Microsoft MEMO

TO: Gary Ericson
David Greenfield
Nick Holt
Barry MacKichan
John Morey
Rob Shurtleff
Allen Wells
Jeff Weems
Chris Zimmerman

CC (cover letter only): Richard Bowen
Eric Candell
Bob Matthews

FROM: Bruce Burger
SUBJECT: **Laser Visions**
DATE: March 26, 1990

Attached are the four "vision" write-ups that have been done for Laser v1. The cool name for each is as follows:

- MicroWave – A product that is very tightly integrated with the Windows and PM shells
- Groupshot – A product that helps people work together
- Day-Timer – A product that you think of as your personal assistant that helps manage your time
- (Nameless) – A product that is highly customizable to support various office scenarios

Please give any comments to me as well as anyone else you think would be interested.

To summarize status briefly, we are most seriously considering MicroWave and Slingshot (as currently specced with various simplifications that David and Richard are working on). But thoughts on anything are invited.

Bruce

MICROWAVE

File : c:\wzmail\vision.fld
Messages : .

1
From bobm Tue Mar 20 14:22:44 1990
To: barrym brianb bruceb chrisz darrenr davidgr ericca jeffr nickh
richardb
Cc: darrylr davidco lisacr tandyt
Subject: EMail, DocLib & Shell Integration Thoughts
Date: Tue Mar 20 14:11:36 1990

This memo describes some thoughts I've had on what it would mean to try to maximally leverage MS's strength as a systems vendor in our WorkGroup strategy. This is just one of many possible product visions for our first serious WGA product, but I think it's an interesting one, because it fills a significant gap in a way that MS is particularly well positioned to do.

It's pretty sketchy, and the technical principles in particular need a lot of scrutiny. I think you'll see a lot of ideas that have been tossed around before, and discarded for one reason or another. I think maybe some of underlying assumptions for those reasons have changed, such as reasonable timing for various software components.

Hope this stimulates some good ideas.

Bob

****

GOAL: To assert control of the OS Desktop Services market, by revamping the shell, defining a few key extensibility mechanisms, giving away some simple desktop service components with the OS, and creating a market for more sophisticated desktop service components such as advanced mail and document management.

BASIC COMPONENTS OF THIS STRATEGY

1) Target Win 4.0 + DOS 5 as our required platform. We'd have to figure out what to do about OS/2. Key elements are: we need a new shell and ability to rely on a filesystem that supports EAs and a few new notification hooks.

   We'll need a codename - maybe LaserVision or MicroWave.

   Just kidding.

   Sort of.

2) Theme for the new shell is extensibility and taking maximum advantage of the new filesystem. To be extensible we probably have to kill MDI in the shell. I would also argue for much more liberal use of EAs, getting rid of the Program Manager (using the File Manager instead), and starting to head in a more program-free, object-oriented direction.

   An example of extensibility: subclassing directories as container objects. Things like InTrays, OutTrays, enhanced document managers would all be

subclasses of filesystem directories, and we'd actually use directories as our primary storage medium. Each directory would have exactly one viewer, corresponding to the subclass-type of the particular directory. In some cases parallel information would be kept for special needs like fast header display, indexed queries, content-search information. FS hooks would let this parallel information be kept up to date.

3) Give away a real simple email client as part of the Shell. It would plug directly into Spitfire (if present). It would also work in a super-simplistic shared filesystem mode if Spitfire weren't there - no admin required, little or no security provided.

   More details on this client below.

4) Announce to the world that this is a significant first step to us having an IOS architecture that the world can start plugging their apps into.

   Our guideline would be to make sure existing apps worked unmodified, although perhaps with some loss of function. We'd also want to make the work to modify apps to plug into this fledgeling architecture as simple as possible.

5) At or very near the time of MicroWave introduction, we'd also have some other key value-added components ready to sell. These would be:

   - o Enhanced EMail with great personal message/document management
   - o Document Library / Bulletin Board
   - o Personal Calendar / Group Scheduler
   - o Thunder or Silver or whatever would be designed to let people easily construct this type of shell-extension application.

A LITTLE COMPETITIVE ANALYSIS

As a response to OfficeVision, New Wave, WP Office, etc:

These systems all exist because there is a perceived need for OS and UI extensions beyond what Windows provides, and in fact they include some nice ideas. They all suffer because a) there is an intrinsic seam between what they provide and the OS filesystem and UI, and b) they fight a major uphill battle in getting ISV attention

As a company, we can kill these other products by taking some important first steps toward offering some of the most important OS and UI extensions within DOS/Win and OS/2, and beginning to align some of our other offerings with this model (e.g. document library).

As a response to existing email vendors like CC:Mail, daVinci, etc:

This approach represents a one-two punch at our existing email competition. By giving away a simple client that works either with SFS or Spitfire, we erode heavily into their low-end customer base. By significantly changing the rules on what it means to be part of the "office environment" we force them to adapt (at significant development cost) or look like anachronisms. We are creating millions of potential sockets for advanced services, where we will have both a time and a name-brand edge over the competition.

As a response to Notes:

Notes is scary. The one intrinsic advantage we have over Lotus is our position as a systems vendor. We can make our response look like a more natural extension of the OS, and in fact we can claim that significant functional components would properly BE part of the OS.

We position ourselves as more open than Notes (handling documents from all sources), and as more architecturally aligned than Notes. And from the desktop UI perspective, Notes will suffer the same challenge as the email competitors: they will have to do some significant work to catch up and have their stuff look like a natural fit to the desktop.

HOW THIS WOULD GET BUILT

Windows group would build a new shell for Windows 4, supporting the kinds of extensibility (e.g. directory subclassing) I mention elsewhere in this memo. They might want to use some of our code, if it made their lives simpler.

DOS group would build DOS 5 filesystem to support reasonable performance EAs, and would provide the necessary notification hooks to allow the more powerful subclassing ideas to work even with non-MicroWave- aware applications. Might require a little bit of work by OS/2 and LanMan to support similar or complementary hooks.

WGA would write the simple, bundled email client (InTray, OutTray, super-simple transport, message composition, etc.). It would be based in part on other work we're doing, like our UI framework and, to some extent, forms.

We'd also build the more powerful email client and personal document management code, both data storage and UI. Exact features TBD, but would include great message management and customizable forms.

Our goal would be to also write and include Calendar, time permitting. Might also include other modules, e.g. PIM. We would also help design the various protocols that let different modules work together well.

BrianB & co. would write the Group Document Manager back-end. We'd write the front-end for this in WGA.

Programmability (EB, Thunder, Silver) is an issue. My initial thoughts are that we should concentrate on providing durable APIs, but to not try to support internal programmability or customizability. This will be a nice product strength when we do it, but there's just too much uncertainty about DABU's plans and schedules for us to build a reasonable dependency at this point.

THE BIG RISKS

Risks #1, 2, 3 with this approach are that we get caught up spending too much time figuring out what the perfect UI and architectures for this stuff would be. I think it's doable in a 2-2.5 year timeframe, but only if we are super-creative in figuring out how to keep development from being spec constrained. We'd have to keep very clearly in mind that our prime directive is to take a pragmatic first step, not to define the grand,

all-encompassing UI and Architecture of Doom.

The other main risk is that some of this turns out to be not possible, not good enough performance, or gets snarled in OS-release synchronization.

****
Some initial thoughts on EAs, the filesystem, and how this might all work:

Consider the InTray. It's a directory of filesystem objects that is special in that a delivery process puts things into it. Delivery attributes like sender, suject, submission time, etc. are stored as EAs on the objects in the directory. A view on the InTray is basically just a specially formated and sorted DIR listing, showing a particularly helpful set of attributes.

An "email message" is just a file of type "email message". When opened, from the InTray viewer, that causes the email form viewer to be called up.

A message with attachments can be represented as a special class of directory, where the directory itself represents the message (and carries the delivery EAs) and the message body and all attachments are files within the subdirectory. In some ways it would be cleaner if the message body could be represented as part of the directory object itself (rather than as a file within the directory), but that's probably too significant a change to the FS to ask for.

There are some sticky issues to be resolved, around the distinction between filenames and "subject" attributes. For email messages, the filename isn't very important, and it's not reasonable to expect the subject to be unique within a directory. Since we're providing the message viewer, I'd propose that we generally show the subject rather than the filename, but that the filename be generated in a way that is reasonably readable and mnemonic - perhaps by concatenating sender, send time and subject. That way, in the rare case where the actual filename does appear, it won't look too weird.

Other document types are harder. I'll assert that it's important that applications, including existing apps, be able to open files in the InTray, and perhaps those that are attachments to messages, by getting directly at them through the filesystem. In this case, I think that preserving the original filename is very desireable, the meaning of Subject is less clear, and requiring unique names within a given directory is probably acceptable. We might have to doctor filenames slightly in cases where delivery of a document would create a duplicate in the intray.

By the way, it may be that applications will be classified New or Old, where New apps are more oriented toward presenting documents by subject/author/other attributes than toward filenames. New apps would be better geared toward dealing with new container types like InTray and Document Library. We might be able to dodge the uniqueness issues for such New apps.

The fact that the InTray (in this example) is an "email viewer" class of directory object is stored as an EA of the InTray directory itself. This tells the shell what viewer to call up when the InTray is opened, and may also tell the FS what hooks to invoke when the contents of the InTray are changed.

We'd add some protocols so that the email message viewer could find out

information about next & previous messages, etc - we could still put command buttons at the bottom of a read screen. Since the protocols would also be available to other document types, this could be generalized to allow stepping through documents, if the viewing applications wanted to support this.

The FS notification hooks are important to our ability to provide more advanced functions. While the bundled InTray would be a pretty simple-minded implementation, we would want our non-bundled email InTray and personal document manager facilities to do significantly more. For example, for the InTray we might want to support active filtration and sorting on the InTray, better performance with more information, etc. These would require that we maintain some kind of high-performance table-engine that maintained parallel data for the header attributes of the objects in the InTray, either in memory or on disk. Notification of changes would allow us to insure that the parallel data (a cache, really) was up to date.

Similarly, my personal document manager (AKA great message management that also works on other types of documents) would be responsible for providing rich search and browsing functions through my collection of filed messages and documents. We could do this by having a Personal Document Management class of directory. Getting good performance for these types of operations might require maintenance of a parallel index or indices. Ditto if we wanted to provide Notes-like views on the collection of docs in a directory.

You can think of the Group Document Library as another super-nice extension on a basic filesystem directory. It would have a lot in common with the Personal Document Manager class, but would add functions specific to group document management. CheckIn and CheckOut would require new APIs, but since the documents are just stored in a directory on the LAN, apps could open them directly through normal FS calls. Content Indexing would be accomplished through FS hooks and a separate content index. Views would be accomplished through a special viewer, that would have a lot in common with the personal document manager viewer.

>From this perspective, the bundled email package is pretty simple. You require people to keep their InTrays on the LAN. We add a little module to Spitfire that puts the right data into the right EAs in the right InTray directories at delivery time, for these "wimp-mail" users. The super-simple SFS transport does the same thing - just creates files in the right InTray directories on the server. Then you provide an InTray class viewer, maybe an OutTray, a few basic forms for message reading and composition, access to the LAN directory, maybe a personal directory.

GROUPSHOT

Microsoft MEMO

TO: Eric Candell
Bob Matthews
John Morey
Jeff Raikes

FROM: Bruce Burger

SUBJECT: **"Groupshot" – Laser Groupware Vision**

DATE: March 21, 1990

Following is a description of the "Laser as groupware" vision. This is one of the three potential Laser visions that we will discuss at our meeting later today.

This product is referred to here as "Groupshot."

GOAL

To produce the best product that helps people work together. We will help people send email, produce and share documents, schedule meetings, and tell other people where they are and when they'll be back.

(The latter feature, similar to a product that CE offers on the Mac, is referred to here as "personal status". It is the least important of the components in this vision, so we shouldn't get hung up now on the question of whether it's worth doing.)

FUNCTIONALITY

Groupshot will be an application separate from the shell. By default it will feel like a mail product. However, it will have several major additional features:

Library / Bulletin Board – There will be a facility along the lines of the DLS proposal.

User Interface – This repository will appear to the user as a set of shared folders. A user can access shared folders and private folders using a similar UI.

Application Objects – It will be easy to mix messages (forms with rich text) and application objects in a folder.

Annotation – It will be easy to annotate application objects as well as forms. All will use a common annotation facility. There will be facilities for both text and voice annotation.

Replication – As in DLS proposal.

Access Control – Similar to the DLS proposal, but these can apply to folders, application objects, forms, and form fields.

Distributed Access – As in DLS proposal. If an object is replicated, the copy on one server must be designated as the master. If a user on another server tries to check an object out, they will be connected synchronously to the server where the master resides.

Revision Control – As in DLS proposal.

Forms and Workflow Processing – There will be a forms package optimized for use by administrators and sophisticated users. An object can go through several states, each of which has a different form. There will be a high-level macro language for form actions such as "archive" and "send to Joe". There will be an easy way to see the history and state of an object.

Queries – There will be a powerful query and filtering facility. This same facility will be used for both private and shared folders.

Notification – A user can choose to display the status of selected folders (both private and shared). The UI might be to display these folders on the app desktop, or might be something different. The status of each folder – no new messages, some new messages but none urgent, or some urgent new messages – will be prominently displayed. The user can also request other forms of notification for selected folders. The user can also place filters on notification (e.g., flag or ignore items with certain attributes).

Calendar – Users can synchronously schedule appointments. If the appointment involves users on other servers, those servers will be synchronously accessed. There will be tight integration between mail and calendar (e.g., send agenda and link to appointment).

Personal Status – The user can easily indicate their status (e.g., in, busy, out) and a short message (e.g., where they are, when they will be back). This information can be easily accessed by other users to whom they have given permission.

A LITTLE MARKETING STRATEGY

Packaging

The mail product – client and Spitfire – will be sold as Slingshot. This enables us to have a low-cost, universal email platform. Slingshot will include the annotation facility. The other functionality will be packaged separately, for example in two optional packages: library / bulletin board, and calendar / personal status.

Third parties

We will encourage other app developers to use our database and transport facility via DDE, DLLs, or named pipes (which of these are supported is TBD). We will also make our annotation facility available to other app developers.

PRODUCT COMPONENTS

Built by other groups

The following components would need to be built, probably *not* by WGA:

Multi-user database – This will store mail messages, documents, calendar entries, and personal status.

MTA – This will be Spitfire as currently conceived.

Built by Workgroup Apps

The following components would need to be built, probably by WGA:

Overall UI and browser – The browser will provide a unified summary screen view of mail messages and communicated documents. To reduce dependencies, it will be separate from the shell where the user's personal documents are stored. However, there will be drag and drop integration between Groupshot and the shell, so we can market the concept of shell integration.

Forms – We will probably need to improve Slingshot forms to support most rich text attributes, in order to prevent Notes from looking much sexier than us. We will also need a forms designer.

Calendar – This will be a separate module, although moderately integrated.

Personal status – This will be a separate module, loosely integrated.

Annotation – This will be a separate module, built by us working closely with other apps. It will be replaced eventually by compound documents when we become truly object-oriented.

Macro language – Initially this might be absent or limited to a few specific workflow functions. Eventually it will be EB (not built by WGA of course).

PLATFORMS

Initially, we will develop only GUI versions of all components. We will support DOS clients for basic mail functions only via a client gateway to CSI's DOS client. Later, we will develop stripped-down versions of the calendar, library / bulletin board, and personal status if demand justifies it.

The system will support LAN Manager and Novell LANs. Version 1 will be limited to LAN Manager due to Spitfire limitations.

A LITTLE COMPETITIVE ANALYSIS

As a response to Notes

Groupshot does many of the things that Notes does.

Our biggest advantage over Notes is that we view email as the core of groupware, and will make sure that our email product is competitive with the best. (Unfortunately, Lotus is likely to improve their email component eventually.) In addition, we will have the calendar and personal status modules, and we will have a modest level of shell integration which we will play for all it's worth. We have a more diverse set of apps that can use our annotation facility. We will support application objects better than Notes. And we will be in a better position to evangelize our APIs as an extension of the shell.

We will probably fall short of Notes in some library / bulletin board features, such as the ability to develop customized apps and security.

As a response to OfficeVision, New Wave, WP Office, etc.

We need to convince people that the Windows/PM shell is adequate for personal use, and that a separate shell is acceptable (perhaps even desirable) for interpersonal activity. Meanwhile, we will offer far more and better functionality (mail, calendar, etc.) than any of these. And we are well-positioned to evangelize our APIs.

Our risk here is that people will find these shells compelling. Also, WP Office might add a lot of mail and calendar features, and take advantage of their WP base.

As a response to email products

We will have the library / bulletin board, calendar, and personal status modules. We will have a modest level of shell integration. And we are well-positioned to evangelize our APIs.

Our risk here is that the existing email vendors will add a lot of mail features, we won't keep up, and customers will consider these features important.

ANALYSIS

Strengths

- It fits neatly into the MS Office vision. Our other apps are personal productivity tools; Groupshot pulls it all together.

- It takes advantage of our position as a systems vendor in two ways: we provide modest shell integration (not nearly as much as we will eventually, but enough to excite people) and we evangelize our APIs as a systems service.
- The only major *integral, critical-path* technology beyond Slingshot as currently conceived is the multi-user database. The other components – calendar, personal status, and annotation – can be developed by relatively independent teams, and even shipped in a later release if necessary.

Risks

- Notes, with their considerable head start, could do a better job of library / bulletin board.
- The existing mail vendors and WP, with their considerable head starts, could continue to sell better email products.
- We won't have a Novell solution for a while, and that excludes the vast majority (at least today) of the market. Most of our competitors will work on all shared filesystem LANs.

Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717

Tel 206 882 8080
Telex 160520
Fax 206 883 8101

DAYTIMER

Microsoft Memo

To: Richard Bowen
Bruce Burger
Bob Matthews
John Morey
Jeff Raikes

From: Eric Candell

Subject: Laser—A computerized daytimer

Date: Wednesday, March 21, 1990

Goal

The goal of "Daytimer" is to provide users with a tool that frees them from the pressures of remembering to do things. It will manage their calendars. It will follow up on their requests and activities. It will send them reminders. It will schedule their meetings. It will maintain their task lists. It will track the progress of their projects.

Functionality

Object Management

Initially, we'd be building our own simple object manager. Our goal would be to switch to OOSH when it becomes available. The functionality of the object manager would include:

Event Driven Features

User will have control over his objects based on timed events (relative or absolute). Examples include archive at specified time, remind me of object after certain time, send object to another user at certain time, delete object after certain time, tell me if I didn't get a reply to an object I sent , move object to destination after certain time.

UI will be tied to the calendar. Potentially, you drag between various markers on the message connecting to dates in your calendar.

File System Integration

User can pull documents from the filesystem into the object structure and use the event/calendar aspects to control them. Potentially, this would be done through linking/embedding FS files into object folders.

Forms

Users can also create objects that are filled-in forms. Potentially, these are "index cards", "rolodex entries", "to do" list items or other personal forms (like templates built in Word). Or, they might be more message-like with addressing fields.

Email

All objects in the object system can be sent to other users. Likewise, objects can get into your object system by being received from other users. These capabilities would exactly match those described for Slingshot today.

Graphical calendaring

GUI interface

Each day will be an object in the object manager. We will provide some specific GUI interfaces for accessing these "day" objects. You will be able to move other objects from the system into your calendar and associate messages to events in your calendar (along the event driven lines described above).

Scheduling

A scheduling form that users can send to others will allow for schedule comparisons and meeting/event arrangements. Any object in the LMDS directory space can have a time index stored for it (including resources, conference rooms, files, printers, etc.)

Project Tracking

Some integration with Whimper

There will be some easy way to get the information from Whimper into the time management aspects of the model. How this might work (and whether it would be there at all) is TBD.

Marketing

Replaces Paper daytimer

A point to note is that daytimers cost between $75 and $100 initially with incremental costs of $25/year. The present value of the cash flow (assuming perpetuity for simplicity) would be as much as $350.

Runs on Laptop

With the advent of notebook computers and the acceptance as laptop computers, we would be delivering an application that justifies the vision of "a computer on every lab". For any application to succeed as a laptop accessory, we will need to make the object store (optionally) client based. Perhaps in the first cut, there will be a way to take a snapshot from a central server and push it to the central server at a later date. But, the capability to use the calendar while not connected to the network is essential.

Product Components

WGA staff produced

UI for object manager (generalized Slingshot)

UI for calendar
Object manager (generalized message store)
Calendar back end
Remote access tools

Third Party/Other groups

Spitfire
LMDS

Platforms

First, we would develop GUI versions of all UI pieces. A Mac calendar piece, as well as a throw-away CW calendar piece probably would be important additions in a fairly short time frame.

Competition

We will compete on the following planes:

Calendaring that is richer than WP.
Electronic mail facilities that are better than Slapshot and more comprehensible than Notes.
Integration of calendar and email, user files, and time management more creatively than all other products on the market.

Analysis

Strengths

We would be developing a product that would be almost inarguably a boost to productivity. Getting people to manage their time more efficiently is improving productivity.

We would be providing a fairly rich email solution built on a high end platform in addition to other features. So, we'd have appeal to system administrators.

We would be tying in with other Window's products and giving an overall user model that, over time, could be absorbed by the shell.

The calendar aspects are potentially separable from Slingshot as currently conceived. Also, the object orientation and forms functionality could be coerced into a generalization of Slingshot.

The event driven nature of the object management model would differentiate the email aspects of the product from other email systems. The concept of a daytimer is something that users can grasp without exceptional amounts of learning or imagination.

Calendar technology is not something that is likely to be replaced as other groups get further ahead with their products.

Risks

The OOSH is likely to reinvent the management premise of the product in a more encompassing fashion.

Our calendaring may not stack up to WP office, while our mail might be inferior to other products on the market.

The product won't be viewed as being innovative enough (it won't get the press that Notes has gotten) because there are essentially no new ideas in it (with the exception of the event driven objects for mail management).

File : c:\wzmail\laser.fld
Messages : .

(NAMELESS)

185
From richardb Fri Mar 16 13:53:02 1990
To: barrym chrisz nickh
Cc: bobm
Subject: Visions of Laser
Date: Fri Mar 16 13:52:57 1990

Here is a vision statement to get us going. It is significantly different from anything we have discussed so far - there's no point in rehashing old ideas!

Laser will provide tools to manage time valued information in a work group environment. We will build email and some other basic applications with it as an initial product, but the goal is to sell it through ISVs and users that customize it for particular applications. Something very similar to our current Slingshot could be built with it, but we could do a lot more. The fundamental features of this product are:

1> tools to address and distribute objects to users on the LAN

2> tools to define events involving these objects and notifications based on these events. Events are predefined, or defined by type:state transitions

3> facility to install object management drivers for new objects

We would develop an email/bulleting board product with some other features TBD as a demonstration of Laser's capabilities and to provide the value that would initially generate sales. Our ultimate goal, however, is to make money from the many other products that ISVs would build using the same underlying capabilities - just as spreadsheets have obvious capabilities of their own, but provide the underlying technology that ISVs and and users employ to make other products.

Laser must include facilities for user name services, user object delivery services based on these names, time and state valued events associated with these objects, facilities to notify processes about these events locally and across the net (the UI notifications would be based on this lower level notification), and storage of information about these objects, their current state, event history, and location. Laser would manage this meta information about objects, and not the objects themselves. This meta information would be used for high performance lists of items in folders that we have discussed.

Forms would not be fundamental to this technology, although we would use them to implement our first application. I presume that if we use our own forms we will have a slicker interface that will help sell the initial Laser release. However, once Thunder or some other product supports the performance and features we require, we can switch. We could implement the low end extensible email product based on Thunder separately.

Management of particulat types of objects will be implemented via separate data access drivers that can be installed much as we can now install drivers for different printers. This enables us to support heterogenous data using a single model; i.e. Laser will be able to handle file system objects, PIM objects such as messages in a message store, OMEGA data base entries, etc. by installing the appropriate drivers. These would be abstracted to the same browsing interfaces in Laser.

The driver model is similar to the 'installable object' model of OOFS, but has the advantage that it is independent of OOFS but consistent with it.

For initial release, we will implement a message store data access driver. We will encourage ISVs to explore ways of leveraging this capability with their own data access drivers. Features such as double clicking on objects and opening their app can be supported using this model as long as we define the driver interface properly.

Information about objects will be recorded locally/privately and/or remotely/publicly. The same event/notification applies in each case and the application will determine which one is used. In any case, the app identifies the type of an object and its state when it is sent or changed. There is a data base of event definitions; some of these may be predefined (like submit) and others may be defined in terms of transitions from state to state to customize the system.

Laser retains a database of message types, states, and events which can be modified by apps and, if the UI is provided, by users as well. This facility would support, document control systems (this is different from DLS - document conrol refers to paper processing as in purchase order processing or mechanical engineering procedures). It would support other applications such as controlled backup, archival, and disposal of objects, and other applications that we cannot anticipate.

The application database would contain at least the follow information. It might exist locally, or be shared globally, or both. At some level, users could actually program applications just by creating definitions and rules in this database.

Legal States = object tags definable by the app. Laser has a minimal number of predefined states.

Legal Types of Objects = Definable by the app. They may be our form types, file types, or Omega record types.

Legal types of Notification = action to take when an event occurs. This might define define a callback, DDE messages, and/or LAN broadcast messag
es

as actions to take for notification, specific tasks to initiate, batch programs to run or users to notify.

Types of Events = Laser would predefine some events and an app could define others. Events could be defined as state transitions of objects rather than requiring code. For each event, the user could define notification rules. E.g.:

X 581063
CONFIDENTIAL

ObjectType	Old State	New State	Event	Notification
Order Entry	Null	New Order	NewOrder	OrderProcessin
Order Entry	New Order	WaitingForShip	SoShipIt	Shipping
Order Entry	WaitingForShip	Shipped	ReadyToBill	Billing

File : c:\wzmail\laser.fld
Messages : .

188
From richardb Mon Mar 19 15:28:37 1990
To: bobm
Cc: danab
Subject: Re: Laser -- the vision thing
Date: Mon Mar 19 15:28:31 1990

Here is a brief product description by Dana that gives an impression of what my proposal can do:

> The goal of Laser is to improve office productivity by automating white-collar work. Small companies will buy it because it makes their white-collar workers more effective; large companies, because it will make some of them expendable.
>
> Laser models work as a flow of documents among processing nodes, which may be people or computer programs. It automates the mechanics of that flow -- addressing and sequencing -- as much as possible. It creates opportunities for automating processing nodes by structuring the documents themselves, making the information digestible by programs. (Right now, companies are realizing big gains just by image processing of paper-like forms. Laser can provide that gain, plus opportunity for automating processing steps incrementally without the overhead of IP.)
>
> Out of the flow of documents grow a shared base of knowledge and a history of workgroups and projects, most often captured in paper files and (especially) people's heads. Through document management, Laser can make this knowledge accessible for new employees, reporting, tracking, and post-mortem.

This "shared base of knowledge is the database which I described; it contains message types and forms, states, event definitions, and notification rules corresponding to a particular business process. We found that users needed a lot of help to understand how to create this "enterprise knowledge base" because they do not know how to analyze their business operations. We were forced to write an additional set of manuals describing how to do this and map it into the database definitions.

A separate group developed a consulting business based on performing this analysis and creating customized applications.

The full system was called a "Data Management and Control System"; presented properly, this concept rang bells with higher executives in many large companies. General Electric started the ball rolling in the late '70s by putting out a request for proposal for such a system but later dropped its request. The technology was not available to sustain it at the ambitious scale GE contemplated. IBM bid on it and continued to pursue it independently afte GE quit. I believe that they are still working on it in Atlanta.

GE does not own the application category so there is no legal problem jumping into this application area, and many companies would be happy if they could find a demonstrable DMCS product that they didn't need to pay someone to develop. To them, high prices seem cheap in comparison to the potential savings.

There is a near term market for a more modestly scaled product: There are many mid sized companies which could save a lot of money by automating relatively stable paperwork procedures with Laser. As LANs grow and our experience with Laser gives us more experience, we can tackle larger problems such as the one addressed in GEs DMCS RFP.

> Laser must be useful in a generic way right out of the box, but also provide opportunities for addons specific to a company or industry: there is no way we can automate a company's critical functions in a generic way. An open architecture is crucial, as is a general and flexible model of white-collar work (class hierarchy of transactions).