Internet Applications

Transcript of Cameron Ferroni's presentation at the Professional Developers Conference.

March 13, 1996

Cameron Ferroni
Microsoft Corporation

Download the Microsoft Word (.doc) format of this document (zipped, xxK).

Editor's Note: In a few cases, we have added in words that were apparently dropped in the transcription process. They are set off by brackets: [ ].

CAMERON FERRONI: Good morning. I appreciate the chance to speak with you this morning, although I must admit I have to stop getting this slot. The last time I spoke I spoke right after Chris Jones and he had had a bunch of 3-D V[R]ML scorpions on the screen.

Today I have to follow the level of wealth. I'll try to keep you awake through the break and we'll see how we do.

I'm Cameron Ferroni. I am a group program manager at the Desktop and Business Systems Division at Microsoft. I used to work on the Win32® SDKs and I have just taken a new role working as the lead program manager on the main note server.

Today we're going to talk about the Win32 applications, COM applications and changing slides.

So what is the Internet? There are two main pieces to the Internet that we have seen evolve. The first is the fact that it provides the productivity. This means that there is no longer a need to worry about whether or not your users are hooked up to a network, and you don't have to worry about modem configuration or network protocols in order to communicate your applications. You can now take advantage of the fact that they are on the network to improve product support, to improve communication between you and your users, and improve communication between your users and each other.

The second part of the Internet is the Web. The Web has come a long way and we have put a UPI on top of the Internet. It used to be this type with utilities, like FTP and Ping and Gopher, and a lot of text-based information. But then the Web came along and really revolutionized the UPI on top of that information.

So what we're going to deal with today is how to take your existing application, your Win32 apps, your OLE apps and add new features to those to take advantage of both the new type and the new UI metaphor.

We'll also look at ways you can write new applications to take advantage of the distributing computing features that the Internet provides.

So the first thing we're going to do is take a look at how easy it is to add simple HTTP support to your application within the latest revisions of Windows. In the Explorer we're handing out we're providing a new DOL Worldwide net, DLL low-level communications protocol support.

With one or two API calls you can simply save and load documents to any server anywhere that has FTP connections. Your application can now publish data in its native format. You don't have to save it to your hard drive. Start up FTP, hope the fire wall is not getting in the way, and

then copy it out to some server with some strange commands that your users may not be familiar with. With a simple menu click they can save their data. Nat, a program manager of our OLE team will show us what he has done to do some FTP support.

NAT BROWN: What I did is I took everybody's favorite [Microsoft Foundation Classes] MFC application, millions of sample programs, Scribble. What I have done is I have just added a couple new menu commands here simply using the Class Wizard add menu, items using the visual tools.

I added the Publish and Subscribe menu commands. They may not be how you want to show them in the UI. What I did is add this Publish command and I can just type "FTP." Let's see, No. 1, "food.doc," and I save that right out. Whoops. Well, it actually did save, but that's just a message. So I'm going to show you the code I wrote to actually do that. Pay no attention.

So what I did here was I added a couple lines of code to the standard MFC on Save Document. I put in code. I'm not going to show you where I identified the fact I typed NFTF colon. What I did was I used a new—I added an Internet Open. I did an FTP Get File. I grabbed the file and put it into a local temporary file. Then I just called the standard on Open Document inside MFC and that just opened up the file.

I'm actually showing you the publisher, or the open here rather than the FTP post. But the other code is exactly the same, about five lines of code and a menu item. I'm able to post documents up to FTP servers and pull documents down from FTP servers.

CAMERON FERRONI: So that covered the type side of it. What came next? A little while later came the Web. The Web put a UI on top of the Internet. Suddenly there are browsers everywhere. Everyone is using HTML documents as simple front ends.

What we're going to do is take you from static HTML pages in your application to a richer case where you are doing interactive work across the Net.

So the first thing that many people want to do is add HTML support to their app. There are many ways you can do this, but in Windows one of the simplest things you can do is SHELL.EXE the browser to a World Wide Web page. You could publish your help files, your FAQs, updates to your products on your own Web site and simply have a menu item in your application that allows people to access that information. Once again, Nat has done this.

NAT BROWN: What I did is use MFC; added a new menu command to a Help menu. I go to [Microsoft] MS home page. If you have a custom home page, all I did is when somebody selects this item I do a SHELL.EXE, type "www.com." This Explorer will default browser. The SHELL.EXE works [with] whatever the browser is that is established for your ease as the HTML viewer. That's what is going to come up.

I have this as a menu command. You can do the same thing putting a button on a dialog box. I have a hard-coded home page. You could put it in some configuration dialog or edit control and say, "Where do you want to jump?" Literally, it turns into one line of code SHELL.EXE and it does the work for you.

CAMERON FERRONI: That covers document access. That's two of the simple things you can do to your application today. Your application may be in its last stages of production and you just want to add a couple of quick features to make it more Internet-aware, to catch this wave of the Internet.

Now we're going to start talking about a few things you can do to your applications that may require a little bit more work and a little bit more testing but can really allow you to take full advantage of the technologies that we're introducing for the Internet.

We talked a lot about Active[X™] Controls yesterday and how you can make your applications into an active control to plug into our browsers, to plug into other control containers. That is a great thing to do but there is another side to that. That is to be a control container yourself so that you can easily take other people's controls and put them into your application.

For example, we just showed SHELL.EXE in the browser to get some HTML information around your application. Wouldn't that have been cleaner if you actually put a control in your application that was an HTML browser control, such as the one that we provide?

By doing that a user can stay within the frame of your application and get the HTML. The other thing you can do is maybe a RealAudio control in your application to get telephony feeds down. Maybe you create sort of a product support thing where you press one for information about product X, and press two for information about product Y, and simply host the RealAudio active control in your application.

To do this you simply need to implement one interface in OLE and that's **IOleContainer**. There is quite a bit of code you have to write behind that, but the Visual C++ environment actually allows you to graphicly add controls to your application without writing any code at all. And it allows you to do any type of control. You can do a control that allows you to type across the Internet or just does a calendar or a clock.

The second piece of technology we talked about in detail yesterday was the concept of Active Documents. Active Documents allow your native data format to be displayed inside of the browser window, inside of other document containers such as the office binder or inside the shell as we saw in the "Nashville" demo.

In order for you to make your application an Active Document, you simply need to add two interfaces. If you are already in OLE, a PROC server, by adding **IOleObject** and **IOleDocument** you can also be an Active Document.

And, once again, we will have a demo here in a second and it will show you how you can take your document and integrate seamlessly with the shell, and integrate seamlessly with the browser, provide a common UI paradigm for the user. They are used to going back, forward. They want to look at HTML. They want to look at your application. They want to look at a Word application without starting multiple instances in having Windows open here and Windows open there.

So, Nat, why don't you show us what you have done.

NAT BROWN: What I did is I took this MFC application again. You will have this on one of your CDs. It's a scribble enabled as a doc object. What we've done is added some code to the Scribble project that gives you doc object support for free by giving you a new base class in MFC.

We have another sample that we'll put online that shows you how to do this from scratch with the existing base control framework that is part of the Sweeper or Internet SDK. What I did is took this code and basically compiled it. It wasn't that tough. I just derived it from this base class that we built.

Now all I need to do is I'll go to some Web page where I have a reference to a Scribble document, which happens to be right here, I guess. I'll just browse straight into that. So if

Scribble comes up in frame, we've seen this with Word and Excel, you have seen this happening. But it's real easy to do just by adding a couple interfaces to your existing application.

If you have MFC there are instructions that walk through what you need to add to it to make it a document object or Active Document. This was real easy to do. We basically took the sample, added files, recompiled, added interfaces to the base class.

And here it is as an Active Document right inside the frame. So I browse to it. I can browse back. I can—I have got toolbar negotiation. This is just my standard Scribble application I was using a second ago, same executable, now just running as an Active Document.

CAMERON FERRONI: You are getting the code, the entire source code to this application when you leave the conference.

So we've talked a little bit about the early protocols. We've talked about the UI paradigm that's being introduced because of the Web. I want to talk a little bit more now about the protocols in detail. FTP was a very simple one-way protocol. HTTP is a very simple one-way protocol. For the most part the way people are using it today is to do gets from a server returning HTTP responses.

In order to overcome this limitation many people have started to use Windows sockets as their method for communicating across the Internet. Windows sockets provides you with two-way communication and it provides you with a way to do real-time streaming data for audio and video.

It requires smart clients and smart servers on each side in order to do Windows Sockets. One of the problems with this, however, is that you have to parse your own data out of the stream. You get a big blob of data back, find your data types, find your parameters. It requires a lot of work for you as an ISV to take advantage of this protocol to get the two-way communication.

So what can we do to make your lives easier? We saw a picture yesterday during Chris Jones' talk where he showed how URL monikers were built on top of HTTP which was built on top of the caching, which eventually talked to Windows Sockets. In fact, the way that actually works is there is a COM layer underneath the URL moniker and custom binding that does talk to the Windows Socket.

Why don't you call the COM layer directly from your application? That essentially is what Distributed COM is all about. You talk [to] COM, and if the object you are talking to happens to be on the network the COM layer handles that for you. If the object you are talking to happens to be on your local machine, the COM layer handles that for you.

You don't need to worry about the network in this case. So why Distributed COM [DCOM]? It's all about leverage. Most of you already know and understand COM. If you were at the '93 PDC you heard it over and over and over again; Win32, OLE 2, Win32, OLE 2. We drilled it and drilled it. You did it.

So it takes your existing skill set. It takes your existing tools, all of the visual tools that we provide as well as the visual tools provided by third-party tools, ISVs support developing these applications.

It's programmable. It's a programming interface that you are comfortable with. It provides data marshalling with you. You don't have to parse your own parameters. You can simply and easily use the COM interfaces and let them do the marshalling for you to get your parameters out of the calls.

It provides security. You can put security on the objects either on the server or on your local machine, so only users that are allowed to use that object are able to. And that's provided by the built-in security model. And they are robust. If a COM object is being run out of process and something happens to that COM object, it doesn't take down the machine. It doesn't take down your application. It only affects itself.

It protects your existing investments. Any service you have written in the past now works across the network. Any existing 32-bit COM application works with DCOM. EXEs generated with V[isual] B[asic] 4 just work with DCOM.

Any FTC application works with DCOM. All of the applications that you have written will just work with the new Distributed COM.

How does this tie back into the active platform? Well, there are two ways. Many people's clients will be Active Documents, document objects. For a document object when you make a reference up to the server, the server will start the remote COM object, pass an interface pointer back to the client document object and the two of them can work across the Distributed COM.

Alternatively, you can start the document object on the client, ask the user for the name of the server where they want to run a remote object. The user can type that in and they will be communicating.

For a control we saw yesterday, one of the main ways for controls to bind is via the URL monikers. The active control is binded to a URL moniker and then you have a direct interface back and forth between the server and client. They no longer are needing to go over HTTP to do this. What does it really look like? I think this picture explains it a little better than the last slide.

Here is your client. It's your standard browser. You do an HTTP get like you do today. On the back end we run an ISAPI out that takes a look at this get and says, "Hey, this person is trying to invoke a COM object on the server." So I'll invoke the server object. Here you can see we've invoked a server-side Scribble.

That object returns an interface pointer to the ISAPI app. The ISAPI app takes the data from the object and interface pointer and passes that back to the client. And the doc object version on the client now shows the native data on the client side.

We've seen that happen today but the cool part is that now they are talking DCOM back and forth because they have each other's interface pointers. They can communicate without going over HTTP and have great two-way communication.

(Applause.)

CAMERON FERRONI: So as a matter of fact the demo we were just looking at was doing this. Nat, why don't you bring it up again and show us what we got?

NAT BROWN: We should have the client on one machine. That's over there on the right. We have the server over on this other machine. Why don't you take a look. So what happened is when I browsed into this document this document got launched on a server and now we're both looking at the same document. When I scribble over here it appears over there on the server.

CAMERON FERRONI: And vice versa. You can see the difference here because on the machine on that side we have the browser menu still there as well as the Scribble toolbar, but up on the server it really is just a version of Scribble running up there.

NAT BROWN: It's the same Scribble version we were using. Let me take a second and show you actually what we did in a couple places inside of the code.

What we did is we defined a custom interface we'll use to communicate between the client applications, client Scribble document and server application. So that's over there on the right.

So I defined one interface. I went in and typed this in. It was easy. **IScribbleInterface**. It has two methods that we've chosen to remote between the client and server, **AddStroke** which takes a pen width, a number of points that are in the array that make up this stroke. And then an array of strokes.

You will notice here that all I'm doing is specifying in this IDL code what the parameters are to the methods. So if you are familiar with creating ODL files or IDL files or you would like to see —we get it on this other monitor as well. The left half of the audience is having trouble.

CAMERON FERRONI: Can we get No. 15 on both?

NAT BROWN: So we have these two methods and we used the data types we're familiar with. I'm familiar with points. These are the same data types I was using in the Scribble application already. I wrote an interface that did that. So that's the client side.

Then I made another interface called **IScribbleServer**. The Scribble server has two methods to it, register a client and revoke client. So what happens is when the client connects up, when it gets a Scribble server interface—and I'll show you how it does that in a second—it takes its Scribble client interface and hands it up using this register method on the remote server. Then the remote server has a connection to this client, and that's how the two-way communication becomes established.

When it's done it calls for **RevokeClient** and, actually, I didn't add any code in this sample that cleans these things up automatically, but that is something you would want to do in a real distributed application and something we'll talk about this afternoon actually in a breakout session called Building Great Distributed Applications II.

So I wrote that code and then I went into the Scribble document and added those methods, added those methods on here into the Scribble document. So I have this method that I just implemented called **AddStroke**. Here is an example: Here is where the server calls us back and says the server has scribbled.

When Cameron scribbles on the server he calls down to me, the client, and I get this method call and I simply create a new stroke object. I add the points that are in the array that were passed to me. I update my view and I notify myself that I have changed.

So it's just a couple lines of code to do that. I didn't have to do any COM activity. I didn't have to manage the connection. I'm using COM the same way I did.

Further down here I happened to use the same code between the client and server. What I have here is I am notifying the server. So on the client side in this set of code right here, if I have got a reference up to a server that I'm communicating with when I draw **NewStroke**. I just call this **AddStroke** method on the server and he gets the information. Then we'll talk about this other one in a second.

And then I also want to show you how we actually pass this reference between the server and a client. It was kind of glossed over in some other people's talk, but if you paid attention carefully in the online demonstration they mentioned these are VRML world format.

They mentioned one thing. They added a comment to the VRML that said the IP address of the server, or the DNS name of the server they were going to connect to who was going to be the chat server for this VRML world.

That's an important point, embedding some information being handed back to the client applications so they can get their connection back up to the server. In COM we have a very easy way of doing that. You can pass a DNSA; DCOM-handled DNSAs. We have a richer way of passing references between people. Let me find it here.

This is it right here. What I do is, Scribble is the standard Scribble application. It stores its application in a compound file. When I'm saving this document out I create a stream, and then I call this function co-marshall interface. That takes an interface pointer that I have. This is up on the server. It takes the interface pointer I have and puts that into the document, into this stream inside of the document.

Then when the client loads the document all he does is open up this stream and unmarshall the interface. That gives him his reference up to the server.

These are the only lines of code I added to this serialized method in order to allow this connection to be established between a client and server. There is no WinSock programming, no send and receive. It's all pure COM.

CAMERON FERRONI: So you launch that server via an ISAPI app?

NAT BROWN: Yes.

CAMERON FERRONI: Did you have to launch it on the same server the ISAPI was on?

NAT BROWN: No, I didn't. With DCOM a server can run anywhere. We're using the ISAPI for COM activity. What we're using that spot for is a spot you go, run a DayView point to a document. If we could extend the sample in the future we could do the peer-to-peer to an HTTP site. Other people could come in and you would end up with pure peer-to-peer opportunity.

CAMERON FERRONI: Object broker. Who determined which server's load was the least.

NAT BROWN: Yes.

CAMERON FERRONI: And run this instance of the server somewhere else, somewhere else in the corporation, somewhere else on the Net, so as to minimize load on your servers.

NAT BROWN: Yes, and we have great ideas of that for future versions of DCOM.

CAMERON FERRONI: Great. Let's go back to slides. I think you can all understand how powerful that can be. You can now easily write high-performance applications. You don't have to worry about WinSock or HTTP. You can use an interface that's well known to you. But there is something else here. I know when I was working on this PVC for the last end months there were a lot of us working together to pull together agendas and schedules and session times. It always seemed like someone had a document open and they were working on it. And I really needed to change this field so I would send them a piece of mail, and they would copy it from their mail and paste it into Word.

They would screw it up so I would have to go in late at night and fix it. It became a real pain to work on the same document at the same time.

When Nat wrote this application he had Smarts on the client and Smarts on the server. And the server was keeping track of what the client was doing and how it was connected. So we thought about it for a while and said, "Jeez, couldn't we have the server keep track of what multiple clients were doing at the same time?" As a matter of fact, we could.

So what we have here is what we call multiuser Scribble, which is actually the same application we just showed. I can go to my browser, which will be on the right-hand side. And Nat can go to his. I can go to the group Scribble page. We've got one here, the two clients.

NAT BROWN: Yep, it's called two clients. Now we both just browsed straight into a document off of the same Web page. We're both running on client machines and let's see what happens. It looks like we're both connected and both able to draw on the same document at the same time.

So what happened here was that piece of code that I skipped over a few minutes ago—most of this is the code that came with Scribble. I have this little four loop here which is when there is a server application. The server just keeps a list of all the clients that have registered with them.

It goes through this loop and tells all the clients a new stroke has been added to this document, add that in. We can have as many clients [as] we want hooked up and it fans it out. You can imagine you would do a lot of better things here to allow higher-performance server application to make this three-threaded to ensure consistency in different ways. But in terms of simplicity you are just using the things you are familiar with.

So any two people who happen to be up here on stage, any two people browsing this document would be connected with each other and scribbling together. It's an easy way to write Groupware applications and complex Groupware applications.

CAMERON FERRONI: It's not two people, right? Any number of people could be going up to this document now and adding information, and we would all see it and the server keeps track of state and all those things?

NAT BROWN: Right.

CAMERON FERRONI: We think this is pretty powerful. What we're saying is HTTP is a great metaphor for browsing, for locating information. We've seen that on the Web today. It's a great way to move around and find data. Once you have found the data and you really want to work with it, then leveraging DCOM to do that is a really rich and powerful set of features.

As we talked about obviously it's not this simple. What we've done today, you will get the code, it really was this simple. But in order to make a real, full-blown distributed application there are a lot of issues you should take into consideration.

Designing interfaces for the network. When you are on the Internet you have low latency [to] high latency, low bandwidth interfaces. You should be careful about how much data you are taking across, how many round trips you make. When you are on the intranet you have high bandwidth, low latency and you don't have to worry as much and you can pass as much data back and forth as you want.

In topology, the Scribble demo we showed was both client/server but it looked like peer-to-peer because it was the same object on both the client and the server. And it didn't matter that it was a server machine. They could have both been on clients talking back and forth to each other.

You need to make a decision whether you want to break some functionality down between the server side and client side, or you want to be able to do peer-to-peer communication.

The server can do a lot of things for you. It can handle keeping clients in sync, making sure everything has the same date, handling a situation where you have a certain client that needs to access an object, locking the other clients out from accessing the object until he is done with it.

The client should really take care of local operations that don't affect the rest of the clients. If I'm looking at something and I change the size of the window, everyone else using that document shouldn't see that I have done that. If I zoom in to get a better view, everyone else using that document shouldn't do that.

Basically, what we're saying is you can make this as simple or as sophisticated as you need. What is the availability of this? Is this real? You bet it's real. The Distribut[ed] COM that we're using on these machines today is available on the PDC CDs you have. It will be available in Windows NT™ and 4.0 Beta in April and Beta available for Windows® 95 by [second quarter] Q2'96.

At the same time one of the big issues is fire walls. Great, you are doing all this wonderful communication. How do I get through my fire wall? The only thing my fire wall knows about is HTTP and maybe RealAudio.

What we're going to do is tunnel DCOM over HTTP by the second quarter of this year in that same 4.0 Beta.

What is your takeaway. It's easy to enable your application. Forget about the DCOM stuff for a second. We showed you how to add FTP in three codes, browser code in one line of code. It's great. It's simple for you to participate on the Internet within the framework of the application you have today.

If you are already a COM application there's lots of things you can take advantage of today with your existing application in the sense that you just work distributed.

There is opportunity now in the document world. Everyone is doing really cool things with documents and making their pages more active and putting controls on them. That's great. Take advantage of that. But we think there is even more room for innovation in the application space and taking advantage of the fact you are always on a network to make your applications more distributed, more powerful and better for users.

That's all we have today. There are two breakouts this afternoon on writing distributed applications, which cover one from the WinSock perspective and Henry Nash from the DCOM perspective.

Nat Brown will be presenting in two breakouts on COM itself and Mike Ryland and Nat Brown will be presenting this afternoon. We're about ten minutes into the break, so we won't take any questions right now. And thank you very much.

(Applause.)