SLAC-PUB-9592
November 2002

Bringing COM Technology to Alignment Software[*]

Lothar Langer
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309, USA

Abstract

Software tools for alignment purposes should be versatile and flexible handling data in various formats. Microsoft ‘s Component Object Model is the base for an appopriate software architecture to be built upon. COM-components comply with different programming environments like web applications, C++, Visual Basic, and MATLAB. The benefits for the user and the program developer are discussed.

Contributed to 7th International Workshop on Accelerator Alignment (IWAA2002)

Spring-8/JASRI, Sayo-gun, Hyogo, Japan, November 11-13, 2002

[*] Work supported by Department of Energy contract DE–AC03–76SF00515.

Bringing COM Technology to Alignment Software*

Lothar Langer

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309, USA

1 Introduction

The Component Object Model (COM) is an application programming interface (API) that allows for creation of so called 'objects'. These objects are software constructs to be useful for the service they provide for a client. The client may be another piece of software, a conventional application program or possibly another 'object'. Objects are packaged in 'components'. Here 'component' means the (more physical) representation of software – a data file on some storage medium.

If you boil it down to a level of software using software – what is special about COM (except being originated by Microsoft) ? There are already successful concepts like libraries, class libraries, dynamic link libraries and the like. To make these software pieces work together you need 'interfaces' which are basically contracts about how to give and how to take. Either side has to comply to these contracts. But - writing computer programs comes with a bunch of different languages on different hardware and software platforms. To link to a libraray is very dependent on language and platform. It is about compatibility of data types and calling conventions. That is why libraries are reused mostly inside the world defined by a computer language or a platform boundary.

But software still is "only" bits and bytes. Here contracts (interfaces) specified on a binary level – that sounds like a cute way to go. COM is just a binary specification. It doesn't tell you how to build components. It tells you how to connect to a COM object. That makes it language independent - and platform independent too (at least in theory). There should be some software vendors offering COM on Unix.

COM has competition, in terms of technoloy. The Common Object Request Broker Architecture (CORBA) answers the same questions as COM. But it has not reached the same importance.

Around COM exists a bundle of higher-level technologies, heavily promoted by Microsoft. Most famous perhaps is ActiveX that has been designed for use in web applications and many other kinds of 'containers'.

* Work supported by Department of Energy contract DE-AC03-76SF00515.

2 The Component Object Model (COM) and ActiveX Technology

Objects and Interfaces

COM objects are well encapsulated. You cannot gain access to the internal implementation of the object. That means for example you have no way of knowing what data structures the object might be using.

Objects 'live' through interfaces. An interface is the object's point of contact to the outside world. The outside world in terms of COM are various application programs. They act as clients asking the server object for some kind of service. All what a client gets from COM is a pointer to the interface.

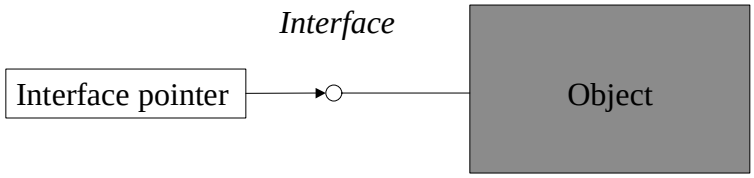

Figure 1 COM objects are 'black boxes' with interfaces as their only points of access.

An interface is some data structure that sits between the client's code and the object's implementation through which the client requests the object's services. Actually the interface pointer is a pointer to a pointer to an array of pointers to the functions in the interface.

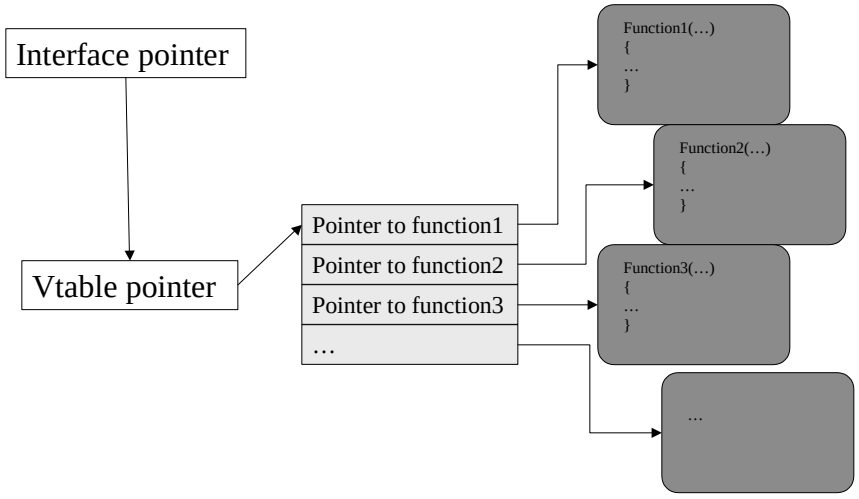

Figure 2 The calling mechanism referred to as the 'binary standard' makes language independence for COM object implementation possible.

The binary standard for a COM interface means that an object creates a vtable (virtual function table) that contains pointers to the implementations of the interface member functions and allocates a structure in which the first 32 bits are a pointer to that vtable. The client 's pointer to the interface is a pointer to the pointer to the vtable.

Function tables can be created 'manually' in a C application or almost 'automatically' with C++ (and other object oriented languages that support COM).

Interface characteristics

In order to avoid name collisions interfaces are uniquely identified through a 16-Byte value called a GUID (Globally Unique ID)[1].

In binary terms, a GUID is a data structure defined as follows, where DWORD is 32-bits, WORD is 16-bits, and BYTE is 8-bits:

[1] "The term GUID … is completely synonymous and interchangeable with the term "UUID" as used by the DCE RPC architecture …The GUID design allows for coexistence of several different allocation technologies, but the one by far most commonly used incorporates a 48-bit machine unique identifier together with the current UTC time and some persistent backing store to guard against retrograde clock motion. It is in theory capable of allocating GUIDs at a rate of 10,000,000 per second per machine for the next 3240 years, enough for most purposes." See p. 69 [COM 95]

```
typedef struct GUID {
    DWORD  Data1;
    WORD   Data2;
    WORD   Data3;
    BYTE   Data4[8];
} GUID;
```

One object can support multiple interfaces. You can have and in many cases will have multiple interfaces on each type of object. As interfaces evolve over time changes in definition or semantics of functions should result in a new identifier meaning a new, additional interface on the object. Thus versioning problems are avoided.

Interfaces - kind of object oriented

When dealing with COM objects the client always handles interfaces of objects, never the object itself. Yet there are some object oriented features to interfaces.

Inheritance is possible among hierarchicallly derived interfaces, but only for function definitions, implementations have always to be provided.

Interfaces are polymorphic through the special interface *IUnknown*.

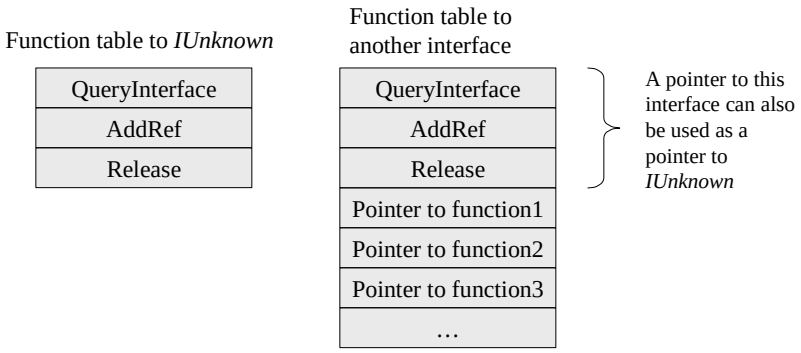

Figure 3 The three basic functions of *IUnknown* are inherited to every COM interface.

Base Interface *IUnknown*

"*IUnknown* is the label of the interface that represents the functionality of an object when all else about that object is unknown."[2] *IUnknown* provides the most essential services with only three functions.

- *QueryInterface* provides navigation through the interfaces of an object. Always a pointer to *IUnknown* can at least be obtained.
- *AddRef* and *Release* provide lifecycle management for an object through reference counting. That means an object is kept 'alive' in memory as long as there exists at least one reference to one of its interfaces. When the last interface has been released the object will go out of service. It will then be cleared from memory.

Clients and Servers

The concept of client/server interaction does not only mean a client program (EXE module) using a server object (DLL module). Objects may exist inside executable programs. And one object (DLL) can also be using another object (DLL or EXE).

COM 's idea of *location transparency* is that clients and servers never need to know how far apart they actually are, that is, whether they are in the same process, different processes, or different machines. COM provides transparent access to local and remote servers through proxy and stub objects.

If the client and server are in the same process, the sharing of data between the two is simple. However, when the server process is separate from the client process, as in a local or remote server, COM must format and bundle the data in order to share it. This process of preparing the data is called *marshalling*. Marshalling is accomplished through a "proxy" object and a "stub" object that handle the cross-process communication details for any particular interface (depicted in Figure 4). COM creates the "stub" in the object's server process and has the stub manage the real interface pointer. COM then creates the "proxy" in the client's process, and connects it to the stub. The proxy then supplies the interface pointer to the client.

[2] See p 65 [COM 95]

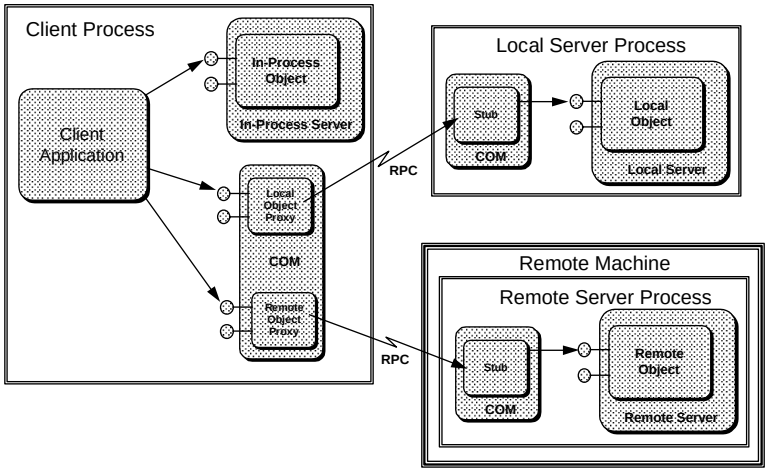

Figure 4 Three different situations of location transparency. The client application does not have to care about where the server object is actually running.

As mentioned in the introduction COM is an application programming interface (API) providing vital services for the functioning of client/server interaction. But the presence of the COM Runtime Library in memory is “only” needed for object creation and passing the first interface pointer to the client. Pointers to other interfaces on the same object can be obtained through the above mentioned *QueryInterface* function. By inheritance through *IUnknown* every interface posseses such service.

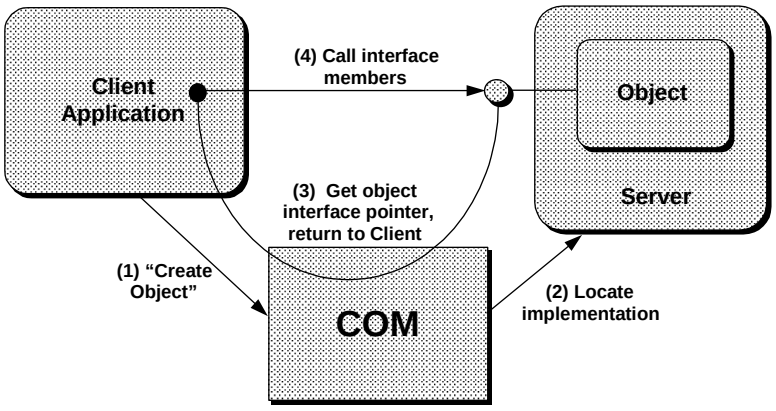

Figure 5 The COM runtime library basically initializes the client / object communication and then drops out of the scene to reduce the overhead.

The COM Runtime Library uses a class identifier to provide 'implementation locator' services to clients. A client need only tell COM the class identifier it wants and the type of server—in-process, local, or remote—that it allows COM to load or launch. COM, in turn, locates the implementation of that class and establishes a connection between it and the client.

COM classes

According to the paradigm of object orientation there also is a concept of classes of objects. COM classes give all the information that is needed to create a COM object. Because there are no details of implementation specified in COM the only thing COM needs to know about the class of an object is where to find it. So there exist class identifiers similar to the above mentioned interface identifiers. COM classes are globally uniquely identified by GUIDs – and not by their names. So providers of COM objects can use identical names for their object classes and interfaces. Still they can be kept apart by COM. They will have different GUIDs because of the way these identifiers are generated.

How does COM know about all the object classes and interfaces?

The information about available object classes and interfaces on a given system is centralized in the Windows registry in case of Win32 platforms. With help of this

‘database’ for systemwide information COM can do the mapping from component name to GUID and finally to the file where the implementation code resides.

ActiveX Technologies

ActiveX is Microsoft ‘s marketing name for a bundle of technologies that are built on COM. Very popular are ActiveX controls. These are components which provide all kinds of visual support and effects in a context of graphical user interfaces (GUI). They also can be applied in HTML document through the <OBJECT> tag.[3]

ActiveX has the power to integrate technologies that are useful on the web. As a consequence you can see more and more scripting languages offer COM support. Besides Microsoft ‘s VBScript and JScript there is PERL, a very popular ‘web language’, that offers ActiveX support.[4]

3 Building Tools for WinGEONET

WinGEONET is built using Microsoft Visual Basic (VB). It handles a number of file formats in which data are stored during various steps of processing. Here the use of COM objects brings a lot of simplification and enhancement.

Visual Basic environment

Visual Basic is particularly COM friendly. If you try to call a conventional DLL (Dynamic Link Library) written in C or C++ from within Visual Basic you will know that it is a job with very unsatisfying results when you pass parameters to a C/C++ function. Everything is at ease when you build an ActiveX component in C/C++ and use it from VB as the ActiveX client. One reason is VB ‘s support for using ActiveX objects. Another reason is VB ‘s preference for the ActiveX friendly data type VARIANT. Also building ActiveX components from Visual Basic has got a standard job for VB programmers.

[3] See Objects, Images, and Applets in [HTML4]

[4] See [PERL]

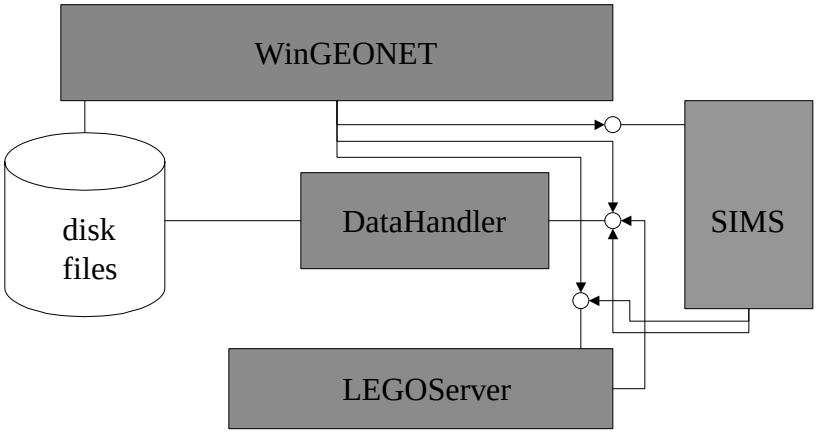

Figure 6 Component design promises numerous benefits concerning practicability and performance for a toolbox like WinGEONET.

Data Availability from different sources

WinGEONET collects and processes data from various sources in different formats. Raw data may come out of ASCII files which are formatted in at least 3 different ways. Data may have to be read from an Excel spreadsheet or a relational database. This kind of job happens almost all the time. Various applications have to import data before they can process it in some way. It is obvious that handling data storage and making data available are tasks that can be centralized in one separate component. This component is called ‘DataHandler’. It has three kinds of objects.

The DataConnection object provides services needed to make geodetic data available out of diverse structures and formats. It comes in a close relation to a second object, namely DataItem.

The DataItem object is a container for raw data items. A DataItem object is basically a hash table where the values are VARIANTs. The value can thus be another DataItem object. DataItem objects are extendable and very flexible.

A third object, named DataItemViewer, provides just a basic view on the tree structure of DataItem containers.

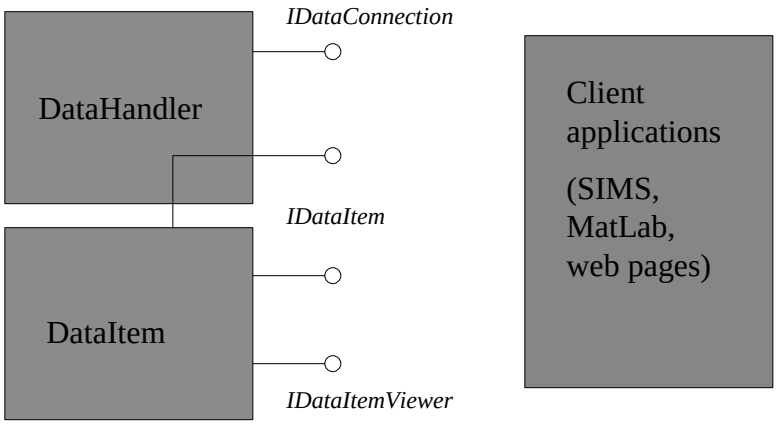

Figure 7 The DataHandler object makes data available in a container object called DataItem.

Mathematical Computations

Math routines is another issue for reuse from various clients.

For WinGEONET the math package LEGO is vital.The connection to LEGO has been a difficulty because LEGO is written in C++. LEGO used to read an input file, do computations on the data, and write results into an output file.

LegoServer is LEGO in a COM object. That is why it can accept data from a Visual Basic client in WinGEONET and do calculations. The results can be collected by the same or a different client, for instance SIMS. Thus the intermediate steps of creating input and output files can be passed. If necessary for persistence purposes data in various formats can be stored using *DataHandler*.

LegoServer knows a bunch of COM objects and their respective interfaces (*ISta*, *ITar*, *IMes*, *IResult*). But they are not creatable from outside. The client can only create and use them through an interface called *IProject*.

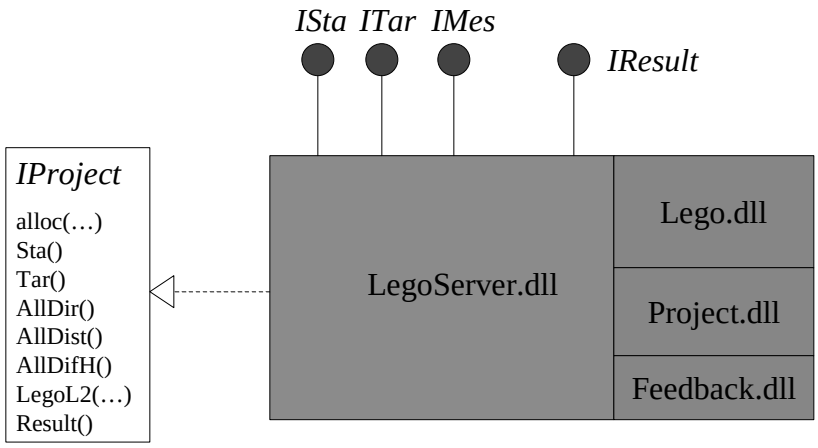

Figure 8 LEGOServer handles LEGO ‘s computational services through function calls into the three ‘traditional’ DLLs.

Graphical Presentation Tools

Sims has got ActiveX server capabilities. It is possible to call some document-related services (open, close, and import of data into a document) from an ActiveX controller like VB, for instance.

More integration with more COM

In the near future there will be more integration through the ‘Data Availability’ services which will be extended by data persistence and shared containers.

4 SIMS & LegoServer

SIMS has originally been an application program for visualization and simulation of geodetic data. Since it is integratied into WinGEONET it is on its way to a generalized tool for graphical presentation. SIMS is able of displaying geodetic networks in different views from a single set of data in a document file.

For displaying error ellipses in the different views it uses LegoServer (see Figure 8). In order to get reasonable response behaviour in the viewing window the calculation step is performed in a separate execution thread.

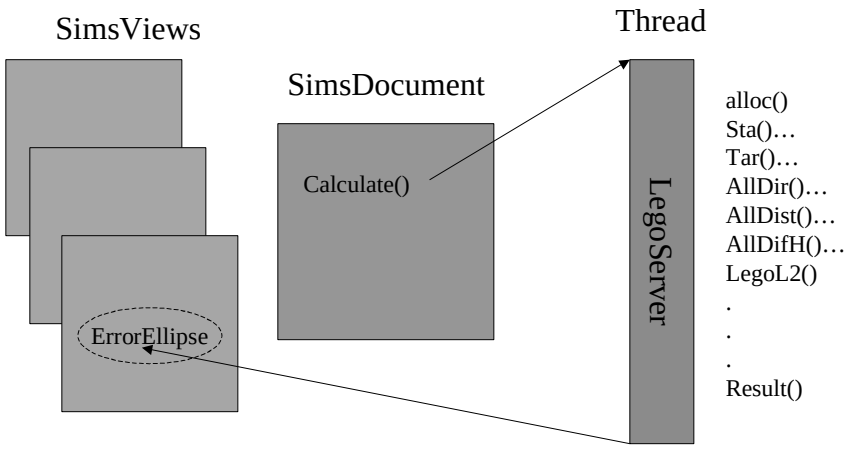

Figure 9 LEGOServer handles LEGO ‘s computational services through function calls into the three ‘traditional’ DLLs.

5 Reusing ActiveX components in MATLAB

MATLAB is a powerful language for ‘technical computing’. It is based on calculations with matrices which are multidimensional arrays of numbers. MATLAB has an elaborated graphical user interface (GUI). A variety of specialized ‘toolboxes’ can be added to MATLAB.

MATLAB environment

The MATLAB language can be extended by user defined functions. Functions reside in M-files with the function ‘s name. They can be called with a list of input arguments and they can return a list of output arguments. A MATLAB function can call any other MATLAB function that is in the search path.

Using MATLAB ‘s function concept powerful applications can be programmed involving own GUIs.

ActiveX connection in MATLAB

MATLAB supports ActiveX among other external interfaces (to Java classes, C and FORTRAN).

There is ActiveX client support. MATLAB can create ActiveX controls or server objects and handle there respective interfaces. Event handling is possible only for ActiveX controls.

There is also ActiveX server support. MATLAB itself exposes interfaces for ActiveX clients (MATLAB being the server). A client can invoke a dedicated MATLAB ActiveX server, a MATLAB instance that only serves him. Or it can share a MATLAB ActiveX server instance with other clients.

Example: MATLAB Utilities

MATLAB Utilities is a collection of applications combining services of MATLAB and other COM objects.

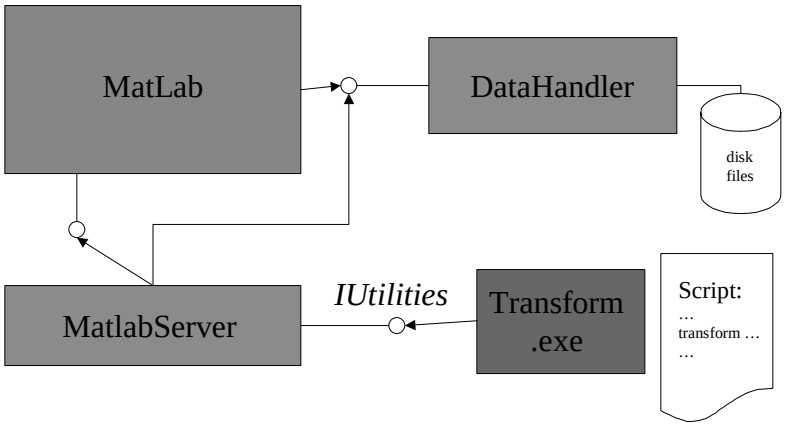

Figure 10 An application involving MATLAB and COM objects.

The ‘transform’ command is a service available through the *IUtilities* interface of a COM object named ‘MatlabServer’. *MatlabServer* is a windowless server component (EXE file) and had been created for the purpose of centralizing several services which are using the MATLAB engine. Here *MatlabServer* uses *DataHandler* (see Fig 7) to get data out of disk files and it uses the MATLAB engine to calculate the transformation.. Finally *MatlabServer* lets *DataHandler* store the results in a different set of disk files.

In another utility a *DataHandler* object is used directly by MATLAB to retrieve data from files in order to perform graphical evaluation.

6 Example of a presentation on the Web

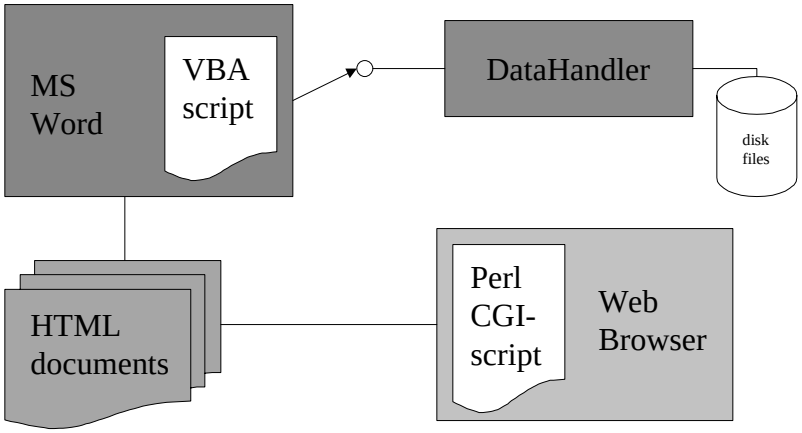

Figure 11 Example of a web presentation using the COM object DataHandler.

This application involves two steps because of performance reasons.

- A VBA (Visual Basic for Applications) script that runs inside MS Word retrieves data out of disk files via a *DataHandler* object. Then it inserts the data into predefined locations in a form document which is generated from a document template. The resulting MS Word documents are stored as HTML files.
- For the presentation a CGI (Common Gateway Interface) script written in PERL provides the web browser with the appropriate HTML.

7 Conclusions

A software toolbox gains flexibility by component design. Tools can be programmed to be very versatile by centralizing often reused functions. For the WIN32 platform the Component Object Model would be the concept of choice for component design.

When agreeing on the basic design and coordinating the definition of interfaces a small team of developers can create software components in a relatively independent way. They can use different implementation languages.

8 Acknoledgements

The author feels indebted to Catherine LeCocq as the supervisor for the Alignment Engineering Group for her essential support in the development process concerning especially component design. Thank you also to all other members of the Alignment Engineering Group at SLAC for their readiness to help.

9 Bibliography

[Brock 95] Brockschmidt, Kraig. Inside OLE, 2nd edition, Microsoft Press, 1995

[COM 95] Microsoft Corporation. The Component Object Model Specification, Version 0.9, October 24, 1995 [online]. Available WWW <URL: http://www.microsoft.com/oledev /> (1995).

[HTML4] World Wide Web Consortium HTML 4.01 Specification, 1999 Available WWW <URL: http://www.w3.org/TR/REC-html40 />.

[PERL] O'Reilly Perl.Com Downloading the Latest Version of Perl, 2002 Available WWW <URL: http://www.perl.com/pub/a/language/info/software.html />.