Smarter Programming Brings Smarter Computers

A big limitation of almost all computers is that they are passive: They only do what we tell them to do. But what if they could actually learn from our actions? What if they could take the drudgery out of locating files? More of us could use computers if only our computers were smarter.

Scott McGregor, general manager of Atherton Technology's Ahead Division, thinks smarter computers are the next step beyond the ease of use we get from graphical user interfaces, or GUIs

In his article, "Prescient Agents: A Radar O'Reilley for your Desktop," (The X Resource, Fall 1991; O'Reilly ad Assoc.) McGregor draws the analogy between the Radar O'Reilley character on the television series M*A*S*H--who was always able to anticipate the colonel's needs for files and information--and computers that can actively give us information with minimal effort on our part.

Many people are just beginning to understand how GUIs make computers easier to use by making them easier to command. But prescient user interfaces, or PUIs, promise to teach our computers to learn from what we have done and serve as an active helper. While McGregor doesn't expect commercial products for two or three years, he is convinced of their usefulness to attract new computer users.

Here's an example of how a PUI would help me work. I manage my DOS files using Microsoft Window's File Manager. Each time I use File Manager I must tell it how I want the files arranged and what I want to know about each one. These are options that I can select, but I must select them each time I use File Manager. If File Manager were smarter, it could anticipate my needs based on my habits of using it.

Of course, the best way to handle my files would be to have my computer handle them for me--by understanding how I work, what I need when I work on certain jobs, and so on. In this way, my computer acts like an assistant that hands me the right plans, tools and materials as I need them.

PUIs could also help us work smarter in group situations. As I'm writing this, I 'm also handling off part of a project to another person who would be more productive if she could work at my desk--complete with all my tools, notes, files, and lists of contacts. In a PUI world where all these objects are linked, she would be able to recreate my desk as her own.

McGregor goes into far more detail about how prescient user interfaces and session managers work than I can fathom. For me, more interesting than how they work is how the thinking of programmers is changing in response to the needs of non-programmers and the new technology of object methods.

It's critical that both my computing environment and my applications be configurable in ways that make them easier for me to use. But I don't want to become a programmer to get this benefit. This leaves me with two options: hire a programmer to help me run my life, or buy software that understands what I want and anticipates my needs.

We all want computers to make us more productive. But before that can happen, there is much to be done to make computers easier to use and less problematic. If programmers are beginning to recognize the needs of naive users, we are likely to see improvements in how computer work for all of us.

PRESCIENT AGENTS: A RADAR O'REILLEY FOR YOUR DESKTOP

HOW APPLICATIONS CAN SUPPORT AND ANTICIPATE USER NEEDS BY MONITORING USER ACTIONS

Scott L. McGregor

"Reprinted from *Prescient Agents, A Radar O'Reilley for your Desktop*" in *The X Resource: Issue 0,* October 1991, pages 95-111 (*X Window System Series*), edited by Adrian Nye, ISBN 978-0-937175-79-8

ABSTRACT

Past systems have been largely passive, acted upon by the user. But new systems are becoming more participative. Instead of mere slaves that do only precisely what they are told, we are entering an era in which systems will participate more like clerks or secretaries in a "team of two" with the user. This leads us toward interfaces that monitor actions and anticipate needs, automatically reconfiguring themselves to facilitate future actions. Prescient agents, as they are termed in this document, can improve the productivity of both individuals and groups. They represent the confluence of many research areas such as Ul design, agents, database technology and Computer Supported Co-operative Work (CSCW) into the applications of tomorrow.

Scott L. McGregor has been managing software development for more than 20 years, including ten years at Hewlett-Packard, where, as manager of an R&D team managing software development of commercial CASE systems, McGregor developed the initial concept proposal and prototypes for Prescient Agents described here.

McGregor is now founder of *Cell Pay Me*, where he continues his interest and work in Prescient Agents. McGregor welcomes contact from interested readers. He can be contacted at Swift Design Group, 412 Kings Court, Campbell, CA 95008 or by electronic mail at mcgregor@swiftdesigngroup.com.

Reprinted "Prescient Agents, A Radar O'Reilley for your Desktop" in The X Resource: Issue 0, October 1991, pages 95-111 (X Window System Series), edited by Adrian Nye, ISBN 978-0-937175-79-8

Reprinted from *Prescient Agents, A Radar O'Reilley for your Desktop*" in *The X Resource: Issue 0,* October 1991, pages 95-111

PRESCIENT AGENTS: A RADAR O'REILLEY FOR YOUR DESKTOP

HOW APPLICATIONS CAN SUPPORT AND ANTICIPATE USER NEEDS BY MONITORING USER ACTIONS

Scott L. McGregor

WHAT ARE PRESCIENT AGENTS?

Most computer systems today begin execution at the same state each time. When you start up a PC or Unix box, you typically begin with a directory prompt for your home directory. It doesn't matter if the last time you were using the system you were working in a different directory. A few systems, such as the Apple Macintosh Finder, remember the state from the last time you used the system. This saves you the effort of having to navigate to re-establish your previous working context. Only a few systems, such as the MIT Media Lab's NewsPeek system, actually learn about your preferences from many previous usage sessions. Prescient computing systems derive from this last approach.

Prescient computing systems are systems that learn about your work patterns by monitoring your use of the system. They simplify future work by creating and managing agents or other forms of accelerators that simplify recognition of and access to multiple work contexts. We call these systems prescient because they appear to users to have anticipated the need to change work focus by the way they simplify those changes. While this is possible due to the heavy habitual nature of much human work, we observed that it still could be pleasantly surprising to users. The systems only provide accelerators for probable work focus changes, based upon past usage patterns.

Heightening this feeling of prescience is that these agents are not merely personal but also actively support work groups of collaborators. Part of the surprise aspect comes from the fact that the system agents can not only learn your work artifacts, contexts and habits, but that the system agents can not only learn your work artifacts, contexts and habits, but also that the system agents can not only learn your work artifacts, contexts and habits, but also those of your collaborators. The system simplifies access to contexts and artifacts (created by collaborators) that are germane to current work, including those previously unknown to you.

Current systems are often confusing for users because they require the user to remember and specify details that are peripheral to their tasks. My project team at Hewlett-Packard and 1 have collected a large sample of Unix shell history files and found that 25-66% of each users' commands are just to navigate around and search for related artifacts in the file system. (See Figure 1). A primary cause of this problem is that most of today's computer systems do not manage the context in which you are working. This forces you to remember how the file system instantiates

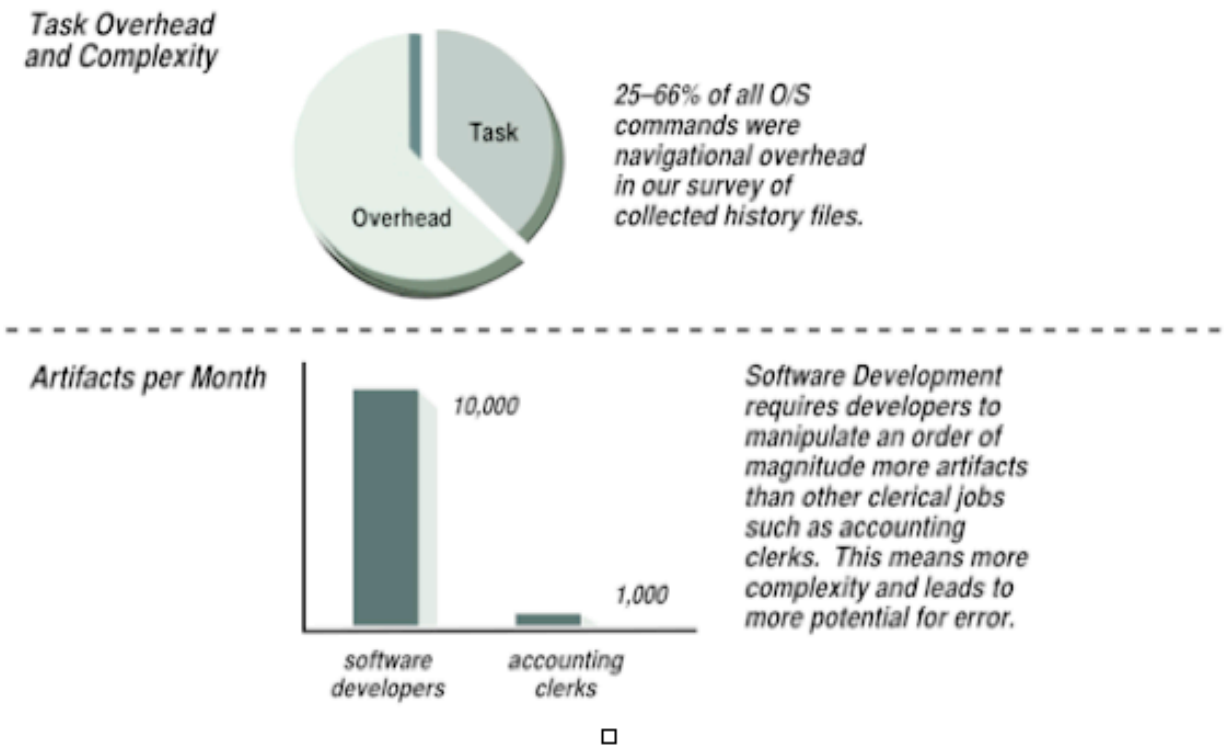


FIGURE 1: TASK OVERHEAD AND COMPLEXITY

your work contexts, (for example, by local naming conventions). You must then continually manage any translations or navigation implied by this mapping. These overhead tasks compete for short-term memory with your goal directed work. As the number of things to be remembered increases, cognitive performance suffers, leading to increased errors, and decreased task performance and productivity. We believe this was particularly true for the task areas such as we were studying (software development) that are characterized by an extremely high number of artifacts to be managed.

THEORY OF PRESCIENCE

Having recognized some cognitive aspects of current systems that were not well matched to the user's capabilities and tasks, my team and 1 sought to develop a model for how computer systems might further simplify personal work and collaboration with work partners. We found that having a model of how a human agent accomplishes such work was helpful for discussing how our computer agents could facilitate such work as well. One example of the type of agent we envisioned was that of an administrative assistant, and in particular we use as such an example the character of Radar O'Reilley, the company clerk in the movie and TV show M*A*S*H.

Reprinted from *Prescient Agents, A Radar O'Reilley for your Desktop*" in *The X Resource: Issue 0,* October 1991, pages 95-111

RADAR O'REILLEY - A HUMAN PRESCIENT AGENT

Radar O'Reilley is an example of how changes to the user interface can support management of complexity and interaction with others. We frequently saw Radar O'Reilley standing outside the door to his colonel's office with a handful of file folders. His colonel would open the door only to be startled by Radar there ready with a handful of files. The colonel would barely get out one or two words out when Radar was already handing him the one file he needed and offering other files he might also want to consult. Other times we'd see Radar finding out what was going on elsewhere in Korea by talking on the phone with Sparky, the local radio telephone operator. Later we might see Radar telling everyone to get ready for incoming wounded just before anyone else detected the sounds of approaching helicopters. In one episode, Max Klinger replaced Radar. Max was rarely able to find requested files and often suggested that officers rummage through the file room themselves. This is how most interfaces are today.

Now, let us look at a theory for how computer agents can simplify users' work, and how computer agents can offer some of the behavior of a Radar O'Reilley.

Imagine you are Radar's Colonel, Radar augments your memory by remembering where all the files are kept and having them ready when you need them. You don't need to remember their name or storage location.

Radar improves your communication through his keeping in touch with Sparky and the other clerks around Korea. He can find out things that will help you, which others forgot to mention. He also knows about other people who have interests that intersect with yours and others in the unit, he thus can help with bartering and other forms of exchange of value that begin with communication.

Lastly, he can enhance your reasoning by allowing you to stay focused on the task, while he manages the communications and data storage details. He can help you anticipate outside changes (such as incoming wounded) which will alter your current work tasks, so you are always ready to proceed to the next task.

It is important to notice that Radar gains this knowledge of what information you will need next, just by watching you at work. Radar has realized that when you are working on something you also are likely to need related files. He is able to learn to anticipate what flies relate to your current work based on what flies you used together before. He uses that information to "pre- fetch" the handful of files. Then a one or two word cue is sufficient for him to choose just one from the handful. Radar also can anticipate what work will require interaction with others by noting where it came from, who it might be shared with, etc. But he does this unobtrusively without having to ask constantly or having to be told. How different from most computer systems today!

AN EXAMPLE OF PRESCIENT AGENTS IN X

In this section, let us take a look at the conceptual features of a Prescient system, along with the features of a prototype system in X, MindShare, created by David Williams, Mike Monegan, and myself at Hewlett-Packard in 1989. The original impetus for the work was to try to find alternative ways of enhancing the usability of computers for developing commercial software. We quickly observed that many problems that distracted software developers were common to many computer users. This led to a study of computer usage patterns and error patterns. Out of this study came our plan to address these problems through the creation of a prescient computing system. We began with a scenario, "A Scenario for Future Software Development", which 1 delivered at the 1989 GroupWare Workshop at Xerox PARC as part of the World Computer Congress. We also built a prototype of MindShare, to test the initial theory of prescience.

The major conceptual features of our system are:

- The enhanced window manager system menu,
- The links' pop-up menu,
- The ad hoc link creation mode,
- The link attributes' dialog box, and
- The WebLenses (linkage viewers), including
    - The neighbors' menu list,
    - The ToDo list manager and
    - The node and arc representation.

Our major goal was to augment the normal window management with a form of prescience that understood that various windows were related to each other, allowing them to form work contexts that can be saved, restored, searched and retrieved. Other systems use the notion of "rooms" to show contexts. Some limited abilities to restore context states across reboots are possible for some of these systems. A fundamental difference between a "rooms" system and MindShare was that in a rooms system it is up to you to decide consciously how to organize your work contexts. In MindShare, work contexts are derived observationally form everyday usage with no other user action required. Users could al specifically show associative relationships between windows (and by implication their underlying files) manually.

HABITS AND PRESCIENT AGENTS

Jef Raskin, a designer of the Macintosh User Interface, has pointed out the critical importance of habits to understanding successful user interfaces. Our experiences bear out his views. Wherever people can leverage old habits they learn quickly. Where they can quickly develop

new habits (similar things done similarly, different things done differently) task performance is improved. Where user interfaces frustrate habits (dangerously different things done similarly) we find errors increase and performance decreases as the user has to think about what an input might do. E.g. if Ctrl-P always means "print" you might type it automatically when you wanted a file printed. But if it sometimes means "purge" (non-undoable delete) selected object" your habit would be frustrated because you couldn't afford to learn this as an automatic motion.

While many anticipatory systems have been good about habit formation, a few have not. This leads to the wide spread fear that anticipatory systems will in fact confuse the user because the state of the system is not predetermined. We have found it is not the predetermination of the systems that is important to usability, but instead their predictability and their ability to form useful habits. Predictability and predetermination turn out not to be the same. Take for instance a menu of links. If the menu is ordered in terms of most recently used items, then which item will be at the top is not pre-determined, yet habits can be easily formed—particularly to retrieve the last used item. If the cognitive cost of being wrong in your selection is low, this habit can be formed quickly and become an automated part of working despite its non-determinism. In our system we took great care to use prescience only for non-destructive activities such as pre- fetching information or ordering menus.

GOALS FOR THE MINDSHARE PRESCIENT AGENTS

How could a standard X environment be modified to be more prescient? Imagine you could work on a task without ever having to remember the names of the files you are working on, nor where they are stored. Imagine you could find out about all the related files to what you are working on, even those created or modified by others. Imagine you could instantly return to the set of windows you had up two months ago, or even to the set of windows that a former colleague was working on last year. Imagine a Radar O'Reilley for your desktop that kept your files ready for you-not interrupting you, but not making you wait while files were being sought.

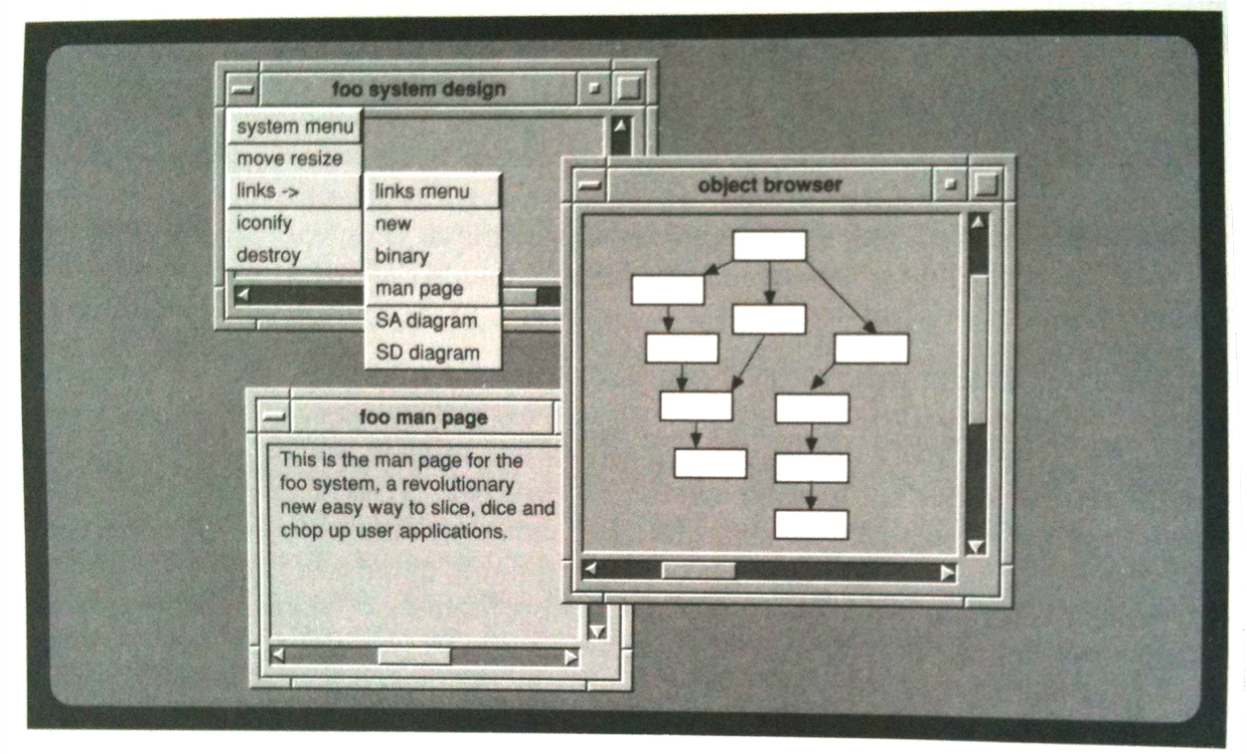
GRAPHICAL USER INTERFACE FEATURES

We designed our system to require as little additional learning from experienced computer users as possible. Our target environment was for users of the Motif window manager *(mwm)*. The Motif window manager normally provides a "system" pull down menu associated with each window on the desktop. From this menu you can normally access abilities to resize, move, iconify, and otherwise manipulate the current window.

LINKS

We added a new capability as an additional menu item, which allows you to traverse or create new links from the object underlying this window, to other objects. When you select the link menu item, an additional window is displayed that lists a menu of neighbor objects that are

linked associatively to the currently selected window. Clicking on an existing item causes the related object to be "re-animated" (program restarted on related data) using the proper edit or display method for data of this type. (See Figure 2).



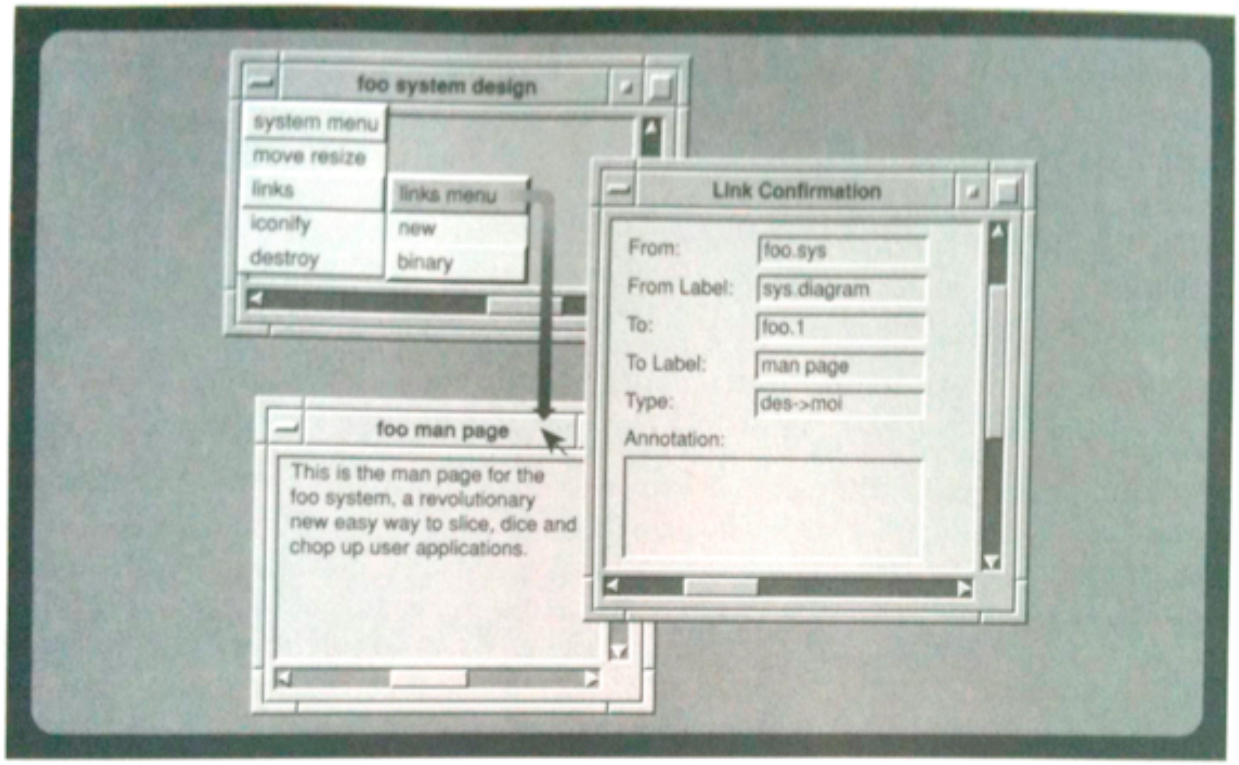
Traversing Links

Associative Links between objects can be used to easily re-animate a related artifact, causing it to be re-displayed for viewing or editing. The user doesn't need to remember a filename or directory path, nor the name of the proper program to use for redisplay or editing because the link object already remembers this. Thus, the user simply picks the related item and it re-appears. Complexity is reduced because the list of links is much smaller than the items that would appear in an object browser that does not distinguish among artifacts.

FIGURE 2: TRAVERSING LINKS

NEW LINKS

The menu also contains a button that will allow you to create a new link. New links can be created in two ways. If the destination of the link is an object with a currently visible window, then the user can merely mouse over to the other window (the sprite changes to a cross-hair image) and then click. The system will automatically compute the underlying object and other attributes of the link according to a set of user definable rules. It is also possible to fill in a dialog box and specify a link to an object not currently visible. (See Figure 3).



Creating Links

Users can create ad hoc links by simply requesting link service from the system menu and then pointing at another visible artifact and clicking again. A non-blocking dialog box appears with information about the link that can be edited by the user. Defaults are derived using rule-based heuristics. Link creation must be fast and simple if people are to create many ad hoc links. The solution here seems to work well if the server is responsive. In addition to ad hoc links, other links can be generated automatically by agents triggered according to additional heuristics. One important agent is the context agent that derives implied links from spatial and temporal co-location. Thus the user can get the benefits of a wealth of artifacts very inexpensively.

FIGURE 3: CREATING LINKS

When the user creates a new link, MindShare displays a dialog box allowing the user to specify labels and attributes of the link. It creates defaults for each frame automatically according to user specified rule sets, and it is common to dismiss the box immediately by confirming the pre-filled values. But, the dialog box is non-blocking and can be left around to be manually modified at any other time, so that labeling can be deferred. This is important because usually you create links as a by-product of work in progress. Having to interrupt this goal directed work to specify attributes at creation time can be distracting from the main task. Keeping the creation of links "cheap" in terms of cognitive load on you is important. If creation of links is too cumbersome,

you will rarely or never bother to use them. Since their value is related to their frequency of use, this is an important usability design goal.

WEBLENSES

The advantage of the neighbor's list is its compactness. When the pull down is not selected, it does not take up any screen real estate. When pulled down, it shows only the related links for the currently selected window. Mostly, that is all you want-you are working on one thing, and you want to now switch to a particular other thing. You want that selection list to be minimal and easy to manage. Occasionally there are other tasks that you want to do where this type of retrieval is not best. So it is useful to create other display forms and filters. We called these WebLenses.

One WebLens, the Graph WebLens, allows the entire transitive web of associations to be displayed as an active node and arc diagram. We display objects as node "buttons" and links as arcs, in a widget we called XmGraph, developed by another member of our team, Luis Miguel. You can re-animate objects by clicking on the nodes, inspect link attributes by clicking on arcs, and create and destroy links by editing the network representation.

For most work the Graph WebLens is overkill. Rich association webs grow quickly and soon the webs are too complex to manage. Mostly, the neighbors' menus provide much less complex views that are far more useful. Sometimes (such as when determining the full impact of a 2-line change to one program) there is no substitute for the richer view. It is precisely the magnitude of the complexity of the situation you want to determine in such cases. Still, it is usually helpful to reduce complexity where possible by hiding or filtering out nodes and links that are not of interest. WebLens filters allow characteristics of links and nodes to be specified for inclusion or exclusion from the current graph.

TO DO LIST AND AGENDA LIST

Another very important set of WebLenses is the ToDo list and Agenda list WebLenses. These are simple menu lists, but instead of attaching to a particular window, they belong to the overall context (root window). Selecting one item on the list does not merely re-animate one object; it re-animates all the objects in that context. This is equivalent to re-establishing yourself in a new room. This is useful if you want to be able to return to the same screen set up you had two months ago when you were working on a different project (a typical ToDo list use). (See Figure 4).

In meetings, this can be useful for managing switching between multiple presentations (a typical Agenda list use). (See Figure 5).

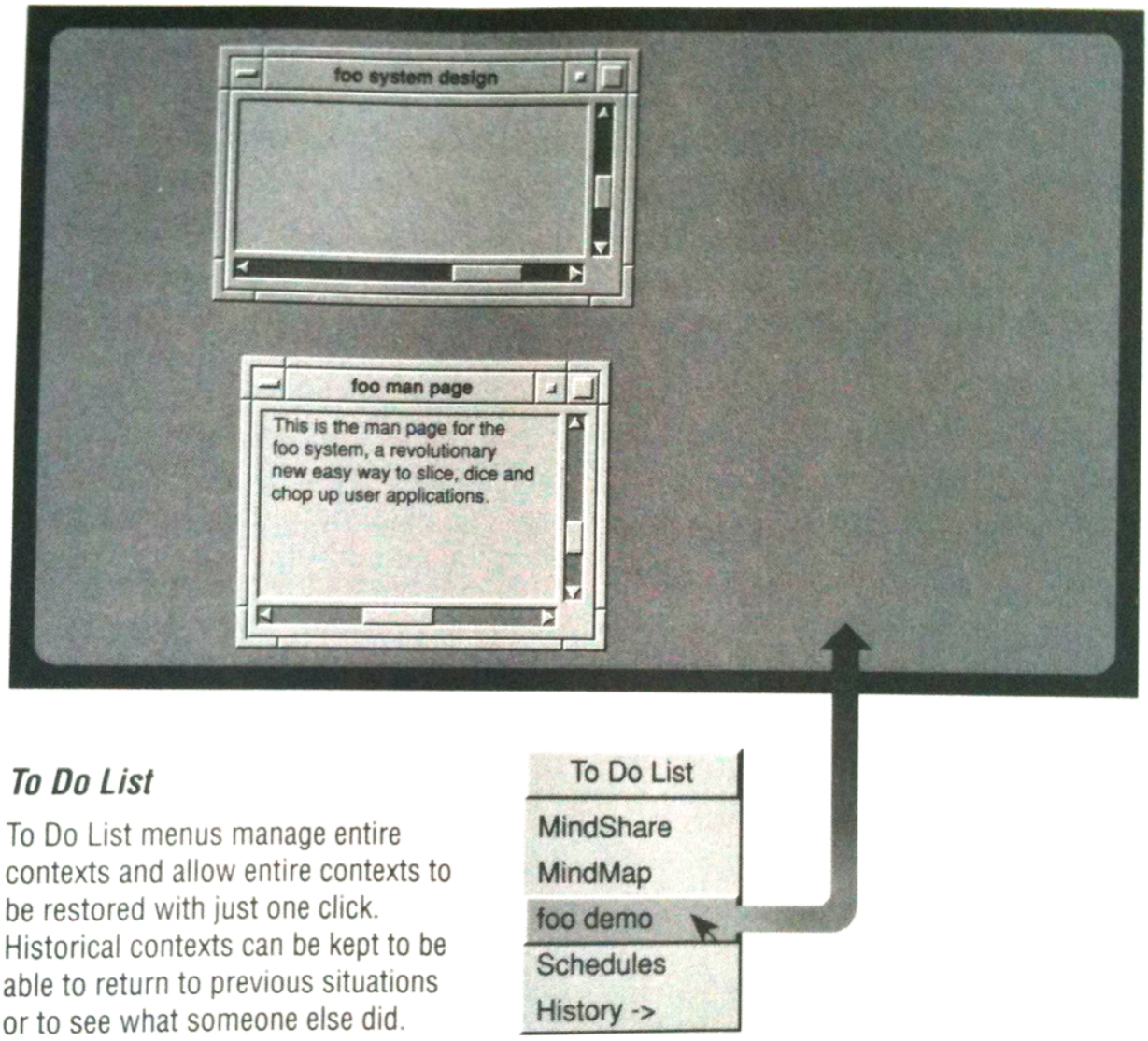


FIGURE 4: ToDo LISTS

AUTO-ENCAPSULATION

Prescient agents can be created to be evolutionary by providing an encapsulation feature. This can be done by creating and associating a new object with existing executable flies (which act as methods) and data files on which they act (private data stores). (See Figure 6). In some environments this would require a one-time manual change for each new artifact to be created. By using a session manager (see below), we can monitor your existing commands to the base operating system and automatically register them as you use them. We call this auto-encapsulation and it allows the system to learn the objects that you use without you having to instruct it. This is analogous to other learn-by-doing systems and to the way Radar learned about his commander. Radar O'Reilley learned to recognize contexts as his commander did work; his commander did not define and explicitly manipulate these contexts by name beforehand.

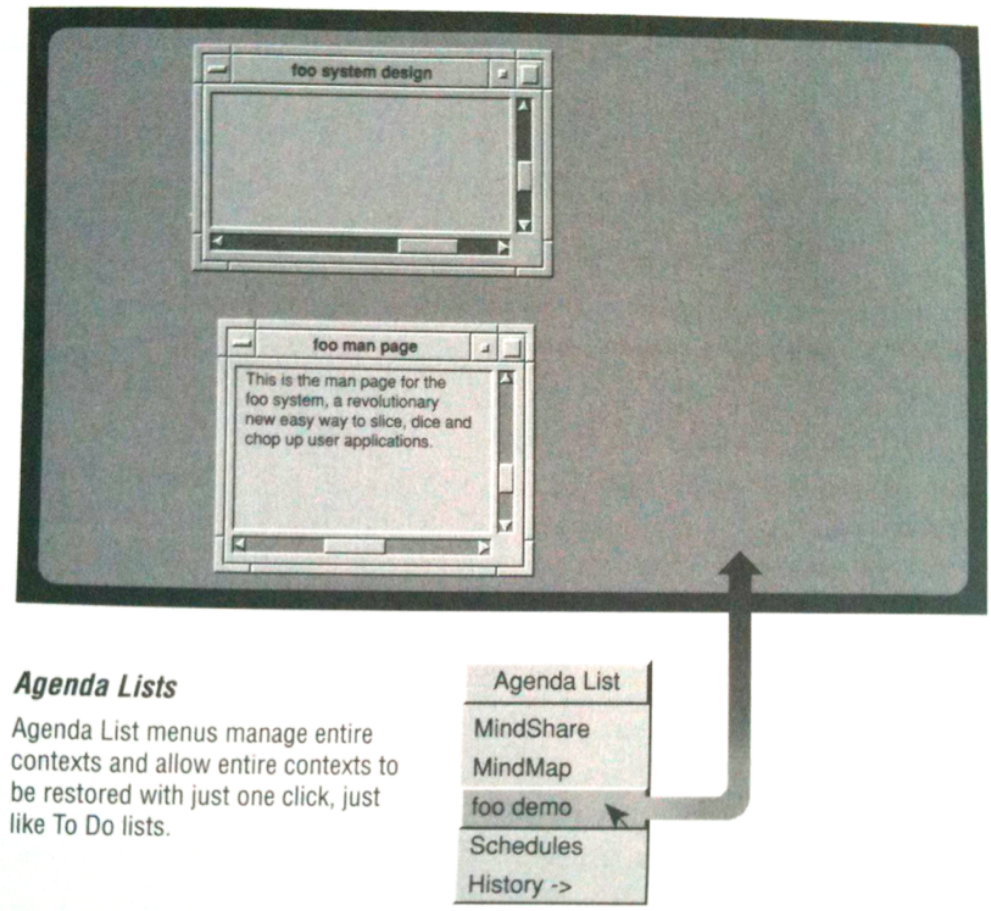


FIGURE 5: AGENDA LISTS

RULE BASED AGENTS AND PRESCIENT AGENTS

Invisible agents also can create links automatically in the background as you work. User- configured rule sets control and trigger agents. In the software development domain, we had an agent that could be run every time a user edited a source file. The agent would scan for "include" references in the source and automatically create links to the referenced include files. Once it creates the links, it is trivial for you to bring up windows for each include file when debugging.

The most important of the invisible rule-based agents are the prescient agents that build links according to rules concerning the spatial and temporal collocation of windows on the screen. These can be used to automatically intuit the need for a link between a source file and the associated design diagram whenever both appear on the screen simultaneously. The use of co-locality and co-temporality to deduce associative work relationships is very important to giving the illusion of prescience and the Radar O'Reilley effect. Much more on this topic could be said than can be covered here.

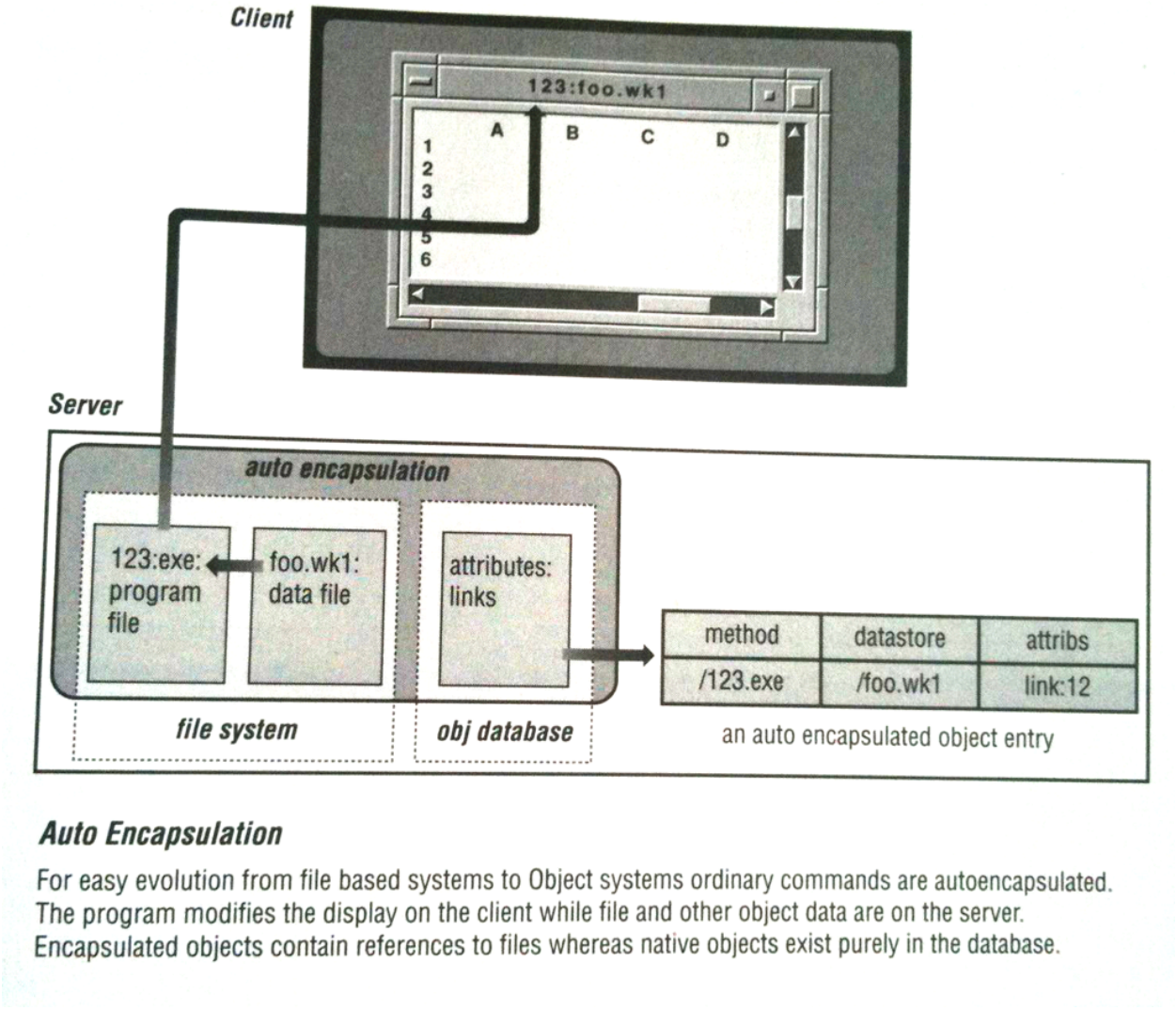


FIGURE 6: AUTO ENCAPSULATION

SELF AWARENESS, SHARED CONTEXTS AND PRE-FETCHING EXPECTED INFORMATION

One of the novel aspects of prescient agents is that they appear to be self-aware. When a user's action changes the state of the system, the system not only performs the user's request, but the prescient session manager notes the action and the corresponding changes. This allows the system to notice changes to the current context by noting additional new windows added to the current display (spatial and temporal co-location). These could introduce a new artifact into the context, or could indicate a switch to a new context. The session manager can then use this information to pre-fetch other related artifacts or to pre-fetch a new context. The prescient session manager also notices if any of the artifacts are shared with other collaborators. It can then determine if there are any shared contexts with that collaborator that should be pre-fetched (see Figure 7). By pre-fetching information, instead of waiting for a specific response, you avoid the usual waits for data-retrieval that are common with systems today.

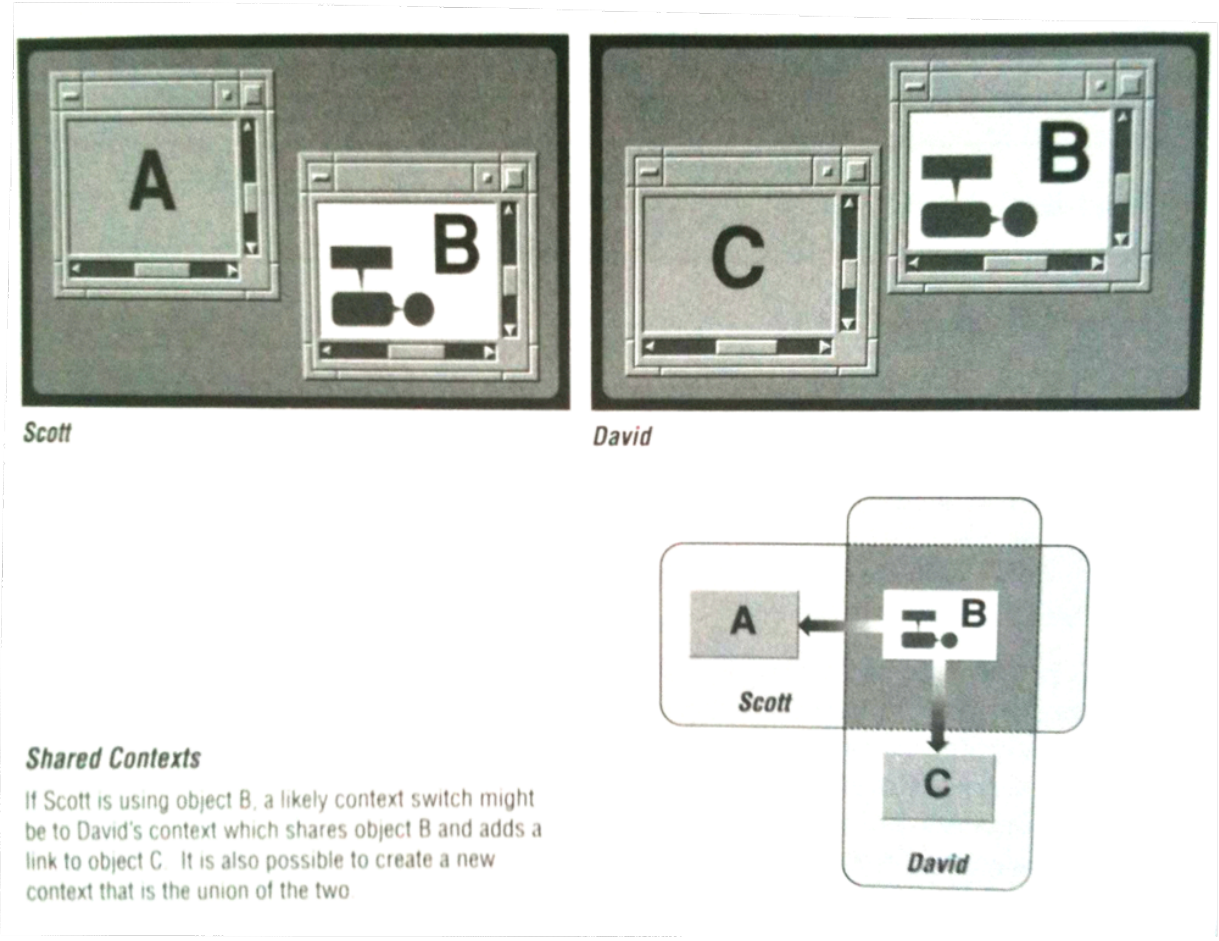


FIGURE 7: SHARED CONTEXTS

MULTIMEDIA

A system for managing artifacts becomes more valuable as it manages more artifacts. We felt it important to support not just text, but computer generated and scanned in graphics, animations, digital voice, and video. It must be easy to capture such non-textual information or people won't bother. People will often do a voice annotation of some text or animation that they won't take the time to write about. Or they will scribble a note on a piece of paper but not re-type it. These are still valuable artifacts, so it is desirable to bring them into the system for the historical and communication record. Thus a nearby scanner or microphone can lead to easy capture of more valuable artifacts. (See Figure 8). Some of the searching capabilities must be changed when dealing with non-textual artifact: you can't do string searches. But, you still have the associative link paths. Additional searching capabilities such as fast forwarding or indexing on voice and video segments could add more value. People may be moderately deterred from accessing such media, but this is okay. A natural equilibrium will form naturally between the difficulty in locating a particular piece of information and its value to the current task.

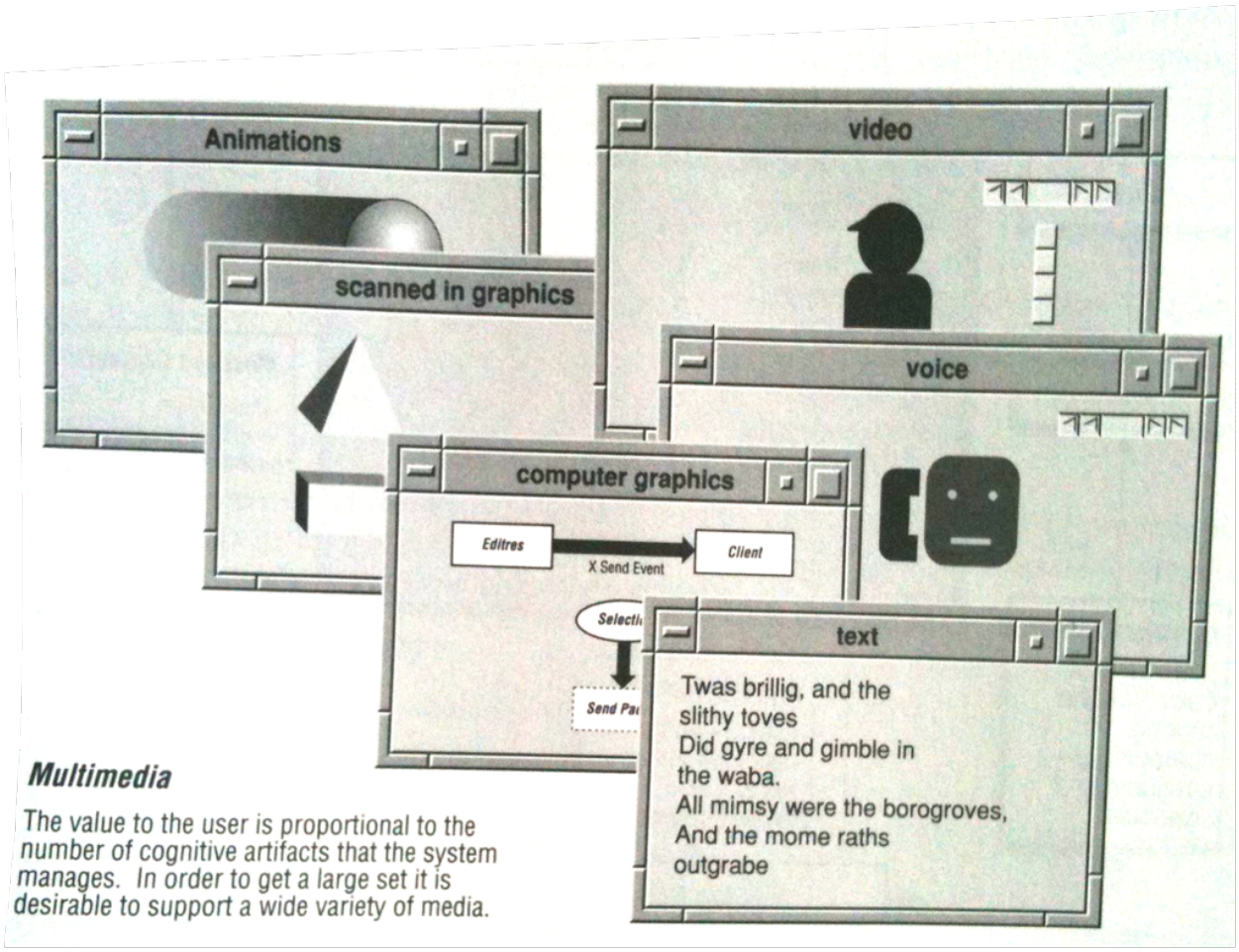


FIGURE 8: MULTIMEDIA

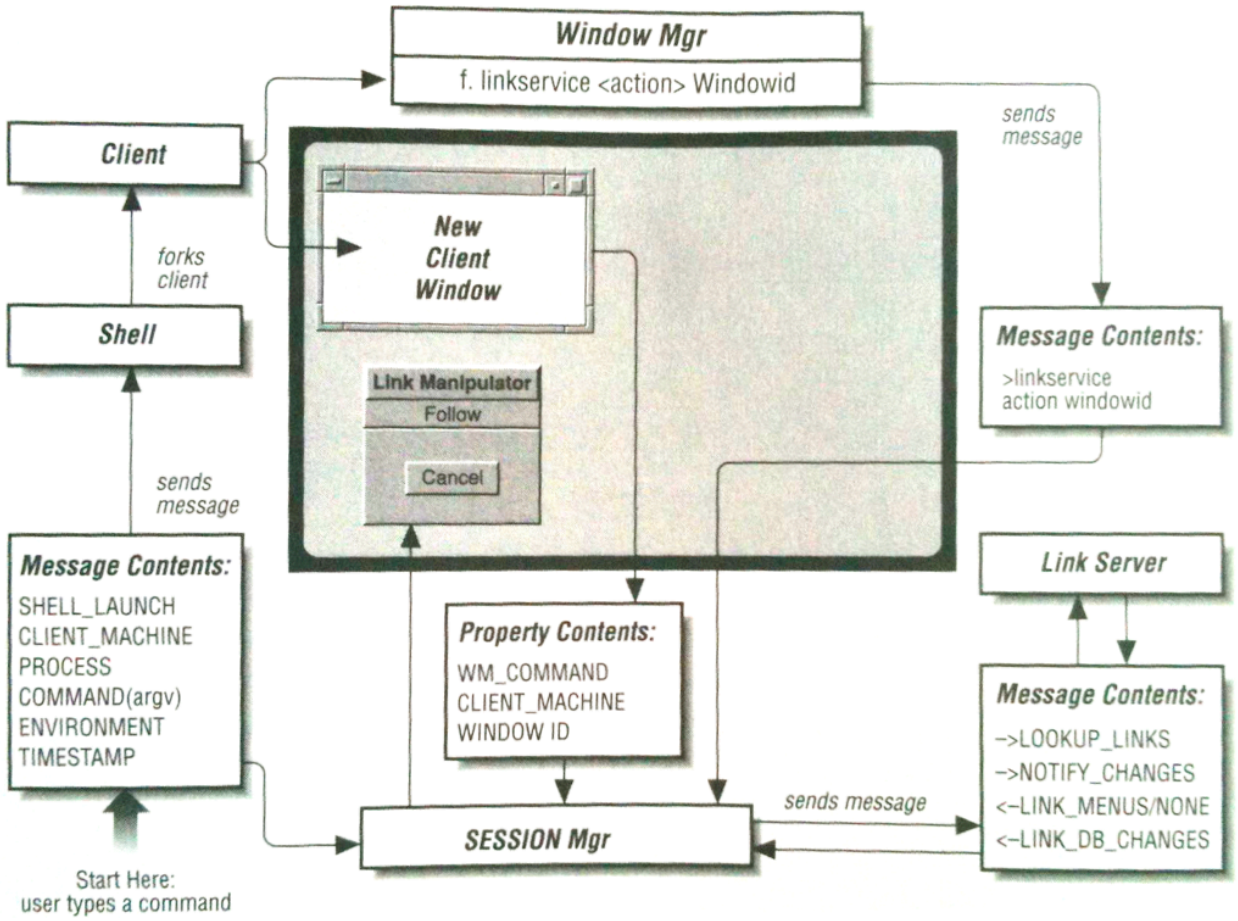


FIGURE 9: ARCHITECTURE

ARCHITECTURE OF MINDSHARE, OUR EXAMPLE SYSTEM

Specific implementation details about our system are too lengthy to include in this article, but I have included an implementation overview diagram for reference, (see Figure 9). Major components include the following:

- The X session manager watches for window events that signify a new active object. Information sufficient to identify the object is in window system tables. The object is then looked up in the link database. If found, information concerning its neighbor links is pre- fetched. If it is not found, then we automatically create a new object entry using auto- encapsulation.
- The Softbench Broadcast Message Server (really a subscription message redistribution server) communicates among interested agents.
- The link server interfaces with an object-oriented database repository that contains information about the links and objects. Typically this is on a remote database server and in communicated over a LAN from the session server that runs on a bit-mapped workstation.

Some special agents and maintenance tools can access and manipulate the OODB directly using an object oriented SQL extension. Other agents can post access requests and link- creation requests via the standard protocols supported by the broadcast message server.

Note that in this example system, the objects consist of methods and data stored in the file system, accessible to standard Unix utilities. When following a link from one artifact to another, the best you can do is link to file level objects. We call this large granularity. By creating objects that are smaller, such as paragraphs in an article or statements in source code, it is possible to make even more useful links between artifacts because the links can be localized. At its extreme, this small granularity becomes a full hypertext system. We can protect these objects from unpredictable changes made directly to the underlying file systems, but they lose the capability of being acted upon by existing file-based tools.

RELATED WORK

The ideas for prescient agents did not arise in a vacuum. We acknowledge a variety of sources that lead to the work described here.

DOMESTICATING THE COMPUTER

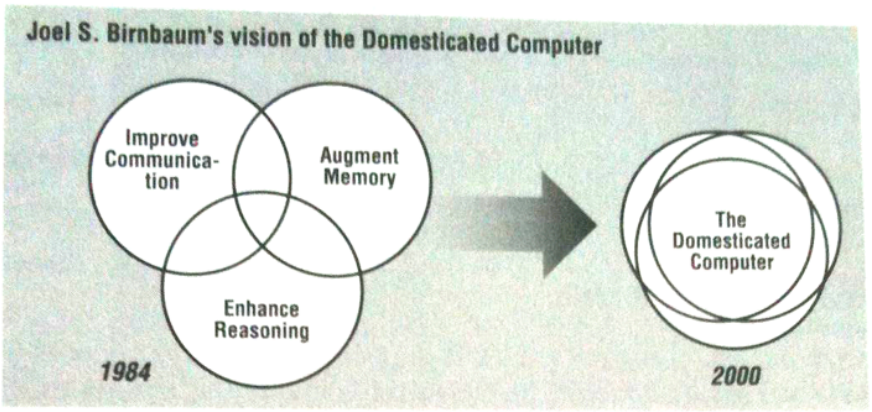
In order to analyze our metaphor for prescient support, we adopted a model from Joel Birnbaum, VP of R&D at Hewlett-Packard, in which he gives three capabilities that computers must give to humans for computers to become "domesticated":

- Augment Human Memory.
- Improve Human Communication.
- Enhance Human Reasoning Ability.

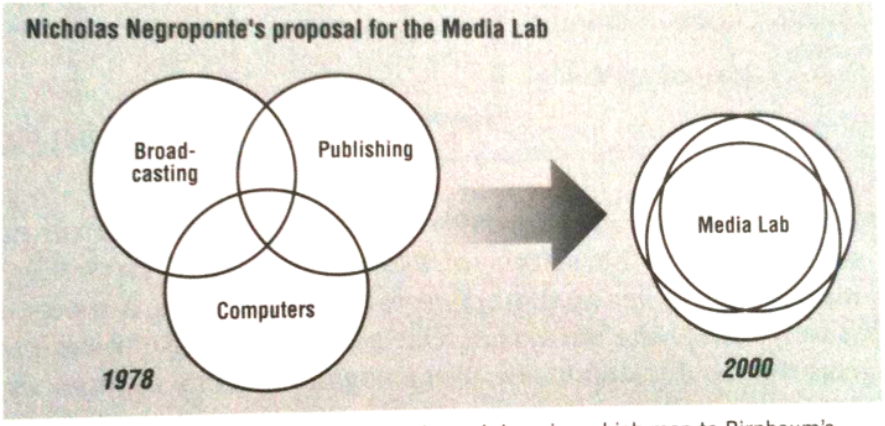
As we have seen, prescient agents focus on the first two areas, and through them contributes to the third.

There are two additional human capabilities to be concerned with, human perception and human motor actions, notably the subject of Donald Norman's "Psychology of Everyday Things." In our model, a computer monitor displays the system state. A mouse and keyboard allow you to physically manipulate the system. Our goals were to use these interfaces to create prescient agents that could support the user's cognitive needs of memory, communication, and reasoning.

Domesticating the Computer



Note how Birnbaum's Improving Communication is supported by the Broadcasting and Telecommunications Industry, Augmenting Memory is the domain of Publishing and Enhancing Reasoning is the focus of the computer industry in Negroponte's Model.



Within the computer industry, there are also subdomains which map to Birnbaum's model. Specifically, Networking supports improved communication, Databases support augmented memory and user agents enhance reasoning.

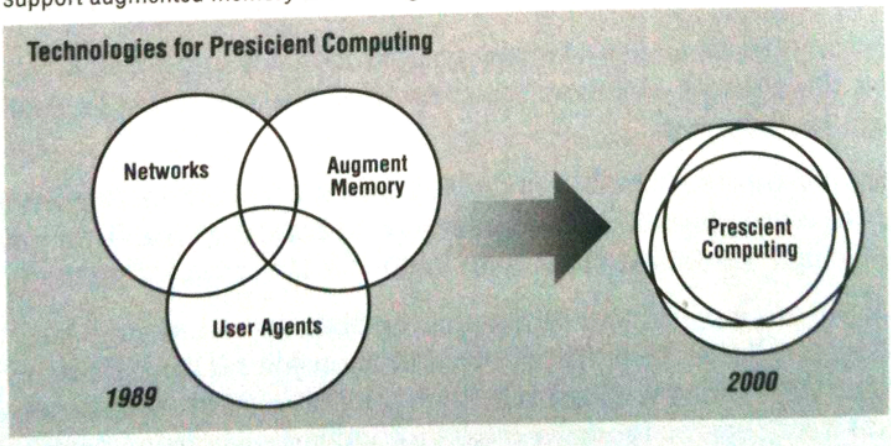


FIGURE 10: DOMESTICATING THE COMPUTER

MEDIA LAB & DOMESTICATING THE COMPUTER PRESCIENCE: A CONCORDANCE

While Birnbaum's model describes a worthy goal, it does not suggest how this might be accomplished. It is illustrative to combine Birnbaum's model with other models of how those capabilities might be delivered.

In 1978, Nicholas Negreponte, director of the MIT Media Lab, created a figure that illustrated his prediction that Broadcast/Telecommunications, Publishing and Computer industries would become a single combined industry by 2000.

In Figure 10, I have also mapped Joel Birnbaum's capabilities on the same kind of diagram. It is easy to see the parallels between the two models when you recognize that published documents are a major way of augmenting human memory. The Telecommunication and Broadcasting industries have been our primary means for improving human communication. Computers, through programs as diverse as linear programs and VLSI simulators, have been automating tasks and enhancing our reasoning.

It is also illustrative to map these two disciplines within the computer industry. Here we see that databases provide a place to store information akin to publishing, while networking software supports communications. User Agents provide technology to enhance reasoning. This provides guidance on what technologies to use in building prescient agents.

CONCLUSION

The principles of prescient agents are not new cognitive principles but instead the basic principles we have long known, now applied in a different way. Most of the activity at the computer keyboard can tell you a lot about the primary work tasks, but our passive systems have done their work without examining or recording this information. Prescient agents allow a new form of support of the windowed user interface user, not merely a passive slave interface, but active, helpful agent-based task support. As such, they represent a logical next step for today's desktops, rooms, and browsers, implementable with today's X technologies.

Readers may wonder how they can get a Radar O'Reilley for their computer, and why one is not standard with their window manager today. It is not the technology that is missing: the building blocks such as versioned file systems, hypermedia systems, object oriented databases, rule- based systems and windowed user interfaces already exist. They can be combined into the sort of system outlined here, just as we did with our prototype. We are starting to see components of this technology leaking out from various companies today. Considerable work in user interfaces and CSCW has been published lately. But sociological understanding is lagging technology somewhat. There are also the normal marketing, positioning and pricing issues for each vendor to work out.

A major problem that has made availability of this technology difficult has been the lack of specific customer demand for it. Users don't know how to ask for it, nor who to ask. Developing a common vocabulary and sample prototypes systems that users can point to should help.

If you want a Radar O'Reilley for your window manager or software application, you will have to express your needs for such systems to the software vendors. And vendors need to understand your needs in terms of their strengths and weaknesses to begin envisioning new opportunities for such systems. Awareness of prescient agent technology may come slowly, but will provide substantial value when it arrives on your X desktop.