ActiveX - an internet strategy for applications development

Rosca, Doina and Banica, Logica

University of Craiova, University of Pitesti, Faculty of Economics

April 2007

ActiveX - AN INTERNET STRATEGY FOR APPLICATIONS DEVELOPMENT

Doina Roşca, University of Craiova, 200585 Craiova, str. A. I. Cuza, nr. 13, tel.: 0721632019,
rosca2na@yahoo.com
Logica Bănică, University of Pitesti, 11040 Pitesti, str. Targul din Vale, nr.1, tel.: 0745227774,
olga.banica@upit.ro

Abstract

ActiveX is a set of technologies that has the potential to change the way information is accessed and used on the Internet; allows software components to interact with one another in a networked environment, regardless of the language in which the components were created. Powerful abstractions based on OLE have been developed to enable fast, scaleable integration of your objects within the Internet. Microsoft is making a major effort to make the Internet everything it can possibly be. By using ActiveX, developers can make the best use of their system resources while providing instant, dynamic content and functionality in their Internet applications. How information is presented greatly affects how interesting and usable people find it.

Keywords: Internet, ActiveX, OLE, COM

Introduction

Microsoft has unveiled an extensive new solution technology for the Internet called *ActiveX*. Microsoft ActiveX is a broad and powerful abstraction for Microsoft Internet Solutions. Content providers and Internet application developers now have a robust and extensible frameworks that enables them to develop a new generation of Internet applications. ActiveX started out as an Internet strategy. It now covers all aspects of OLE/COM/Internet development.

Microsoft introduced the term *ActiveX* at the Internet Professional Developers Conference (Internet PDC) in March 1996 and referred to the conference slogan "Activate the Internet".

ActiveX is the new corporate slogan of Microsoft--similar to the term OLE in the early 1990s and in a very short time, has come to mean much more than "Activate the Internet."

ActiveX has become the all-encompassing term used to define everything from Web pages to OLE (Object Linking and Embedding) Controls. It has come to signify, on one hand, small, fast, reusable components that can get you hooked into all the latest technologies coming out of Microsoft, the Internet, and the industry. On the other hand, ActiveX represents Internet and applications integration strategies. These days, products and companies that don't have ActiveX and Internet somewhere in their nomenclature are considered, both internally and externally, as being behind the times.

Why use ActiveX? With ActiveX, you can make the most of Internet resources with less effort. ActiveX Controls and Scripting give you the infrastructure needed to add language- and tool-independent extensions to Web pages. Using ActiveX Controls lets developers take advantage of existing OLE development tools and the investment they have already made in OLE.

ActiveX, OLE, and the Internet

ActiveX and OLE have become synonymous. What people once referred to as OLE Controls (OCXs) are now refered to as ActiveX Controls. OLE DocObjects are now ActiveX Documents. In some cases, entire documents on how to implement OLE technologies have been

updated to be ActiveX technologies, and the only thing changed was the term OLE, which now reads as ActiveX.

ActiveX was not meant to replace OLE, but simply to broaden it to include the Internet, intranet commercial and in-house applications development, and the tools used to develop them.

In addition to the specific technologies for creating ActiveX components, Microsoft has set a standard for the use and integration of ActiveX components. Every product from VB to Microsoft Word to Java is inherently capable of using ActiveX components.

ActiveX exposes a set of *Application Programming Interfaces* (*APIs*) that enables developing a new generation of client/server applications for the Internet. ActiveX has interfaces to integrate almost every media technology within an application. It provides extensive support for animation, 3D virtual reality, real-time audio, and real-time video.

ActiveX gives developers an open framework for building innovative applications for the Internet. ActiveX technologies form a robust framework for creating interactive content using reusable components, scripts, and existing applications. Specifically, ActiveX technologies enable content providers and application developers to create powerful and dynamic Web content and Web server extensions quite easily. This feat is achieved by using ActiveX controls, Active client and server side scripts, and the Active document interfaces and ISAPI (Internet Server Application Programming Interface).

An ActiveX control is an object that supports a customizable, programmatic interface. Using the methods, events, and properties exposed by a control, Web authors can automate their HTML pages. Examples of ActiveX Controls include text boxes, command buttons, audio players, video players, stock tickers, and so on.

You can develop ActiveX Controls using Microsoft Visual Basic, Microsoft Visual C++, and Java. Because ActiveX Controls are complex, Microsoft offers some tools that help a C++ developer create an ActiveX control. The following table describes these tools:

Tool	Description	Ships with
Microsoft Foundation Classes (MFC)	A set of C++ classes that support Component Object Model (COM), OLE, and ActiveX (among other things). MFC provides the simplest means of creating ActiveX Controls.	Visual C++ version 4.2 or later. (MFC ships with earlier versions of Visual C++; however, these versions do not support ActiveX.)
Microsoft ActiveX Template Library	A set of C++ templates designed to create small and fast COM objects.	Active Template Library (ATL) 2.1 ships with Visual C++ version 5.0. (ATL 2.0 is a Web release that relies on the Visual C++ 4.2 IDE.)

Classifying ActiveX Components

ActiveX components can be classified and broken into the following categories:

1. Automation Servers
2. Automation Controllers
3. Controls
4. COM Objects
5. Documents
6. Containers

Automation Servers are components that can be programmatically driven by other applications. An Automation Server contains at least one, and possibly more, IDispatch-based

interfaces that other applications can create or connect to. An Automation Server may or may not contain User Interface (UI), depending on the nature and function of the Server.

Automation Servers can be *in-process* (executing in the process space of the Controller), *local* (executing in its own process space), or *remote* (executing in a process space on another machine). The specific implementation of the server will, in some cases, define how and where the server will execute, but that is not guaranteed. A DLL can execute as either in-process, local or remote; an EXE can execute only locally or remotely.

Automation Controllers are those applications that can use and manipulate Automation Servers. A good example of an Automation Controller is VB. An Automation Controller can be any type of application, DLL or EXE, and can access the Automation Server either in-process, locally, or remotely. Typically, the registry entries and the implementation of the Automation Server indicate which process space the server will execute in relation to the Controller.

ActiveX Controls (formerly known as OLE control) has a broader definition. It refers to any COM objects. For instance, the following objects are all considered an ActiveX control.

- Objects that expose a custom interface and the IUnknown interface
- OLE automation servers that expose the IDispatch/Dual interfaces
- Existing OLE controls (OCX)
- OLE objects that make use of monikers
- Java Applet with the support of COM

ActiveX Control used inside scripting languages make this binary reusable components reused in the Internet world. Almost any type of media wrapped into an ActiveX control can be seamlessly integrated into your Web page. Sound, video, animation, or even credit-card approvals controls can be used within your Web page.

COM: The Fundamental "Object Model" for ActiveX and OLE

COM (Component Object Model) is the technical cornerstone for the ActiveX technology; it defines how objects expose themselves for use within other objects and how objects can communicate between processes and across a network. You can easily integrate COM objects for use in many languages, such as Java, Basic, and C++. COM *objects* are reusable binary components.

The following concepts are fundamental to COM:

- *Interface*: The mechanism through which an object exposes itself.
- *IUnknown Interface*: The interface on which all others are based. It implements the reference-counting and interface-querying mechanisms required for COM objects.
- *Reference Counting*: The technique by which an object keeps track of its reference instance count. The instance of the object class should be deleted when there is no reference to this instance.
- *QueryInterface Method*: It is called with the Interface ID (IID) to which the caller wants a pointer. Can be generated by Guidgen.exe by choosing DEFINE_GUID(...) format. QueryInterface enables navigation to other interfaces exposed by the object.
- *IClassFactory Interface*: This interface must be implemented for every object class. It provides functionality to create an instance of the object class with CLSID and locks the object server in memory to allow creation of objects more quickly.
- *Marshaling*: The mechanism that enables objects to be used across process and network boundaries, allowing interface parameters for location independence by packing and sending them across the process boundary. Developers have to create proxy/stub dll for the custom interfaces if exist. The custom interface has to be registered in the system registry.
- *Aggregation*: COM object supports an interface by including another object that supports that interface. The containing object creates the contained object as part of

its own creation. The result is that the containing object exports the interface for the contained object by not implementing that interface.

- *Multiple Inheritance*: A derived class may inherit from multiple interfaces.

ActiveX Object Model

There are two primary pieces to the ActiveX Object Model: the Microsoft HyperText Markup Language (HTML), Viewer component (MSHTML.dll) object and the Web Browser Control (shdocvw.dll). Both are in-process (DLL-based) COM objects.classes.

All interfaces defined in the ActiveX Object Model are "dual" interfaces. A "dual" interface means that the objects inherit from IDispatch and IUnknown. They can be used by client application at "early-bind" via Vtable and at "late bind" via OLE automation controller by using IDispatch::GetIdsOfNames and IDispatch::Invoke.vtable).

MSHTML is the HTML viewer part of Microsoft Internet Explorer 3.0. It is an in-process COM server and a Document Object. It can be hosted in OLE Document Object containers.

MSHTML implements the OLE Automation object model described in the HTML Scripting Object Model. With this object model, you can develop rich multimedia HTML content. VBScript running inline in the HTML and Visual Basic 4.0 running external to the HTML can use the object model.

The Web browser control object is an in-process COM Server. It also serves as a Document Object container that can host any Document Objects, including MSHTML, with the added benefit of fully supporting hyperlinking to any document type.

The Web browser control is also an OLE control. The IWebBrowser interface is the primary interface exposed by the Web Browser Control.

The Web browser control is the core of what customers see as "the Internet Explorer 3.0 product". Internet Explorer 3.0 also provides a frame to host this control. Internet Explorer 3.0 supports the following HTML 3.x0 extensions:

- Frame: Creates permanent panes for displaying information, supporting floating frames or borderless frames
- NOFRAMES: Content that can be viewed by browsers not supporting frames
- OBJECT: Inserts an OLE control
- TABLE: Fully compliant with HTML 3.x0 tables with cell shading and text wrapping
- StyleSheet: font size, intra-line space, margin, highlighting and other features related with styles can be specified in the HTML by the user
- In-Line sound and video

ActiveX Documents, or DocObjects as they were originally called, represent Objects that are more than a simple Control or Automation Server. A document can be anything from a spreadsheet to a complete invoice in an accounting application. Documents, like Controls, have UI and are hosted by a Container application. Microsoft Word and Excel are examples of ActiveX Document Servers, and the Microsoft Office Binder and Microsoft Internet Explorer are examples of ActiveX Document Containers.

The ActiveX Document architecture is an extension of the OLE Linking and Embedding model and allows the document more control over the container in which it is being hosted. The most obvious change is how the menus are presented. A standard OLE Document's menu will merge with the Container, providing a combined feature set; whereas an ActiveX Document will take over the entire menu system, thus presenting the feature set of only the document and not that of both the Document and the Container. The fact that the feature set of the Document is exposed is the premise for all the differences between ActiveX Documents and

OLE Documents. The Container is just a hosting mechanism, and the Document has all of the control.

ActiveX Documents are used within a uniform presentation architecture, rather than within an embedded document architecture, which is the basis for OLE Documents. Microsoft Internet Explorer is a perfect example of this. The Explorer merely presents the Web pages to the user, but they are viewed, printed, and stored as a single entity. Microsoft Word and Microsoft Excel are examples of the OLE Document architecture. If an Excel spreadsheet is embedded in a Word document, the spreadsheet is actually stored with the Word document and is an integral part of it.

ActiveX Documents also have the added capability of being published as Web pages on the Internet or on a corporate intranet. Imagine an in-house tracking system for purchase orders run from the same Web browsers that are used to connect to the Internet.

ActiveX Containers are applications that can host Automation Servers, Controls, and Documents. VB and the ActiveX Control Pad are examples of Containers that can host Automation Servers and Controls. The Microsoft Office Binder and the Microsoft Internet Explorer can host Automation Servers, Controls, and Documents.

With the decreasing requirements defined by the ActiveX Control and Document specifications, a Container must be robust enough to handle the cases where a Control or Document lacks certain interfaces. Container applications may allow little or no interaction with the Document or Control they host, or they may provide significant interaction capabilities in both manipulation and presentation of the hosted component. This capability, however, is dependent upon the Container hosting the component and is not defined by any of the Container guidelines as being required.

Conclusions

A specification is important to establish the basic requirements of the component you are asked to create. Before you can proceed, you must have a clear understanding of what kind of component or application is needed and why you are creating it. Appropriate questions to ask are, "What created the need for the component, and how is it going to be used?" If the person or persons can't describe the problem, they probably don't understand the problem. The last thing anyone needs is an incomplete picture of the problem, which tends to create delays and promote last minute changes that can cause unexpected results. Try to get as much of the specification as possible on paper.

After you determine the need, you can move on to designing the component. Again, it is critical to get as much information as possible. Does the problem require a single component or multiple components? Do the components need the capability to interact together? And, if so, is speed an issue? What about the skill level of your developers? How are they able to cope with change or possibly new and unfamiliar development methods? What are the support and maintenance requirements?

All these issues and more will affect the kind of component you create and how you will develop it. As a developer of ActiveX components, it is your responsibility to know the answers to these questions.

References

1. Weiying Chen et al. – "ActiveX Programming Unleashed", Macmillan Computer Publishing, USA, 1996
2. Group Que – "ActiveX Programming with Visual C++ 5", Macmillan Computer Publishing, USA, 1997
3. http://www.microsoft.com
4. http://www.activex.com