From: Chris Jones
Sent: Tuesday, July 20, 1999 12:47 PM
To: Bill Gates
Subject: FW: thoughts on (and questions on) the platform...

i thought this might be interesting for you given our conversations today. some stuff we are thinking about for neptune. no code, just thinking, but we are going to get there.

– chris

-----Original Message-----
From: Chris Jones
Sent: Friday, July 09, 1999 9:31 AM
To: Jon Thomason
Subject: thoughts on (and questions on) the platform...

to set context, i've been thinking about how we move forward w/ neptune. one thing that i am wondering about is our new client side platform.

first i'll talk about customers, then features of the platform that i think we need, then some views on what we could do to build them. this is all braindump and i'm sure that you and others have thought about most of this. i am writing primarily to give you my perspective and views.

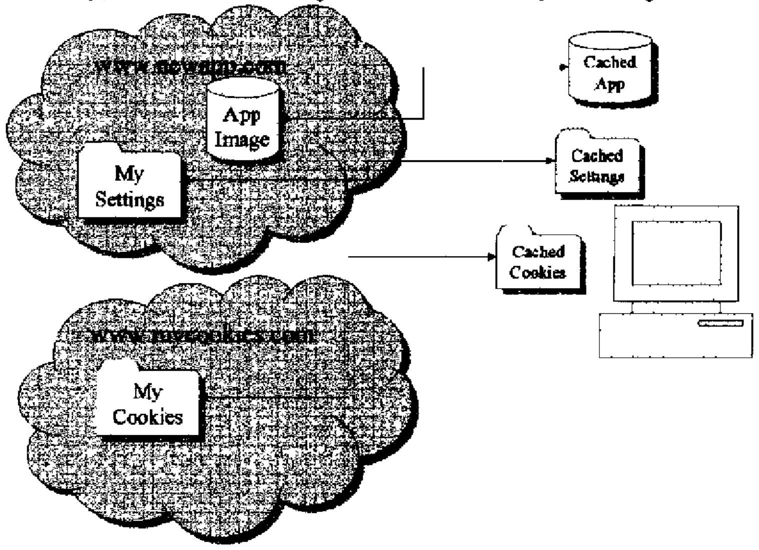
customers (or why client side code)
one question we have to answer is *what is the opportunity/competitive advantage to writing client side code?* our message must start with this – we have to be able to articulate the *advantages* of writing client side code. these should be things that are either *200% better* on clients or *only possible* on clients. my list is (frighteningly) short, would be interested in seeing yours:
a) offline (or local) data. because internet connections are intermittent, having offline or replicated data is essential at least for the next 3-5 years, and probably longer. i would put all existing win32 apps into this bucket -- they all want to work with replicated internet data.
b) editing of anything. picture editing, document editing, music editing -- these take up a lot of cpu time and have the benefit that they require both local data and local processing. with the continuing emergence of digital media i think there will be an increased market for these types of applications.
c) high performance graphics and interactivity. games is certainly the biggest example of this, multi-media or consumer titles also fall into this category. again, increasing marketplace, especially if we are successful with x-box or another windows-based game console.
d) rich, high fidelity display. outside of the categories above, this is the weakest of all. i think we can only sell this as an advantage if we remove some of the problems with client side code (see below). maybe you could convince web-based solutions or applications to do this, but it would be frosting and not core. that is not to say frosting is bad, but it is not sufficient to get them to bet.
e) speech, nlp, or other forms of input. i think we are ahead of our time here but this will be big someday.

why pick the microsoft platform?
once we've convinced people that they should write client side code, the next question is *why will people choose our platform?* this is the second part of the message -- once you have articulated the opportunity we have to talk about why windows is the way to go. here is my list of things that we should deliver and provide, again we should get to one linst and agree:
a) zero-friction deployment. a must, this is the number one blocker. has to be a no-brainer for users to get the application. this must work for both "page-based" applications and also traditional, win32 apps (like games).

Plaintiff's Exhibit
6607
Comes V. Microsoft

b) digitial rights management/protection. i think this is going to be huge, for copyright of software, but also for pictures, music, etc. this will enable a new type of content application that we haven't seen before. also will enable piracy protection (tracking of who's using my software) and upsell.
c) automatic replication/caching of data. this again must be easy for traditional apps (like winword) to use as well as new generation apps (like money).
d) service built in. i think service includes settings roaming (word follows me around), automatic update/hot fix (for the latest changes), connect to help/pss (on the web), upsell new versions or add-in's (again on the web). all new apps should come with www.theapp.com site that supports, roams, and provides service.
e) easy to use/help built in. we can help isv's here, and this will be important. i am not sure what our platform message is here, but this means web-based navigation, text and commands built in, a new user model for presenting information.
f) great tools. we have to have a tools story, this is key. whether we build the tools or visual studio does or a 3rd party does, they must be available and sim ship.
g) "cool." we should not under-value this, our platform must be cool to folks, like 3d was with win95.

so what's this new app model look like anyhow?
next step, what does this thing look like? here's a picture to get started...



part 1: app basics
i'll start with things that i think every app has, then talk specifically about "page-based" vs. "window based" solutions. first, i assert that every new app has (a) a web site, and (b) a cookie. the fallback is that the app "web site" is really just a cd-rom that is read only, but it should still have a "cookie" that is uses to identify the user on the machine and store information about them. why? four features:

1) depolyment/purchase. the web site is used first and foremost for deployment, update, and "refresh" of the application. i think all new apps are sold and distributed on the internet by default, where the cd and retail are distant seconds. so to run winword i navigate to http://www.winword.com, enter my credit card, and off i go. the nice thing about the cookie is that the winword folks can use it to track how many machines and users are running their software, and do some basic piracy enforcement. so each winword user gets a unique cookie when they purchase, winword validates the cookie when connected to the internet, and either shuts down or informs the user if the cookie has "expired" or if more than one person is using it. requires:

- zero deployment for apps. fusion team, has to span existing win32 apps as well as web-based solutions. includes url's for naming everything, auto-cache/install of apps.
- ip protection. story on how to mitigate piracy, maybe using cookies.
- one click shopping. would be nice to have the credit card stuff bundled up so there was one click shopping for these items.
- cookie api? some way for all types of apps to request and use a cookie, independent of the browser.

2) app health/updates. secondly, the web site is used for incremental (on demand) updates, and reporting gpf's. each app will want to run a "health service," where they register their site for gpf reports, and then neptune posts the information to their site so they can track, monitor, and dynamically update their app. they also host additional help updates and troubleshooters for common calls. requires:
- gpf reporting. way to register your app and the gpf's you are interested in.

3) settings roaming. third, the web site is used to roam settings. note that in the picture above i assume that there is a place where my cookies are stored and synchronized, so from any machine i can log in and replicate my cookies, then pass the cookies off to the "app site," which then provides me with my settings (and caches/installs my app as necessary). requires:
- cookie roaming. api's for this plus service to support. probably want to have this with msn and other services (aol, yahoo, etc).
- settings api's. way for apps to replicate their settings locally. related to com+ stuff i'm sure.

4) new purchase/add-on's/upgrades. lastly, but maybe most importantly, the app vendor now has an ongoing relationship with the customer and can use that to promote updates, upgrades, and add-on packages. no new requirements, except maybe:
- notification service. way to promote/provide messages to users.

so part 1 enables all types of applications, but mostly existing client side applications, to add a service to their application, reduce support costs, generate new business, and reduce piracy.

part 2: data replication and sharing
the second part of our new app model has to include data replication. there are two types of "data replicated" apps.
a) traditional publishing. think about moving from my documents to mydocuments.com. pictures, music, video/movies, letters.
b) data-centric applications. have a database plus rich views on the data set. mail, money, schedule -- all of these fall into the same category.

we should have a terrific message for both of these isv's. personally, i think the first is more important than the second, mainly because we will have more success. here are some things we have to do:

1) mydocuments.com. every neptune pc should come configured with mydocuments.com, the folder or place in the shell by default that an app can save information to and it will be published to the web. we should automatically replicate this information out to the web and abstract away the bandwidth.

2) file => open web site. opening over http should be standard, http url's should exist everywhere in the shell/apps and be the primary way apps interact with information. richer file open with search across common sites/documents i've been to.

3) publish/manage collections. i think this is enhancements in the store/file system so that apps can "save" collections of objects (from complete web pages to collections of pictures/images) and then the user deals with them as one "thing."

4) lightweight database. this in my mind is a pri 2 or pri 3. we should enumerate the apps here but i think we are over estimating the number of people who will build applications like this. this is one of the reasons i wonder why we picked mail as the stretch neptune app -- how many isv's are going to build a mail client?

part 3: integration across applications
third part -- integration across applications. think about this as the extension of registering as the default mailer/scheduler/etc for the shell today. isv's want to be able to (a) promote their application for the things it can provide, and (b) leverage other installed applications for the things they can provide. i think about this as "service providers" for common types of information. we may have to write "device drivers" for the common applications to make this work well. services include:

1) buddy list/chat. any application should be able to "call" our service to start a chat or collaborative game or collaborative viewing on a particular topic. e.g. doom should be able to say "find player" and it should launch whatever the buddy list service is, find the right player, and return that player handle to the game. video chat, voice chat, and text chat should be a part of this. we should write "drivers" for icq, aol, and other services to support this, and we should enhance them.

2) mail. simple mapi, exists already, nuff said. again in addition to the api key is to write code *ourselves* for the common mail providers (like aol, yahoo, etc) so that we provide enough coverage for isv's to want to use our work.

3) calendar/schedule/address book. same idea as mail. i'm sure there are more of these.

4) new service provider we didn't think about. we should build this generically enough that it can be extended by 3rd parties. think about this like the file extensions database but for services.

as part of this, we have to ship "default" applications ourselves (these should be the canonical shell apps) so we test/dogfood our experience.

part 4: ui integration and coolness
this is last on my list, but it is critical frosting that we have to have. this is what wraps it all together. i lump the things we are doing in trident into this space, text flow navigation, etc. i put common controls here as well. there is value here (e.g. new user model, easier to use, simple navigation) but the motivation will really be that isv's want to look like us. we should have a story where it is easy for an existing win32 or web isv to get this new look.

closing thoughts
whew. so a long winded braindump. here are some additional things to consider...

points of light. this is what makes a neptune app cool. from above, here's my list:
a) supports zero friction/zero cost deployment and incremental update/fix.
b) runs a service to support/upsell/deploy the application. (think application "cookie")
c) supports mydocuments.com, and open/save from web locations. uses neptune file=>open to find/save files.
d) integrates with neptune services (instant message, mail, calendar) where appropriate, or provides a neptune service.
e) follows neptune ui style guide. new page-based navigation.

microsoft/msn business opportunities
we should think about running a store with msn where users get a subscription to a set of apps. the msn team works with 3rd parties to launch the neptune service, where for $10/month you get access to 5 or 10 applications. we pay back the isv's, get more msn membership, and provide a new biz opportunity. another carrot.

thoughts?
i am not sure we need to change anything because i don't know what the app architecture is. i am going to dig in and try to figure it out. interested in your thoughts on this -- specifically:

a) do you agree with customer list and customer problems? why or why not?
b) how do you think we will differentiate the microsoft platform?
c) what are the features we should be building and are we staffed to do them?
d) what do we need to do to get in the feedback loop with isv's?

thx – chris