From: Christian Fortini

Sent: Thursday, March 04, 1999 2:27 PM

To: Bill Gates

Cc: John Shewchuk (Exchange); Michael Toutonghi; Eric Rudder; Paul Gross (Exchange); David Vaskevitch; Paul Maritz; Anders Hejlsberg (Exchange); David Cole; Chris Jones; Jim Allchin (Exchange); Yuval Neeman (Exchange); Victor Stone (Exchange); David Stutz; Oshoma Momoh; Jean Paoli

Subject: Re: Our presentation strategy

Thanks for the clarification. I guess I owe you a complete and detailed plan of how I think we should push Trident and related presentation technologies further in the next 2 years. I will arrange a time with you to make this presentation as soon as possible.

In the meantime, I'd like to first outline this plan quickly below and then address your points. Here are, in a nutshell, the directions that this plan focuses on so far:

- **Move our API and component model to COM+.** Beside being the component technology that the company is betting on, there are a few other good reasons for doing this: it is necessary for VS7 (VB in particular) to target our API; it will allow to write "binary" (C++), safe, yet untrusted components for the client, something impossible today; and providing that other areas of the system will also be exposed using COM+, it will ensure much more consistency between these and the presentation API. We would like to do this while maintaining backward compatibility with current Web content, at least at the script level. This means that every Internet page out there would suddenly find itself running on COM+. Beside creating huge immediate adoption, this would also allow us to move our current DHTML clients and the rest of the Internet forward with minimal hurdle for the developers. For other languages such as VB and VC, we would modify the APIs to be optimal for these languages and to comply with the COM+ requirements such as the Common Language Subset for example.

- **Integrate Presentation with XSP and XSL.** We want to make it totally natural to use XSP and XSL with the IE/Trident presentation platform. This includes a variety of things ranging from sharing the same user / session state saving model to integrating XSL as our primary data-binding - or rather "building UI from data" - mechanism. Applications like NetDocs, the Platinum Mail Client or the Neptune Shell for example show that having powerful, declarative mechanisms to build arbitrary UI (aka, "views") from heterogeneous data structures is essential to building modern applications. They all use a variation of the XSL language for this and have all run into all sorts of limitations. We need to address those.

- **Improve layout, display, printing, multi-media support.** Modern applications have very stringent needs in the areas of layout and display, as well as multimedia. We have made good progress in this area in IE5, notably with much faster layout, more complete positioning system, many typographic advances thanks to Line Services, vector graphic support (VML) and so on. However, we still suck in many other areas: our printing and print preview support is lame or inexistent, our display refresh speed is still too low to build complex animations, text layout is still missing many features, we do not support modern display effects like alpha-blending or zooming, our multi-media effects (filters, transitions, etc.) are limited, we do not have a well defined notion of time in the product, video does not integrate well with the rest of the presentation, etc. We need to fix these things. Major drivers in that areas are applications like Netdocs for text layout and printing, the Neptune Shell and consumer applications like Publisher, Pandora, etc

- **Plug deficiencies in our model.** The goal here is to enhance our platform until it is possible to write real productivity applications on it that can work on private data and do data analysis, editing, etc. As you say, this means rich UI. We have made some progress in IE5 with support for data transfer, drag and drop, mouse capture, view state persistence, etc. However, the ultimate incarnation of this platform should be a powerful, full featured client runtime including COM+, Trident, an application model, storage and data access, on which it should be possible to write Money, Premiere, Autocad or Excel. We need to build in powerful "controls" or UI widgets and create a framework for developers to build their own. We need integrate drawing services, selection services, richer data transfer support, better input support, etc. I am also looking at the LOB applications that we follow closely (Merryll Linch, Prudential for example) as well as at the Neptune Shell and NetDocs to guide us in the right direction here.

- **Improve our editing infrastructure.** NetDocs and Tools (VS7 and Access) are the drivers in that area. Our goal here is not to provide a finished editing surface and UI, but rather a set of powerful underlying services to allow other groups - or external ISVs - to write one We have made good progress in IE5 by formalizing low level access to the Element Tree and extracting the actual Editing Services out of Core Trident. Groups like VID and NetDocs have build reasonably powerfull editors on top of Trident. There is much more to do though. The laundry list of features that these groups need is pretty long !

- **Continue to embrace standards when it helps us.** Most customers outside of Microsoft routinely ask us for more standard support. The press always makes a big deal of it and our marketing group is pushing us in that direction. There are a few holes in our support of HTML 4 and CSS 1 that we may have to address to continue winning the reviews In addition, if Gecko successfully delivers on its promise of 100% standard support and gets momentum around that message, this may push us to embrace more of CSS 2, DOM and other standards.

Now, regarding the points you are making below:

Standards

I am not sure I completely understand how the Trident API is devaluing Windows. I may be missing some important data point here However, we have the best implementation of a browser on the market and we have the richest and faster client platform for the Internet. Given that it runs best on Windows and in fact is integrated into Windows in many ways (besides being deeply integrated into the Windows UI itself, IE is also a set of generic components that other Windows applications can use) it seems like this is actually adding value to Windows.

There are many areas in Trident and the IE platform as a whole that are not disclosed as open standards, and are actually very much proprietary. In fact, as far as the object model is concerned, the only "standardized" piece is the W3C DOM, which basically describes the Element Tree and defines about 30 methods and properties to manipulate it This is a very small subset of our object model. Everything else is not part of any standard, including the part we share with Netscape Some examples of proprietary APIs are our extensions to the W3C DOM, our style and style sheet object model which defines a programmatic API to CSS, the DHTML Behavior component model which allows developers to create reusable components for DHTML pages, Dynamic Properties (essentially expressions in properties), Data-Binding, the Markup Services interfaces on which NetDocs and our Editing component for Outlook, Access and VID are built, and many many other pieces like ActiveX controls hosting, etc.

For the parts of Trident that are actual or proposed standards like HTML 4, CSS level 1, VML and the DOM, we have the best implementation on the market and cloning it is next to impossible. Even Netscape is struggling at it now and is lagging way behind. Their latest response has been to open up their source code and hope that the Internet developer community would help, but this still seems quite far from actually delivering a product. The fact that they want to stick to 100% open standards does not help them much. the standard specifications by no means defines everything there is to know about the APIs and a great deal is left to the de facto standard created by the wide adoption of our implementation

It is certainly not my intention to give away the Windows value as standards. Standardizing some pieces of the IE api (like the DOM or VML) or embracing some of the existing standards like HTML 4 and CSS has helped us tremendously with ISVs/ICPs and in the press and ultimately contributes to a wider adoption, hopefully encouraging ISPs to create more content and applications specifically targetted at IE, eventually contributing to making the Windows platform more appealing to users. This trend is still timid, but increasing on the public Web and catching on more rapidely on the intranets.

Relation to VSForms

I am not completely sure what you mean by VSForms below. There several different flavors of it that I know of: one is WCF.UI which is a well organized Java and now COM+ wrapper on top of USER/GDI; another one is the COM+ Trident wrapper that JohnShew and his team have started developing. This later one is exposing Trident functionality in a somewhat different way than DHTML, more friendly to VB developers, but enough different to be incompatible with existing Web content, even with HTML 3 2 content as soon as it include scripts The premise was to make a server version of this model that would project itself on dumb clients using HTML 3.2. Lately however, I understand from JohnShew that VSForms is moving away from that idea and refocusing on wrapping the XSP model.

I am going to guess that by "integrating with VSForms" you mean bringing more of the Windows power into the Presentation Platform, for example by making it easier to use GDI directly from within Trident, to embed USER controls, to access underlying Windows APIs or at least functionality. I completely agree. Moving our API to COM+ should help, providing that other groups do the same and deliver other system services using it. The mere fact that the COM+ component model is more strict and specific than COM should ensure more consistency across various domains and make it easier for developers to take advantage of the full panel of Windows services from within Trident.

Richness

I completely agree that people want to deliver as Web applications the same kind of software that they have been building for year as shrink wrapped packages. Our Internet Client platform needs to allow for real productivity applications to be built on it and take full advantage of the underlying power of Windows for that It needs to allow for everything that is possible with USER/GDI and more. We need to expand the set of in-the-box controls and other UI widgets we offer and have a strong, robust component model and framework in the core to allow the building of others. We need to make inter-application operability possible We need to allow Web applications to work on local or private data with the appropriate security checks. We need to make the creation of rich design surfaces such as Visio, Autocad, Photoshop or Excel is not only possible but easy.

In fact, I believe we need to go beyond what Windows can do today and in some areas we already have. We need to continue to make it incredibly easier to create rich looking views including rich graphics, complex text layout and other media types such as sound or video. We need to build alternative types of input such as speech right into the platform. We need to natively integrate Accessibility and IME support into every single of our components. We need to make controls smarter so that they can automatically remember history and do things like auto-completion, automatic retrieval of previously used entries in similar fields (Intelliforms), etc.

We also need to make it possible to build very diverse and rich looks and feels, simply by tweaking a few styles properties, much beyond what native Windows could do without having to completely rewrite a UI framework. The time of common look between applications is gone. The Web killed it It is interesting to see that most shrink wrapped consumer applications go a long way to achieve specific looks even in the simplest pieces of UI such as check or imput boxes. Most try to mimic the "web look and feel" with lots of text, hyperlinks, graphical background, mouse over effects, etc. We need to make this trivial to achieve for all Windows applications.

Integration into Windows

I completely agree that the evolutions of the Trident presentation platform should be more and more integrated into Windows over time. For example, every piece of the Windows UI should ultimately be built upon it. The Neptune project is essentially attempting that. There is a question of how much of it we want to deliver on downlevel Windows platforms though. Even though this is mostly a business decision, it does impact how much we can depend on specific inovations of Windows 2000 for example.

Running on the server

This has been the subject of much debate lately. There are a couple of major problems in my mind Beyond the pure bandwidth problem which makes it difficult to build a responsive enough UI over any wire other than a fast corporate net, a 3.2 browser does not easily allow local refreshes of the UI. Usually, the only thing available is page navigation which is a heavy and slow operation. This makes it difficult to build a UI that feels anything different than the current HTML 3 2 Web, in which case of course, there is no need for a server presentation component beyond UI building logic which can be achieved easily by ASP or XSP/XSL.

The other problem is that the UI organization of an application written for HTML 3.2 is likely to be very different than if it is written for a richer platform that includes menus, toolbars and other complex controls, local updates, advanced gestures like drag & drop, selection, design surfaces, etc. Hence, it is usually impossible to re-use much presentation code between the two, making the "server Trident" less attractive since little code can be shared between the client and server versions of the same application. Pieces that can be re-used are likely to be non-UI business logic or data retrieval / management logic.

As a result, it may make more sense to consider the presentation platform to be client only and focus on XSP and XSL (basically the business logic and the transformations that produce the UI) to be the part that has a chance to easily migrate from the client to the server depending on the smart level - or processing power - of the client device.

We actually have already spent quite a bit of time and effort investigating a version of Trident running on the server The hopes we were entertaining were that there might be a right level of components that could be assembled into pages that would indifferently run on the server or the client. The implementation of each component for the server and the client would be different but the pages using them could be the same. We have not pursued this idea and the associated prototype far enough to really know yet whether this is valid or not. We could do so in the future, but this is a hard problem and it would require quite a bit of brain power. In the very resource contrained world in which my team operates, I am not completely sure this is the best use of my guys, especially when they could be working instead on adding value to the client platform along the lines I describe above.

The VSForms folks have investigated similar concepts with server and client VSForms I am not sure whether they have a working prototype at this point that demonstrates this actually working in an acceptable manner. We could have them spend resources on this subject but again, given the problem that I exposed above, I am not sure this is actually a very useful exercise, vs for example developing great tools for XSP, XSL and the future COM+ based DHTML platform that I describe above However, If you feel that it is, I am certainly willing to go take resources on my team and research opportunities further in that area, hopefully with a lot of help from the Tools and the Developer Parade people

I hope that all of this addresses a part of the frustration that you have been experiencing with the Presentation Platform and that it clarifies the directions that I see us taking with Trident and how we can contribute the the Developer Parade and Universal Runtime effort lead by J Allard. Again, I would like to get some time with you to get through directly and in more details. I that is ok by you, I will try to have this scheduled for some time this coming month.

Thanks
Christian

> ----- Original Message -----
> **From:** Christian Fortini
> **To:** Bill Gates
> **Cc:** John Shewchuk (Exchange) ; Michael Toutonghi ; Eric Rudder ; Paul Gross (Exchange) ; David Vaskevitch ; Paul Maritz ; Anders Hejlsberg (Exchange) ; David Cole ; Chris Jones ; Jim Allchin (Exchange) ; Yuval Neeman (Exchange) ; Victor Stone (Exchange) ; David Stutz ; Oshoma Momoh
> **Sent:** Tuesday, March 02, 1999 9:34 PM
> **Subject:** Re: Our presentation strategy
>
> Sorry I have not responded to this faster, I was away for a couple days and just got back into town tonight I will send a full reply tomorrow morning.
>
> Christian
>
>> ----- Original Message -----
>> **From:** Bill Gates
>> **To:** Christian Fortini
>> **Cc:** John Shewchuk (Exchange) ; Michael Toutonghi ; Eric Rudder ; Paul Gross (Exchange) ; David Vaskevitch ; Paul Maritz ; Anders Hejlsberg (Exchange) ; David Cole ; Chris Jones ; Jim Allchin (Exchange) ; Yuval Neeman (Exchange) ; Victor Stone (Exchange) ; David Stutz
>> **Sent:** Friday, February 26, 1999 11:54 AM
>> **Subject:** Our presentation strategy
>>
>> I have been frustrated with our presentation strategy since it has been confusing, devaluing, and fragmented.
>>
>> This is a critical area for us.
>>
>> We must provide the presentation API of choice in a way that is not commoditized.
>>
>> One approach is to focus on making Windows Terminal Server more popular - however at this stage this can only be a piece of our strategy not the only one.
>>
>> The Windows APIs are still better than HTML even with IE 5 but we keep making HTML better to our own detriment. We standardize great presentation API and devalue Windows more and more.

There is a subtle and powerful way to fix this. It requires us taking the Trident technology and integrating with VS forms but with some new abilities.

1) The ability to run on the server and send a downlevel UI to a HTML 3.2 client. This is hard but important. Active controls would require us to have a Windows Terminal server element in the browser so we couldn't do all things for all clients. It doesn't have to work for all apps. Apps may have to provide hints to help with the downlevel. It has to be doable for new applications though.
2) Being rich so that things people have done with GDI/User can be done
3) Being something we don't give away as a standard. A subset but not the advanced capabilities.
4) Being available on Windows clients as a layer at first but deeply integrated over time
5) Being as evolutionary from Windows as possible and hosting some of the key forms packages.

I would love to hear a strategy that doesn't just kill off our presentation asset and force people to write server only applications that ignore the needs for knowledge workers to have a very rich UI. Knowledge workers don't just want to run applications - they want to combine data between them and analyze data from them. This requires rich UI and we should lead in this. As it is our UI asset is draining away.

When I talked about "super-trident" yesterday it was the idea of something that did these things explained above.

-----Original Message-----
From: Christian Fortini
Sent: Thursday, February 25, 1999 7:06 PM
To: Bill Gates
Subject: "Super Trident" ?

I heard that, during a presentation from JohnShew to you this afternoon on the new Windows App Model, you expressed interest into something you called "Super Trident", which from what I understand would amount to a "virtual Trident" running on the server and projecting itself onto a HTML 3 2 client.

For some reason, I was not invited to this meeting, but this is something that we have talked about quite a bit in previous conversation with the Tools, XSP and Neptune Shell group. I would be interested in your thoughts on this subject.

Thanks
Christian