A Paradigm Shift to IAYF

Jim Allchin
July 22, 1993

1. Introduction

I don't consider any of the thoughts in this memo to be deep. I don't consider them to be necessarily original. There may be some new things included, but from my perspective, most everything in this memo seems pretty obvious. Surprisingly enough, however, a paradigm vision is lacking at Microsoft and a global company implementation plan is missing. What's most alarming is that there are key managers that do not believe in paradigm shifts. Perhaps, this memo will help in focusing more attention on this vision.

2. What is a Paradigm Shift?

A paradigm shift is a change in the way we think about things. It must be a fundamental change -- so fundamental that continuing to consider the world using the previous model is at best sub-optimal and at worse will produce incorrect answers to questions.

Throughout the history of computers there have been quite a few paradigm shifts. Some of these include high level languages (e.g., Fortran), timesharing, PCs, 123, WYSIWYG (e.g., bitmaps, mice, laser printers), networks, consumer devices, etc. It isn't hard to list them.

What are the implications of these kinds of paradigm shifts? There are implications on customers, the market, and suppliers. From a customer base perspective, a paradigm shift

- often requires new investment (e.g., a mouse, memory, consumer device, etc.),
- broadens the user base in some meaningful way (e.g., scientists with Fortran),
- changes the work process (flow, jobs, efficiency) in some important way (e.g., spreadsheets),
- usually changes the answer to the question of "What is a computer for?"

A significant number of people must recognize the model change or it's not a paradigm shift by definition. A very important benefit of a paradigm shift is that they are, more often than not, hard to accomplish. That is, because they involve a new model, old techniques don't lend themselves easily to the new model. In short, he who gets there first wins in many cases. This is true outside the computer field, but we'll stay within our own business sector here. 123 caused a paradigm shift. Microsoft was unable to surpass this until Excel leveraged the "pretty face" aspects of Windows. As good a product as 123W is now, it will be hard for them to surpass Excel. There is usually a high cost of entry to the paradigm shift. The more lead time you have, the more you win. Of course,

this assumes that the paradigm shift does the items listed above for people – that is, the demand must be present. Once you miss the wave and a leader is established it is quite difficult to overcome this. The best bet is to look for the next shift and ride that wave before your competitors. The biggest mistake companies have made is to ride the previous wave too long. Consider consumer devices vs. PCs vs. minis vs. mainframes. Consider character vs. GUI interfaces. Consider direct vs. indirect selling. At the same time it is important to bypass a wave that won't have sufficient user benefit. Doing this leaves you investing in a technological area that will quickly be overcome by a more important wave. Just like surfing, knowing which wave to catch and when is key to being wildly successful.

A final point is that one way to deem whether a change is a paradigm shift or not relates to the difficulty in changing to the new model. Things that are easy to do (or clone/copy) usually do not represent a paradigm shift. Companies that do paradigm shift first win very big because of this fact.

3. So, What is the Next Paradigm Shift?

Lots of people have proposed what they think this next paradigm shift is. Ideas that I have heard people talk about include

- UI changes (e.g., Tabs everywhere)
- OLE 2 applications
- Tighter application integration (e.g., drag/drop everywhere)
- shell extensibility
- SDI
- ability to do add-ons an customization easily
- tools to write applications fast
- OS services viewed as objects
- query
- new storage systems
- etc.

It is pretty easy to argue that some of these (e.g., tabs everywhere, a prettier shell UI face) do *not* change the model sufficiently to address some of the characteristics I mentioned above. (How long would it take a competitor to copy each one of these? Not very long.) I've heard people say the shift is really just lumping some of these together. Even thinking about lumping them shows that the essence of the change isn't understood.

My perspective is that there is one paradigm shift composed of three fundamental parts. These are Information Access, Programmability, and Composition + Components. Each of these parts offer enough power to be a compelling paradigm shift alone. However tremendous power comes from merging them. I believe we are already moving ahead in each of these areas, but certainly I think that the key essence of each part needs to be continually restated.

4. Information Access

The primitive form[1] of this paradigm shift is to change the searching and retrieval strategy at the desktop and then remote this capability. Access to items in file systems and directory systems today is slow and primitive. No commercial system offers native support for attributes on files and indexing support for both attributes and the content of the files. We believe this will dramatically change the way people deal with finding information.

---

[1] A more generic form of the Information Access paradigm shift is covered later since it depends on understanding encapsulated content and structural organizers.

Our implementation of this is OFS and catalogs. Locally, a user will experience significantly more flexibility in access and higher performance than traditional file systems. It is my belief that users will want to use this capability completely as soon as they experience it. Unlike file sharing protocols that require the data to be dragged over the network, we do the (query) processing at the site where the data is stored. The result is blindingly fast information access in a network. We changed the SMB protocols in order to achieve this. We believe customers will be highly interested in Cairo "Information Servers" in their networks. We have created a paradigm shift.

This paradigm shift offers a new model for users and that's great. The real beauty is that Novell cannot duplicate this capability easily. It would require a significant effort to first duplicate functionality such as OFS and second clone the capability to book out the object interfaces that are used on the client to connect to their updated file system. Further, if they try to mimic the SMB protocol, then because of patents we can control that as necessary. We will have a lead in providing the server functionality in this paradigm shift.

As part of the work we have also added DFS and support at the win32 level for link tracking. However, these are not the keys to this shift. They are just features that help making the infrastructure better able to support both of the shifts discussed in this paper.

This searching capability and distributed system infrastructure support we are doing will also hurt Notes, but not as much as Novell. In order to overcome Notes, it requires the other paradigm shifts discussed below also in addition to this shift.

Although I consider this shift fairly simple to understand, the importance should not be underestimated. *I know of no way to beat Novell by a features war.* Certainly if we had competitive features and then priced the system low, we could hurt them. But, adding a directory service, security features, etc. would not be sufficient for a customer to switch to Novell. Even arguing lower administration costs or something like this simply won't work. We have to do something that the customers say "I have to have this feature." It must be so compelling that they are willing to accept pain in having a mixed network for a while and willing to invest in the new approach. I feel comfortable we have defined this.

I simplified the presentation on OFS especially when you consider the capability of Inference technology. We can dynamically cluster and categorize information in ways that are very novel in personal computer systems and servers today. The addition of these features will accelerate the transition to this wave by making the environment more compelling.

5. Programmability

Everyone will be a programmer soon. People want to tailor their solutions to their problem from building blocks. They do not want to have to write tons of code. That's why they want things like an Excel worksheet VB control. Every application object should be programmable. Another way to think about it is that anything that can be done through the UI should be able to be done programmatically. The method for doing this programming should be consistent between all components (between supplied by the system or an application). Moreover, standards must be defined so that even if we don't switched to shared implementations everywhere that exactly the same invocation methods work on similar components in different applications (e.g., canonical IDispatch names and ids).

Finally, even something like VBA is much too complicated for many people. There are many cases where code should *not* be required. There are two approaches to this: wizards which write the necessary code or more powerful negotiation capability where the objects themselves are able to abstract what should happen when one object is connected to another. This is an area where Microsoft should focus and become a leader. This is a huge revenue opportunity I think for the Tools group. If everyone is a programmer, the market is a lot bigger.

6. Composition + Components

The paradigm shift here is to restructure our applications and systems so that content is encapsulated (as objects) so that it can be used by other objects – especially objects focused on structuring these other objects. There are many arguments for making software "object-oriented". You could look at it from the technology perspective and talk about code reuse allowing greater synergy (e.g., a single toolbar implementation and therefore guaranteed look and behavior commonality) or the fact that you should be able to produce more software because of the lack of redundancy. These are all valid, but they only impact the customer indirectly. The question is what is the key shift that is possible by moving our system and applications to be components that are composable/structured?

People now see compound documents (e.g., in-place editing) as an important productivity gain. However, beyond this, Microsoft as a whole hasn't seen the potential of this path. In fact, I believe people are actively fighting adoption of this perspective.

One way to visualize systems is in layers. At the bottom are atomic objects. These atomic objects are then tied to one another to make up somewhat larger building blocks. These are in turn are composed to create yet larger objects and so on. Code is used to glue the objects together at each level. Consider a programming language. You define several *structs* and then define an object that links them together to form a larger structure. If the encapsulation is done correctly, then you could deal either with the structure itself or the individual objects maintained in the structure. Further, the structure can be insulated from the types of objects that are linked together.

This wasn't some wild academic discussion. Everything above applies directly to the paradigm shift. We want objects like the *structs* above to be able to be composed in the same way by larger container structures like tables, lists, documents, dependency graphs, UI forms, etc.

Clearly, objects can be containers as well as be contained, however it is useful to discuss the main purpose of an object: to be composed or to do the composing. This is quite visible in the OLE compound document architecture. There is a set of interfaces that must be supported if you want to be embeddable and another set if you want to be a container for embeddings.

6.1 Content Objects

OLE has focused on making embeddable content objects. There are two issues facing us dealing with content. All the interfaces that are needed to manipulate these objects in a wide variety of containers are not yet defined. The recent work on OLE objects as controls is an example of this. (Even simpler is the fact that we need overlapping, transparent, non-rectangular frames, etc. support in the interfaces we have today.) The good news is that given we decide on the type of container structures we want an object to participate in, we can architect the appropriate interfaces so that object can be fully utilized in the structure. Secondly, we haven't leveraged our current applications through composable pieces. Cairo has defined an RTO which is quite powerful. Why isn't this some form of Word? What will Word's role be after we are done? I believe we will end up with a small hierarchy of text controls (2 maybe three -- from the RTO up to full Desktop Publishing support). Cairo hasn't defined a dependency graph container structure object yet, but why wouldn't that be a slim-fast Excel?

An object is something with a moniker to it. This means that the granule size of an object can vary dramatically -- it can be anything the server code wants it to be. That is, it can reach way down inside itself and promote a cell, a character, a line or anything else into being a first class object. This is a beautiful tradeoff allowing any size entity to be manipulated, composed, etc. efficiently without having to follow some formal object-oriented specific programming design methodology (other than OLE).

The implications on storage of content objects vary depending on object type. Video clips demand certain features and a UI button control requires another set. It is important that storage is abstracted so that these different types

of content objects can be supported efficiently. The IStorage/IStream/etc interface set in OLE and Cairo fits well with the OLE compound document, forms, and file system models.

6.2 Structure interfaces and Container Objects

Ever wondered why as spreadsheets, word processors, presentation packages, databases, project schedulers, etc. get more feature rich that they start overlapping more and more with the other application features? Each of the applications continue to specialize on one *thing* (more on this below), but we are starting to see many of the same capabilities in different types of applications. As content continues to be more encapsulated – what will these applications be specializing in? Stevem has written a great paper which begins to discuss separating content from structure and on the application model. This is an important key that really hasn't been understood widely before.

A spreadsheet should be able to have full word processing capabilities within a cell and a table within a word document should be able to have full worksheet capability within a table. A field in a database should have full word processing capabilities and database values should be able to be used in a spreadsheet. And so on. So, what's the difference between all of these?

These containers differ in a number of dimensions: highly efficient support for some views, assumed transaction model, etc. We need to decompose all the possible structures into ones that are orthogonal to each other and provide the most power to a user. For example, document, table, dependency graph, collection/set (aka folder), etc. Just like in everyday life there are different organizers for different tasks, this is true here too. Our applications today are beginning to share content objects – they do not share structuring at all. Most of the *interfaces* (as in OLE) have not been defined for them to do this yet.

If we can create canonical structuring interfaces and containers supporting these, there are huge benefits for users in a number of ways. If you believe that people deal with heterogeneous pieces of information a lot, then having the structure separate will be a significant advantage in letting someone both ask questions and organize information efficiently. Certain structures are more apt to be used for heterogeneous use than others (e.g., folders, forms), but other containers offer new advantages for manipulating heterogeneous data in new ways. I can imagine being able to set up dependency information among lots of different types of objects by using properties on the objects, for example.

Users want to see data presented in different aspects using different structures and so it isn't surprising that the we want some powerful containers supporting many of the defined structuring *interfaces*. To support these interfaces efficiently the storage system must provide intrinsics that let the information be viewed and reorganized quickly.

In Cairo so far, we have concentrated on providing only a few key containers. I will talk about two of them[2] here: *smart folders* and *smart documents*. Folders are a quite generic form of a set which has much richer capability than what people normally think of as a folder – especially if people relate them to directories of today's file system. Folders are set structuring containers of heterogeneous objects, have intrinsic views on their contents (e.g., chart views over some property or fast report views by pre-indexing certain properties), rich iterations functions (e.g., history like in Help), native usage support (e.g., read/unread tracking), etc. We are attempting to push this one structure to the extreme.[3] We have also focused on supporting easy customization of this structuring object – thus the name *smart* since it is possible to change the behavior easily through add-in code (*prepare to do*, *do*, and *after do* events).

The other key container class that we have focused on deals with simplistic documents that have rich "form-like" structure. We call these smart documents (or *infodocs*). We believe containers of this type address a wide variety

---

[2] Another important container in Cairo is a FORM.

[3] This framework helps us clarify when a structuring container is so fundamentally different to a user that it should be treated separately as a new concept (e.g., consider projects, workbooks, etc. compared to folders).

of needs used to run businesses today. These are directly competitive with a Notes' note. We are using them to build our Help and Mail documents. Clearly, these documents are nothing more than forms with some controls that specialize the documents for their intended purpose (e.g., history control). Just like the set containers above, these documents are smart because it is possible to add behavior to them. Note that although I discussed these documents as a container, they are also a content object as well (e.g., they can be stored in a folder or manipulated by other structuring objects).

The key to Office is the understanding of the key containers and key objects that need to be supported. Cairo has made a start by thinking about this area, but I think it's only the beginning. Throughout the company there are people that are lightly touching on this issue, but they do not have a common vision of the desired outcome. Databases structure data. Spreadsheets structure data. Project management systems structure data. PIM systems structure data. I have a great fear of overlap in these structures which will result in a confusing array of structures for users to choose from. We need to decompose these structures down and find the minimal set.

Certain structures are only efficiently implementable using a particular storage arrangement. Relational database systems separate data from structure by normalizing the data completely and then binding the objects together through a calculus. In order to achieve performance, almost all relational systems use b-trees with clustering behind each table. Moreover, in many cases they dynamically do things like store prejoins under the covers for performance. Storage matters to structure.

What this means is that for each structuring container, a particular form of storage support may be needed. We have found that to be true for forms, compound documents, and smart folders. As an example, we added read/unread tracking to OFS for folders so that we could achieve higher performance rather than maintaining the necessary tables in application space. A rich storage system like OFS is one piece of the puzzle. The moral: understand the structures so that you can optimize the storage.

6.3 Why?

Why is this such an important part of the paradigm shift? Why would a customer care?

First, people want easier to use and consistent products. That means having common content objects (everything from buttons to complex objects). Second, people spend a great deal of time organizing data – whether through outlines, tables, documents, sets, etc. The reason why Notes is so compelling is that it provides a rich organizational capability. Many people attribute this to the Notes database and that is partially true just like when people refer to OFS as the key to Cairo. In reality, it's just that the flexible organizing capability is matched appropriately to the storage. This will offer great power to even native users.

When you integrate all three parts above, you create a rich personal information management system.

5. Conclusion

Microsoft is away trying to catch the Consumer paradigm shift wave. I believe this shift is easier to understand because it involves new hardware that is easy to visualize and it involves the integration of several understood existing technologies: communications, general purpose computers, information/media suppliers. Clearly, there are many hard problems and we have a long ways to go, but we have good vision and are focused on making this shift happen.

On the other hand, I think we are having only very limited impact on the composition + components paradigm shift discussed above. This is clearly the key one as far as our applications are concerned. I am convinced after talking to Ray Ozzie and others from Lotus that they *do* understand this shift. The same is true about Borland. Clearly, understanding and implementation are two different things. I feel that between OLE and Cairo education

we have given competitors[4] both the tools and the thinking to beat us at our own game. Either we catch this wave or we will eventually be overcome by the new wave of our competitors.

This won't happen overnight, but it could happen. In order to command a significant price for our applications, we must have significantly more value than our competitors. Today, price wars are already underway. That means that parity of some sort has almost been reached. Unless we do something dramatic (a paradigm shift), it will be their features compared to our features – and the price *still* will go down. Whomever introduces a significantly compelling product first (that isn't clone-able easily) will win the next wave and be able to command the high price. Notes and Netware should be warning enough that not catching the wave first costs dearly.

As I pointed out in the beginning some people think that doing sexy things in the UI and a little more basic integration will give us a significant advantage. I do not. The work in Cairo is dead-on. But at some level, we don't count – it's the applications that count and they must catch this wave.

[4] I have seen an internal document sent from Ray to Lotus after attending our Cairo design preview. It is clear from that document and from further talks that I have had with him, he understands the paradigm shifts I discussed above. Further, both Iris and Lotus are using ideas received from us in making Notes and Lotus applications better weapons against us. We have also educated Drew Major on some of the issues dealing with object storage. We have given them the general roadmap.