PLAINTIFF'S EXHIBIT
3073
Comes v. Microsoft

From: Eric Rudder
Sent: Thursday, July 15, 1999 9:56 AM
To: Bill Gates
Subject: latest draft, just to have

The Next Wave.doc

The Next Wave

We have many good new technologies being developed in our product pipeline. However, we seem to be lacking a strategy where we make the whole greater than the sum of its parts. At a time when our core franchises are under such strong attack from competitors, this situation is especially painful.

We do not have the luxury of time to change our existing product plans over the next year or so, but looking beyond that, we must set some goals for how our next generation of products will renew their leadership positions.

1 Feedback Cycles

Historically, we've had incredible success when we've built a Virtuous Loop consisting of a great version of Windows, great tools to deliver applications, great applications, and Internet services which enhance those applications. This never-ending cycle of feedback is something we need to continue to draw upon as a company, even as our divisions enjoy ever increasing autonomy.

1.1 Traditional Cycles

Our traditional cycle of opportunity goes something as follows:

ISV's consume the Windows API. IHV's consume the Windows DDI. OEM's consume the Windows OS itself. The OEMs do not want to invest in proprietary OS projects because their commodity business model forces them to stay lean, mean, and focused on very quick design turnarounds and supply chain management. ISV's want an OS because they don't want to waste time and effort on tracking rapid innovations in other vendors' software, or in hardware. IHV's want an OS because they like the leverage that comes from a standardized socket where they can quickly and compatibly plug in their inventions. Windows provides a common ground for all three of these industries - it makes the combined market more efficient.

Our competitors are exploiting similar cycles of their own:

Sun - sells network-centric server hardware, and the Java platform, which links service-producers to application programmers in a network-centric way. Service providers like the safe, standardized, and cross-platform socket that EJB, Jini, and JavaBeans provide. Application programmers like the availability of many services in a clean, modern, component-based API, and the consistency of the extensibility model with the underlying platform. Corporate customers who need code-based server solutions don't want to recreate this extensible platform, and so they adopt Java. Note the similarity to the Microsoft model - and note the similarity to the URT strategy. This means that our best bet is to compete head-on, rather than to coopt.

Oracle - sells database software, joins corporate producer of data/biz logic to integrators. commoditizes the hardware/os. The more that you put in the database, the better the integration story becomes. Oracle has co-opted Java - they use it and market it. Oracle will also co-opt the NT wave. Neither Java nor NT threaten Oracle's underlying circle, and so by co-opting them, they use their competitors' strengths to increase their own strength.

IBM - sells reliable hardware/software. Corporations that must depend upon computers buy IBM's products and services.

Cisco - sells the equipment that makes the internet run. Businesses are embracing internet delivery of their services. Hardware vendors are making "appliance like" devices that use network attachments and standard protocols to deliver their function, whether these be server clusters, cellphones, settop boxes, or PC peripherals. ISPs and carriers want to offer the best set of internet services to their customers, so they buy Cisco. Network operators and carriers are consolidating, creating huge pools of customers. Cisco will coopt any and all operating systems - they are just network leaf nodes.

We can learn from the cycles that are being created by other our competitors that exploit network effects other than our traditional one. These are important to understand not only for their impact, but also for the opportunities that they present.

1.2 The Cycle of the Web

We have a very valuable corporate asset in the form of our understanding of what makes an application and how to build good apps. But we are still trying to push this knowledge into the OS, rather than trying to push it into the web. MS should be focused on building "hosted" apps, on understanding "webified" apps, and on building or investing in infrastructure for networks (including app services).

The web model includes many of the same players, but there are some new ones as well, such as ISP's. The OS has increasingly become marginalized, making weaker alternatives, such as Linux, more appealing than they should be. ISV's now consume "Web API's." IHV's "plug in" using network protocols, and the OEMs are facing commodity pricing like never before. IHV's used to make boards that conformed to PC bus specs; but now they make network appliances. ISV's used to build Windows apps; now they create sites and front ends that exploit open protocols. ISP's are beginning to take on the role of distributor, which underlines the OEMs' tenuous position. While the OS is still important for OEMs, the value of this element is diminished, due to both price pressure and to the fact that open protocols make the implementation of the entire machine replaceable. We have in some sense, failed to make ISP's a key part of our cycle, in part because we compete with them, but mainly because these companies are much more interested in common glue for communications than in glue for applications, and because of this, they tie themselves to companies like Cisco.

1.3 Reinvigorating the MS Cycle

[This section must tie in with the "Why" slide. I'm not sure if this gives you the Linux/Java thing like you want.]

The competitive situation we are currently facing is a tough one. We are in danger of losing our desktop franchise, unless we take vital steps to renew it. We must exploit the integration of our assets, and make the whole greater than the sum of its parts. Yet I fear we are not on the course needed to make us successful here.

Much of our new platform thinking is actually being done by the Tools group, especially with COM+. Yet our applications group doesn't find much of the COM+ work relevant to their short term, and our groups in CCG aren't basing their future plans on the work being done by this group either. In addition, none of this work reaches out in a new way to ISP's, to include them in a new cycle of prosperity.

There are several key imperatives we must deliver on, in order to refresh and renew the Windows franchise. While there is an incredible amount of good work that is going on, I want to take the opportunity to prioritize our efforts, and focus on five key areas.

These areas are: The User Experience, Establishing the Windows Schema, Manageability, Delivering a Clear and Compelling Message to Developers, and Building Internet Scale Services.

[Maybe we can make the initiatives tie to the Cycle elements:
OS – User Experience
ISV – Developer Message and Schema
ISP – Internet Service
IHV – Manageability?]

Picking a small number of areas to focus on will help us prioritize, and should help amplify our most important messages, both internally, and externally.

2 The User Experience

We must innovate in the user interface, as well as continue to attack some of the complexity that we (and indeed, the entire industry) have created.

2.1 User Interface

Windows has long presented a rich user interface that has been embraced and adopted by both end users and application developers. With the rise in popularity of web-based paradigms, developers are now building applications that do not uniquely leverage our UI infrastructure. HTML delivery of UI is seen to be a universal panacea, because it theoretically allows for greater system independence in all ways – operating system, graphics capability, browser version, device form factor, etc. The fact that this is an illusion does not make our task any simpler. We must compel ISV's, both "traditional" and "modern," to embrace our UI innovations. Our challenge is no less than the need to re-establish thought leadership in user interface design.

We are currently investing in new UI design in a few different areas. We can deliver many aspects of these innovations in Millennium and refine them in Neptune, so long as our 'high concepts' are successful. Among the user interface initiatives in the company today are:

- Neptune – Activity Centers, etc.
- ePad – New metaphor of links
- Agent – Engaging the user in a dialogue

Our goals for the next generation of UI must take the best from all of these efforts, and deliver an incredibly compelling overall experience for *both* the novice and the experienced user. Some specific goals are outlined below.

2.1.1 User Interface Goals

2.1.1.1 Significantly simpler

This doesn't just mean losing a few controls. The web UI is popular and successful because it is not based on any high concepts. If there's a link, it's underlined, and clicking on it takes you to where the link points. Every user understands that. There is no right click or double click metaphor on the web. That doesn't mean that these concepts are wrong, just perhaps over-used or chosen as keys to fundamental actions (like double clicking an application on the desktop to open it). The value added by these more difficult concepts is clear – most customers appear to like context menus, and are happy to right click to get them. Customers do not appear to like a lot of windows, though.

Like the web, it means being able to pause halfway through a task and resume it later, or abort it altogether. It means always being able to go back and change things, so that no user action is committed until the very final stage.

2.1.1.2 Adaptable

The user interface must be able to take full advantage of whatever device it is rendered on – a big screen with cool graphics capabilities must be able to be utilized to the full, yet the same code must run perfectly adequately on PDAs, cellphones, laptops, etc.

2.1.1.3 Contextual

The UI must present itself contextually, so that a one-task function presents only the UI relevant to it (such as a book reader), with other UI facilities hidden but always available, whereas customers higher up the functionality scale might want to have easier access to more complex UI functions.

The mouse and keyboard aren't the only input devices. PDA's have caused pens once again to become popular, so our UI must incorporate facilities that take advantage of a pen if it is present

We must make the local versus Web experience seamless, so that viewing and editing Office documents feels the same – is the same – as viewing and editing web pages. Therefore, we must also adopt many of the UI principles found on the web, some of which may have significant impact, such as the mixture of content and control/navigation elements on a page. Clearly, the reverse is true – Windows can leverage the web, so that web pages can be used to add to the user experience in Windows, which means that integration must be much tighter than we have today.

2.1.1.4 Customizable

The UI has to be far more customizable than it is in Windows today. Tools must be provided to allow user interface to be created and edited as easily as content, and in some cases indistinguishable from content (so that forms, web pages and generic UI elements are all treated the same way). Additionally, customers will want to customize UI in other ways, such as adding their own annotations to it

2.1.1.5 Include Rich new innovations

This includes automatically adaptive UI that determines device characteristics, including form factor, and renders itself accordingly. If the UI designer can author once and know that his user interface will work across a range of devices, we have a win. So, this likely means that UI is itself declarative, allowing itself to be transformed to the appropriate set of UI elements and features when it 'lands' on a device. The UI needs to include mechanisms to integrate text, audio and video, so that one could for example have an audio tool tip as easily as one has a textual one. Each of these features must be easily accessible to application developers. Perhaps the UI itself incorporates real time collaboration features, so that the content of a control on a page could perhaps be bound to a URL.

2.1.1.6 Combine the Best of Windows with the best of the Web

We must be the owners and progenitors of a new UI style, "WinWeb", analogous to the Windows (and Macintosh) styles, so that a developer has definitive guidelines about how to construct the UI of a compliant application. The guidelines should include rich samples that show developers how to incorporate these features into new applications, and must provide adequate scope for extension. It is vital that such a style guide be an evolved form of the web UI style, and must add to it rather than changing away from it

2.1.2 Roadmap

It is not plausible that all of this can happen in the Millennium release of Windows. It may be that we can take some evolutionary steps there, perhaps a few revolutionary ones, and then really drive forward in Neptune. Take one of the most promising areas for UI innovation: Activity Centers. We should define the basic concept in Millennium, create a few to drive home the principle, and ensure that ISVs (OEMs?) can add their own. We may even be able to drive an industry akin to WinAMP skins in this area if a) we make activity centers compelling enough, b) we make it easy for ISVs to create and publish them, and c) we leave scope for them to do this.

So part of this will be to define the set of activity centers that we want for Millennium (which ought to be a subset of those we want for Neptune), decide which ones we can usefully farm out, and then promulgate the concept as widely as possible so that interest is created. But we also need to tie activity centers to Windows, so that they can't just be arbitrarily created and used on any system regardless of whether it's ours or not. This is hard, unless we create a UI framework that is easy, extensible and partially supportable elsewhere (so that, for example, a much-diminished activity center experience would be possible on IE4, 5

and Nav 4, 5 systems). Activity centers should also be extensible, so that existing ones can be customized and enhanced.

We also need to provide some of the other UI innovations in Millennium; it doesn't seem to great a technical step to have Agent presume the role of the Run option on the Start menu, for example, or to make sure that many of the pen and stylus based UI innovations from PDAs make their way into mainstream Windows.

Now is also the time to make some foray into the adaptive UI arena. Much of what I have said before tends towards a UI definition mechanism that is declarative and therefore fairly easily transformable. We already have some of the mechanisms in place to do this, even in IE5. With the XML and XSL technologies therein, we can create definitions of UI in XML which can be rendered differently depending upon the XSL transformation which is applied to them. So, if we were to define (and/or adopt) an XML-based UI definition language (c.f. RCML in NT and Netscape's XUL), we can both create and enable the creation of device-adaptive UI, at least to some extent. This would be a pragmatic move in Millennium, and it allows for far greater advances – such as excellent tools support – in the Neptune timeframe. It also allows us to define a schema for UI which we can publish, both as a style guide and as a defining mechanism.

In the longer term, we need to determine how successful the smaller UI enhancements in Millennium turned out to be, and leverage those that are clearly working. Others should be discarded quickly, as we did with channels and the active desktop between IE4 and IE5.

If Neptune can begin to turn the UI into a collaborative mix, that will be interesting. This is the idea where a control/form/page is linked – two-way - to live content. Then, as we work on the content as a group, so we all see the changes. We'd want to have some very clever technology (e.g. DAV enhancements) behind this so that one user's changes aren't immediately destructive to another user's, but we fundamentally know how to do that.

Perhaps the biggest challenge for us, which needs to be resolved by Neptune, is exactly how legacy apps (that is, those we are using today!) fit into this framework.

[Don't quite know what to do with the AC text below.]

One of the most promising areas is Activity Centers.

We need to think about answers to the following:

Can ISV's create new Activity Centers? We won't be able to do them all for Millennium or even for Neptune. Perhaps we want to partner with folks to write a couple of Activity Centers. Maybe HRD writes a couple for us or there becomes a thriving market for Windows Activity Centers that customers buy. There is a broad list of key task centers for consumers. They include games, photos, music, video, communications, shopping, personal finance, and home productivity. If we decide to extend this metaphor to business, then there are probably others as well. We must define the core services and business models that enable others to create them.

Which centers are at the heart of the Windows Experience? As we think about our Activity Center investments, we need to think of them on a continuum. Some we will deliver in Millennium and we will deliver more in Neptune. We need to be able to prioritize which ones we deliver on.

How does "legacy" content fit in with the new paradigm? We need to decide if we are just introducing another new concept, or if indeed, all applications are now invoked from activity centers.

What makes a new application a great addition to an activity center? A clear, compelling message to ISV's is critical.

2.2 Attacking Complexity

In addition to defining some new areas of excitement, like Activity Centers, there are some other areas where we need to continue to enhance the user experience. I will mention just a few. Our PC Health initiative means cleaning up the "error" experience, both preventing errors, and truly helping to fix problems when they occur. We need to continue to leverage Windows Update. We need to once and for all eliminate the problem of "DLL Hell" – this means delivering COM+ Deployment, a.k.a. Fusion, in timely manner.

2.3 A Big Bet

Talk about speech/hw/vision here?

3 Establishing the Windows Schema

The idea of schema is straightforward: it allows us to make intelligent use of data, so that greater integration is possible, data can be put to more uses more easily and more data becomes accessible (through late binding). Once we know that data exists, we want to use it. If we know nothing about that data, we're pretty much hamstrung. If, on the other hand, that data has an identifiable schema, then we can make intelligent use of it. For many of the kinds of data we would store on the typical Windows machine, we'd also have the appropriate set of schema definitions stored *as a part of Windows*. There will be standard schema for a number of common things; where these things are truly generic, we should define the standard. But each standard must also be extensible, and we must make many of those extensions. We ensure our ability to add value by ensuring that we are masters of the schema. We can move away from complex object models, complex APIs and proprietary formats, replacing them all by schema, but we only get value in doing so if we effectively own the schema. Of course, we'll publish those schema, and perhaps some will be totally standard, totally available for general use. On the other hand, many schema will be private to us, legally owned by us, and indubitably controlled by us. That way, there is a series of natural leverage points for our products: if you have Windows, you'll want to get Office; if you have Office, you'll want to use our services; if you use our services, you'll want to run a CE-based PDA, etc.

We have a strong tradition of owning the platform by owning the ISVs, because they write to our API. Now that API advantage is being eroded, and it's actually highly unlikely that another foray into the API world (c.f. WFC) will win us any more customers. On the other hand, if we show customers, developers and system integrators the brave new world of developing, deploying and using Windows systems where applications make use of the standard set of schema we provide in Windows, we effectively move the API battle to a different front – the schema *is* the API. Don't be fooled – we still have competitors, as both IBM and Oracle understand this point. But it is not clear yet that everyone does, and of course it flies in the face of Sun's Java strategy – hence Sun's constant and consistent scrambling to tie XML and Java inexorably.

In the long term, then, we must ensure that we have defined schema for all objects and events of generic use – our systems (this ranges from schema for cards in a PC through schema for UI generation and schema for management events, to schema for system calls), our applications (so that Word's object model, for example, is supplanted by its schema – one can always get to the OM from the schema, if necessary), and our services (we'll define a name, an address, a hotmail user, a credit card, etc.)

Once we have schema defined for everything interesting, and applications that make use of these schema, we can make data more usable, more accessible. For example, Microsoft software can pull data out of web pages from Microsoft services – and make semantic sense of it. This is a hard thing to do in the world of HTML without prior knowledge of the page. Pages change, so systems relying on certain layouts are fragile. Further, consider how useful Office can be when it can make sense of what you type – it could present a list of actions one can do on an address, for example.

When devices have schema attached to them – printers, light bulbs, refrigerators – they become services that can be interrogated and driven through their schema.

In order to make schema truly useful, definition alone is not enough. Schema need to be customizable and annotatable by customers, they need to be queryable, and of course they need to be easy to find as new items appear on a customer's computer. In addition, we need to provide comprehensive systems for creating and editing schema and transforming between them. The transformation step allows us to take data in one schema and convert it to another automatically, so it works most effectively if we can use the schema to drive the transformation, rather than relying on schema authors to provide their own transformation information.

Therefore the actions we must take include creation and publication (on a universally accessible web server) of the most critical sets of schema, ensuring that we have – and keep - IPR where necessary, making sure that Windows, Office and the browser are all totally schema-aware, creation of tools to create and manipulate schema, make sure that our programming languages interact with schema (e.g. by creating an appropriate object model, late bound when necessary, on consumption of a schema), put in the public domain non-IPRd subsets of those schema we want to proliferate generically, and leverage the hell out of them in our UI and functional innovations over the next few product cycles. Each cycle must evolve from those prior to it, so that Office 10 and Millennium do the basics and show the way; Neptune and subsequent releases of Office, Back Office and SQL Server consolidate our lead. The first release of PKM needs to include ways of searching on an item's schema, making the whole search experience that much more fulfilling.

Schema must give us standard ways of describing objects and events. We must have services on top of these things (logging, query) to make them useful.

a) What generic tool do I use to browse management information? How does this relate to MMC?
b) Show me how I program against our management schema - do our languages see the URL descriptions automatically?
c) What leadership is there to suggest what it would mean for Outlook to support our scheme approach?
d) Where are schemas stored? How are they browsed? How does this related to repository?

Although its nice to have standards for synching data I want to have a Microsoft schema for calendar/personal/contacts that is NOT part of any standard. I want us to let people customize the schema and NOT have that part of any standard for replication. I want all of our devices to share the rich standand schema and the ABILITY to customize schema. We need to get schema and schema customization into our PDA and Outlook strategies as part of cleaning up the addressbook/wab/pab/directory mess we have right now.

I want these to be key key proof points for why someone who uses OFFICE should use our PDA and SERVICES.

This is taking a lot of steps but it is key to succeeding. We only want the commodity standard to go a LIMITED distance.

One of the great values that "local processing" power and our software can provide is dealing with information the user in presented and helping the user make connections.

For example being able to take a name that shows up in a document I create and give me all the actions that might relate to that name.

We need to have a practical schema to belongs to us (other browsers or OSes or productivity software can't copy it without a license - we need legal to make sure it is protected).

Web sites can use it and will be encourage to use it.

This fits in pretty well with the idea of self describing objects and activity centers.

To be specific whenever a BOOK, SOMETHING YOU CAN BUY, GROUP NAME, COMPANY NAME, LOCATION, PERSON NAME, ADDRESS, TIME, APPOINTMENT or other common object shows up we should have descriptive information that we help provide and ask to have provided.

There is a synthesis here between the idea of SEARCH, SCHEMA, FACTOIDS, XML and ACTIVITY CENTERS.

To make this happen a number of elements have to come together:

a) Someone has to define these simple schemas and get them protected and figure out how to get broad support.

For example all our email clients ned to be part of this. Getting them to a common schema on these things is a basic
thing we need to do anyway. I include PDAs in this.
b) Windows itself (in the browser code) needs to support these including some "Autoformat/factoid" code.
c) Office has to allow for "objects" to be expressed in the XML hidden text and have Autoformat/Factoid code help recognize objects
d) We need to decide our role in creating the services that these things connect to. This is a HUGE underexploited asset that will bring up all the classic questions of how hard coded is it (answer: TOTALLY - no OEM change - connected to an MS URL and then redirected to the partner in that country/area)

Use of NL and Schema in Office, PKM, and MSN
First, we are working on the use of both NL (logical form ala Truffle) and schema to help improve search results in the PKM server and the Office client in the O10 timeframe. As an initial use of schema, we hope to use knowledge of the exchange/outlook schema and perhaps the schema of at most one or two other products to answer NL queries that span structured and Full Text data. For example, "Show me email from BillG concerning MSN search" would return this email chain.

The starting point for this schema will be a subset of the conceptual schema for Person Places and Time that was worked on with the schema team in DAPD (Keith Short). In particular, we will use those parts that deal with the actual schemas in place today for objects in the systems of interest. We don't at this point propose to own the standardization effort for a conceptual schema across MSFT.

Relatedly, in this timeframe we won't have much of an authoring model for people to add new data sources mapped into existing conceptual schema. This will be done via XML that will likely be hard for "mere mortals" to deal with.

We will support the use of factoid analysis and normalization to improve query results. So, "1/7/99" can match "during January".

Another extremely important Office 10 feature is the use of Factoid analysis to allow our applications to highlight the associated text and users to right click off to appropriate functions based on the type of the factoid. This will be a *very* cool feature. Additionally, we are looking into whether or not the factoids can be exposed to custom app builders through the Office object model.

We are a little late in the PKM and Office 10 planning process with the proposal of these features. Nonetheless, we expect to have the above plan firmly in place and hopefully accepted for the O10 timeframe by the BPG product teams this month. After that, we intend to engage more fully with MSN to see if any of these techniques can help there as well. Of particular interest will be how to use our NL technology to integrate with and/or replace RealNames.

4 Manageability

[I still can't get this section to fit. Maybe it's a sign that this really isn't in the top3?]

UpnP?

Distributed Events
Policy
Enterprise Event Log
Instrumentation [WMI, etc.]

Make it more appealing to include lots of MS infrastructure?

5 Clear developer message

We must have a clear developer message, which will help us enormously in competing with Java. Part of this must include a cohesive presentation strategy (viz. forms)

5.1 API

WFC vs. Win?

5.2 Storage

Neptune will be the first chance we get to deliver our improved storage system, based on the Platinum store. What advantages do we create by deliverlhg this as a native feature? (e.g. Faster?)

I wish we could get team server and Office 10 to share a common vision but a lot of things stand in the way of that.

I think people who do websites want or big documents want a team server like capability.

We should insist that all teamserver forms fit into our new programming framework and as much as possible they do their apps in that framework.

- Building bridges between our two stores.

5.3 Forms

Some text around forms.

Here are some goals for an ideal forms strategy.

For access, take the capabilities of the native forms and datapages and bring them together in one forms approach.

For VB, take the capabilities of the native forms and the triedit forms and bring them together in one forms approach.

Have the ability of "Windows Terminal Server" - that is rich rendering can be done on the server and shipped down to a client - this feature cannot be based on some commoditized open standard. Our rich rendering cannot be given out to a standards effort. We have to have a presentation asset.

Support rich UI. It CANNOT just be the browser.

Have the ability to let you design "down level forms" for HTML 3.2. Having levels in between say IE 6 and HTML 3.2 is not good.

Become the forms approach we use in the Neptune shell.

Be based on XML.

Allow for migration of MFC users?

Support the WFC apis in the form package? The developer group cancelled their VSFORMS effort.

Support document editing on the same surface (Netdocs). This is a very demanding requirement. It also means we need to have a 3 level strategy ideally based on one code base. The 3 evels are: 1. Free browser/Standards based 2. Windows browser 3. Office browser (the version with the editing capabilities). I don't want #3 and #2 to be the same because it makes it too asy to build Office capability.

Allow for migration of GDI/User developers. {realtionship to Hwnd/Window management/RC files} (activeX controls?) Can the compatibilty stuff be done as a layer independent of the forms package if we do it the right way?

The support for this forms engine on WinCE devices is an interesting problem but not the most critical. People talk about expressing UI in a more abstract way to deal with different sceeen sizes. It is unclear if our Forms strategy needs to include this.

A forms package that we get a nice end user development tool for that lets us compete with Notes. Forms3 became a dead end and it is hurting us versus Lotus.

We can't really promote our new URT based platform without a clear message about FORMs so we have to get this done to be at the starting line.

By default we are pushing people to use simple HTML which is promoting the move away from our platform

I can see different strategies.

MS-PCA 1367277
HIGHLY CONFIDENTIAL

A. Try to use a common Trident code base to be all three things a) Document surface, b) HTML displayer c) XML forms package. We seem to have an overload right now where we get (B) but messy (a) and no (c) from the one code base.
B. Keep Trident as HTML focused. Fork off a version to be the XML forms package splitting the world. Try to use conditional assembly for the document display.
C, Keep Trident as HTML focused. Create a new code base independent of Trident for the XML forms/document surface. Use some of the Cooper/Peters or Tony Williams stuff as a starting point.

Maybe I am trying to do too much with one package however I see scenarios where people want to do UI/Forms and mail.

5.4 Data Access

6 Building Internet Services

Increasingly, MSN will play an important role in how an end user perceives the "Microsoft Experience." MSN services must enhance the experience our customers have with all of our products.

6.1 Communications

Neptune must help us prioritize some of the variety of communications technologies that we've delivered – in particular, we must rationalize our multitude of email clients, and we must decide what the future of NetMeeting becomes, in relation to Chat, IM, etc.

I think our current strategy is as follows:

1. Outlook express. It will be expanded into scheduling because it has to for JUMP integration and competition. It will continue to be free and simpler to use than Outlook. It will continue to have its own store.

2. Neptune. The shell will be integrated and include email handling and file management operations.

3. Outlook. We will spend 2 years improving this for Office 10 but it will not have schema, it will not use standard forms
(because we are a mess on those).

4. Netdocs. A strange part of our strategy - a UI of its own that isn't different enough to want to switch to but different enough to relearn.
Kind of like a lot of our developer strategies.

5. Outlook Web Access. The best of these for many reasons - it can be built on our server strategy. It can be the show case for using
client functionality when it is there. It can make our Hotmail/Exchange strategies come together. It does require depending on getting
our platform stuff good enough to make this code run on a client and make it better.

I am not including the email client you have on the PDA. I am not including the email client you have in Project (with the funny web UI stuff).
I am not including WebTV email.

We cannot afford this fragmentation. I want to see us get behind some combination of 4&5 and get the Neptune people bought into using a subset of this for their shell.
I do not think Outlook client as currently written is a code base we should invest in AT ALL.

It doesn't let people easily update their mail client. It doesn't support our forms strategy in an integrated way.

I also think the Outlook express client is a could base we should stop investing in.

We should put the resources on Outlook Web Access and solve the problems that it has and use that code base for:

a) Free email client

b) Free email service
c) MSN email client
d) PDA email client
e) Office email client

There are a lot of pluses to this approach that far outweigh the negatives.

I understand that even shifting resources we may not solve the Offline use or become as rich as Outlook by Platinum. However by Office 10 timeframe we should be able to solve those problems and have a real foundation for the future.

Lotus has very very few resources on their email client because they have a rich framework they use and write script. We have to pursue the same apporach. Otherwise we will move too slow and not give customers the flexibility they want.

This should be the first part of Office that can just "update" itself as part of the experience and be roamable as well. This fits in with our service plans as well.

Communication and community - Timely communication is one of the most deeply desired and highly valued capabilities in any group, where we use communication to establish communities for shared gain and for protection. Phone calls, pagers, emails, and real-time messages all represent common modes of digital communication, and all can be used in a group. While a personal inbox and private individual-to-individual communications will always be important, there are also numerous interesting scenarios that involve group communications and an open social context. Family activities, for example, such as scheduling, coordination, record keeping, and finances, can all be usefully performed as a group. It is trust in the quality and veracity of communication that enables communities like the family to form and flourish online.

6.2 Commerce

6.3 Megaserver

Roaming access to network-delivered services is a liberating experience: people bring laptops to each other's offices and manage calendars, review email, and exchange files face-to-face. People bring palm computers into living rooms or public stadiums and have services beamed to them. Meetings in conference rooms or in convention centers will be more productive as well when everyone is online at the same time. There is no reason that experiences at work, at home, and in public would not all be enhanced, given socially apt devices.

Users and their virtual agents prefer to roam - Devices roam against a fixed network background that includes things such as payphones, AV, wireless network, power grid, telephone net, cable television, and even public computing kiosks. This background also has movable elements - pdas, cars, cellphones, and remote controls to name a few.

Clients, storage, and even peripherals all might move - In the world described above, devices are not dedicated to a single user. To make matters worse, with the emergence of easy to move, ultra-reliable, appliance-like peripherals, peripheral devices no longer have a static one-to-one relationship with the computers that are using them. These two factors combine to mean that driver models and support for dynamic reconfiguration are even more critical than they have been in the past; the requirements for reconfigurability are often directly opposed to the requirements for stability, responsiveness, and predictability.

7 Summary

We have a long way to go and a short time to get there ...

Crazy brainstorm – what abt a Matrix – what abt one for short term and long term? Something like:

	UI	Schema	Dev Message	Manageability	Internet Services
DavidCol	Owner	New UI exploits it	Windows initializes, uses and distributes key infrastructure	User experience is managed; desktop roams, etc.	Builds UI to exploit
BrianV	Biz windows supports same new UI metaphors	Management tools, logs, etc. fit into framework	New apps are inherently "manageable."	Owner	Provides tools to manage
DavidV	Tools and classlibs to help support new UI metaphors	Tools to build, extend, and map schema; tools for apps to exploit	Owner	New apps are inherently "manageable."	Provides tools to create
SteveSi	Office is the showcase application	Owner	Office supports model	Office applications and documents are managed by this framework	Exploits key services provided (email, search, etc.)
JonDe	MSN provides the backend services for new UI metaphors	Provides services for search, etc.	Uses new technologies	Key client of new features	Owner