standard COM interface that can be used to assimilate the objects from many off-the-shelf packages or freeware objects that are available from the Internet or even display objects built at other ranges. Through the use of COM, we have developed a method whereby inexperienced programmers or even non-programmers can change the functionality of our Interactive Analysis and Display System (IADS) via a standard scripting language, VBScript. As an added bonus, COM allows us to integrate our data and displays into popular Microsoft products such as Word and Excel.

In the end we will have a robust system that can accommodate even intricate display objects on short notice by a small staff; thus adding flexibility to a flight test program without compromising schedule.

## HOW COM WORKS IN A NUTSHELL

The heart of COM is to build reusable objects that an application can access, regardless of what machine the object resides on or what language the object was written in and without re-compiling or re-linking. The application makes a request to an object server, possibly on a remote machine, and, providing the operating system security requirements are met, the object server loads the object into memory. Traditionally this type of dynamic binding was accomplished through shared libraries that were essentially loaded and linked into the application at runtime. But these libraries were generally bound to the language they were written and compiled in and had to reside either on the local machine or on the mounted directory of another machine. Likewise, regardless of the location of the object, the actual execution would be on the local machine. If the object really needed to execute on the remote machine, a custom piece of server software had to be written for each object. COM allows you to incorporate the strengths of different languages into your application and makes it easier to create distributed applications by handling the traditional client/server portion for you.

How COM Works




**BUILDING THE FASTER MOUSETRAP: TAKING ADVANTAGE OF THE WEB, COTS PRODUCTS AND SCRIPTING TO DECREASE DEVELOPMENT TIME WITHOUT SACRIFICING APPLICATION SPEED AND INTEGRITY**

Since the COM server handles the data marshalling between your application and the COM object you're looking to incorporate, it is mostly irrelevant what language that object was written in and thus objects can be truly encapsulated for re-use. Microsoft's ActiveX is a standard set of COM interfaces that covers many of the GUI operations one would find in a display object without actually dictating how that object is written or what graphics language it employs. Many freeware and shareware objects are written in Visual Basic and use ActiveX as the COM Interface. Once you have implemented ActiveX into your real-time application, you can download and use existing objects from the Internet almost immediately. Likewise, many COTS products such as DataViews and Virtual Prototypes (VAPS) already support ActiveX in their products. Legacy code would still need a COM wrapper but once that was written other applications could take advantage of the code without re-compiling or re-linking. This freedom of language also allows the developer to identify which pieces of the application are time-critical and therefore must be written in a "faster" language, such as C++.

Any software development effort is under some kind of schedule constraint. If too many pieces of the application depend on having a senior programmer available to write them, then the overall schedule invariably starts to slip. Not only can this hold up the delivery of the final product but it can also lead to programmer burn-out and increased personnel turnover. Ideally one would shift some of the workload

across the combined development and operational teams. By providing an interface to "simple" scripting languages like VBScript, JavaScript and HTML, COM allows the development work to be spread across many more people. Also, integrating scripting languages into your application gives on-site support personnel or the flight test engineers the ability to customize the overall functionality without requiring an entirely new release from the development team. For example, a structures flight test engineer might decide that for a given test, one particular filter needs to be run when Event A is triggered and another when Event B is triggered. The development team has already built the computational-intensive filters in C++ as a COM object but the flight-test engineer added the scripting necessary to glue the filters to the events of that particular test, possibly right before the test.

## DEVELOPING A DISPLAY DEFINTION STANDARD UNDER COM

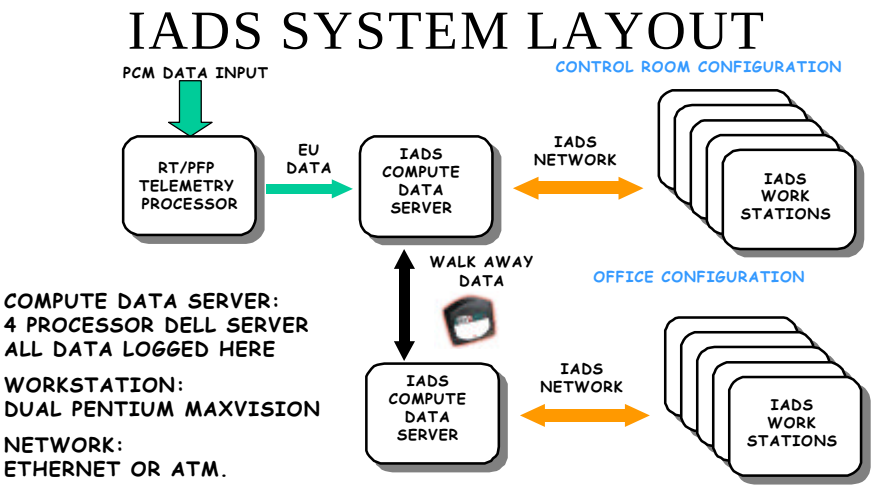
Most proposals for display definition standards revolve around some kind of text file that contains metadata about the display object. The biggest downside to these suggestions is that the object being "imported" doesn't come with any executable code. The system it's being imported into must already support the object and the job of the text file is just to define the specific attributes for a particular instance. But each COM object is both self-executing and self-describing. Each COM object contains the runtime type information that describes the methods and member variables available to the calling application. Object-specific properties are stored in Property Bags that are also self-describing. All that is required for a cross-application standard is an agreement about what will appear in the property bags, an agreement on what the automation interface to the ActiveX controls will be and a COM interface class to the file that stores all the information. Someday flight test engineers will take their entire real-time setup – displays, derived parameters, special processing and all – from range to range as they test in different places.

## BETTER VERSIONING

Another challenge for any development group is maintaining backwards compatibility between versions and regressively testing that compatibility. Most system administrators can recount horror stories of installing the latest version of one application and subsequently "breaking" older applications that depended upon the same shared libraries. Microsoft Windows® is particularly prone to this problem with so many applications installing their own versions of common Dynamic Link Libraries (DLLs). COM requires the programmer to maintain the interface for the COM object once it has been registered. That is, the programmer cannot change the footprint of any method and furthermore no overloading of methods is allowed. So any changes must be handled by interfaces that derive from the initial interface and thus have new names. Likewise, versioning information is registered so that applications requesting older versions of the object will receive the correct instance of that object. All of this makes regression testing quicker and easier because there is less likelihood of older code "breaking" and when it does "break", it is easier to trap and correct.

## A BRIEF DESCRIPTION OF IADS[1]

IADS is one of the real-time systems under the ADAPS program at Edwards Air Force Base. It is a PC-based display and analysis system that is used primarily for flutter-structures flight testing. The flight test data travels via Fast Ethernet from the L-3 Communications' (L3COM) O/S90™ Telemetry Pre-Processor (TPP) into a quad-450 MHz Pentium III server called the Compute Data Server (CDS). From there, the CDS serves up to 14 client workstations that are dual-450 Pentium III PCs over an ATM network.




In addition to serving the real-time data streaming in, the CDS persists all that data to disk and so allows the users to scroll back data on their individual workstations. The CDS is responsible for any bulk processing or monitoring such as nulling and threshold checking that might apply to all parameters coming in. The configuration file for the mission encompasses all the defined displays, the derived parameters and many other settings and is managed by the CDS so that users can add to it during the mission. Likewise, the results of different computational analyses performed on the client workstations are also saved during the mission into the configuration file. The client workstation is responsible not just for displaying the data in various graphical objects but also for applying many computationally intensive algorithms such as digital filters, user-defined transcendental functions for derived parameters and various other analysis procedures.

[1] For more information about IADS, please see "INTERACTIVE ANALYSIS AND DISPLAY SYSTEM (IADS) TO SUPPORT LOADS/FLUTTER TESTING", presented in Proceedings of The International Telemetering Conference, October 1999.

## HOW WE USE COM IN IADS

Our implementation of COM in IADS focuses on the following interfaces:

**Data Source Interface**
This interface defines the interaction between the CDS and the front-end data source. Although right now we support only the L3COM OS/90 and the L3COM 550, the list of available front-ends will expand in the future.

**Standard Data Interface**
This interface controls any client requesting and receiving data from the CDS or any data source that derives from the data source interface. Data Analysis Procedures (DAPs) adhere to this interface and so do any Compute Data Servers that are resident on the network.

**Display Interface**
This interface implements ActiveX so that any ActiveX control can be dropped into the user's display object palette with minimal programmer effort. DataViews and VAPS controls have been integrated into IADS this way.

**Parameter Builder Interface**
This interface governs user-defined derived parameters and analysis objects.

**Scripting Interface**
Our plan is to embrace VBScript as the "glue" for all the components on the user's display. This way many last-minute flight test requirements can be handled in a very short time.

**Bulk Parameter Interface**
This interface controls any type of processing that must be applied to all or most of the parameters in the incoming data stream.

**Configuration Interface**
This interface helps the CDS manage the previously described configuration file.

## WHY WE CHOSE COM

Besides all the great features of COM described in this paper so far, there are some other compelling reasons to implement COM in your real-time data monitoring system.

COM is the de facto object broker under Windows. Based on the fact that our users are running Windows NT on the PCs in their office, we as an organization have already decided to move the client side of our real-time support to Windows NT, which means we don't have to pay any additional costs for object brokers. Also, our display objects can be easily integrated into popular Microsoft Office products like Microsoft Word and Microsoft Excel, which is another huge benefit for our users. If we decide in the future to also deploy Linux on our client workstations or our CDS, COM support already exists from third party vendors.

Unlike CORBA objects, COM objects can be written to run “inproc”; that is, the COM object can operate within the same process as the calling client application, which is a tremendous performance advantage.

## CONCLUSION

By implementing COM, we will decrease the time and effort it takes to write software while simultaneously increasing the robustness of our real-time system.