

Collaborative Aspects of OpenDoc®

Elizabeth Dykstra-Erickson
Apple Computer, Inc.
1 Infinite Loop MS 302 1HI
Cupertino CA 94110 USA
+1-408-974-6462
eade@apple.com

Geoff Schuller
Apple Computer, Inc.
1 Infinite Loop MS 302 1HI
Cupertino CA 94110 USA
+1-408-974-1193
schuller@apple.com

Dave Curbow
Apple Computer, Inc.
1 Infinite Loop MS 302 1HI
Cupertino CA 94110 USA
+1-408-974-4823
curbow@apple.com

Kurt Piersol
Apple Computer, Inc.
1 Infinite Loop MS 303 3A
Cupertino CA 94110 USA
+1-408-974-1201
piersol@apple.com

ABSTRACT

This paper discusses Apple's commitment to collaboration in the design and development of OpenDoc. Collaboration was designed into OpenDoc as a result of research, good design principles, and market surveys conducted prior to the design phase which indicated that collaboration is important to users. OpenDoc 1.0 supports asynchronous collaboration; in addition, its architecture is designed to support real time collaboration, which will be accessible to users in future releases. This paper is an exploration into design problems for user interface elements that support collaboration, describing collaboration features and the user experience.

Keywords

OpenDoc, collaborative work, interaction design, design rationale, component technology, objects, drafts, annotations, electronic signatures

INTRODUCTION

This paper discusses the aspects of the OpenDoc user interface that support collaboration. Our design objective for providing collaboration features in OpenDoc was to allow users to easily annotate documents, and to provide an efficient and easy-to-use means for creating and keeping track of multiple versions of a document. Architectural issues that support collaboration include how to handle the event stream, and how to synchronize displays among multiple users.

A BRIEF REVIEW OF OPENDOC

OpenDoc is a cross-platform component technology development standard available on Microsoft Windows, Macintosh, OS/2, and AIX. With OpenDoc, software developers create parts that interact together in compound documents. A compound document architecture poses interesting interface design problems for providing users

with sufficient visual feedback to identify boundary conditions and state changes of interacting parts. The visual and interaction design of OpenDoc marks several departures from the traditional Macintosh Human Interface Guidelines in order to accommodate a new paradigm in human-computer interaction. Examples of these departures include the two state model of documents (they exist, or not) replacing the three-state model (unsaved, saved, deleted); the inside-out model of selection (click on something to select it) replaces the outside-in model (to select an object, you first select whatever it's a part of and work your way down); and the removal of the Quit command (OpenDoc parts start and stop dynamically, when they're needed). A number of significant changes are noted in other publications (Piersol, 1994; Curbow and Dykstra-Erickson, 1995).

OpenDoc presents a paradigm shift for both users and software developers: the developer breaks down a monolithic application into many smaller, more efficient part editors that the user then combines within compound documents. Using OpenDoc, the user works directly with content, without manipulating the applications that create it. OpenDoc parts can be dragged and dropped into and out of documents and can be directly edited within the context of the compound document in which they are contained. This move toward document-centric computing, and away from application-centric computing, follows in the interface footsteps of the Xerox Star (Smith et al., 1982).

USER REQUIREMENTS FOR COLLABORATION

Requirements for OpenDoc collaboration came from a variety of sources. We conducted user studies from earlier projects including the progenitor of Taligent, Apple's "Pink" project; brainstormed ideas; conducted market surveys; and held discussions with specialists in domains such as publishing, small business enterprises, and higher education. The OpenDoc features that specifically support collaboration were designed based on precedents set in previous software designs and concepts presented in the literature.

Collaborative software has various properties with which single-user applications need not necessarily be concerned.

For example, collaboration has been categorized in the now-familiar 4-square map of place and time (Johansen 1989) as asynchronous, synchronous, and same or different place. Most collaborative software products offer shared work space, implemented as shared screens or shared windows; some products offer a means of annotation, implemented as shared “sticky notes,” annotation layers, or with screen real estate laid out with a specific note-taking area. Other characteristics of collaborative software include a floor control protocol, visual feedback for the boundaries between shared and private work spaces, allowances for privacy and anonymity of shared commentary, and affordances for synchronizing multiple users’ displays (Grudin and Poltrock 1992). Each of these properties requires design decisions which we address in turn: OpenDoc supports asynchronous collaboration; all work space can be shared via annotation as embedded parts; the floor control protocol is turn-taking, although OpenDoc is architected for software developers to easily accommodate multiple input event streams; special borders indicate the distinction between parts, and authentication/authorization allows users to attach read-write permissions to parts; and OpenDoc parts can display simultaneously in multiple locations.

COLLABORATION ARCHITECTURE

How do multiple users work on the same material at the same time? Three particular architectural decisions are important here: how to implement floor control protocol, how many users can input events simultaneously, and how to synchronize displays for multiple users.

Floor Control Protocol

Deciding a floor control protocol has consistently been an important implementation issue in computer supported cooperative work. OpenDoc’s model of collaboration is a single cursor, non-legislated first come-first served protocol. This reflects our belief that people are smart enough to figure out who’s doing what and when they are doing it, as long as they have an appropriate communication channel to converse about what they’re doing. In other words, mechanical legislation of control is essentially unnecessary since human beings can do a much better job of it (for a similar discussion of mechanical legislation, see Dykstra and Carasik 1991). As a result, in OpenDoc, we concentrated more on features for sharing and identifying changes than on control mechanisms.

OpenDoc has several architectural elements which allow various models of synchronized collaboration. There are basically two aspects to real time interaction between humans and their tasks (or content) in a standard GUI. The first aspect is the stream of events by which the user controls the task, and the second is the graphical display by which the program makes information visible to the user.

Event Stream

A key decision about the OpenDoc collaboration model is how many users are allowed to operate on a document at one time. OpenDoc allows only one user to submit input events to a part at a particular moment. This greatly simplifies the part editor developer’s job, since multiple

selections and interacting commands needn’t be considered because only one source of human interface events is supplied to the parts within the document. Multiple cursors can certainly be implemented in OpenDoc, but they are not provided “for free”— multiple cursors are an option for the part editor developer to provide, if they so wish.

OpenDoc manages the stream of user input events through a single object, called the Dispatcher. The Dispatcher has a standard API for dispatching events, and may be modified at run-time. The presence of this single “event gatekeeper,” written as an object, means that encapsulation can be used to provide a synchronized event stream on several computers at once. In this model, all connected users see the same selection, the same keystrokes, and the same menus being activated. This can be achieved without any special code in the part editors.

We feel that this level of functionality is adequate for most real time collaboration tasks, assuming that the collaborators have a high-bandwidth channel like voice or video through which to coordinate their actions, in addition to the document itself.

Synchronizing Displays

The next key design decision has to do with how displays should be synchronized among multiple users. One method is to display synchronized versions of the internal data structures of the document. This method has the advantage of allowing different users to have different window sizes or scrolling positions within the document. Another method is to duplicate drawing commands to the display of each user; this method is simpler to implement. Either method is permissible in OpenDoc, without modification of existing part editors.

OpenDoc assumes that a given part may be imaging in several locations at the same time. This was originally intended to allow the same part to appear in multiple windows (a handy feature for displaying the same data in multiple formats). However, it can also be used to synchronize displays across machine boundaries. Each part is asked to keep one or more display locations, called a “facet,” always up to date. Each facet provides drawing information, including a geometric description of the drawing area and a pointer to a drawing context such as a window or off-screen bitmap.

An OpenDoc developer can either set up the OpenDoc facet structure to point to a replicated drawing context, or set up a facet structure for several windows, some of which are on another machine. Since OpenDoc delivers a synchronized stream of events, the event stream can be applied to a single window structure with replicated display, or to several windows simultaneously.

COLLABORATION FEATURES

Our studies concluded that the most important collaborative features to include in OpenDoc were 1) a means of keeping track of various versions of an artifact (eg, version of a document); 2) a means of annotating an artifact; and 3) a

means of verifying the authenticity, or state, or ownership of an artifact. These three concepts are discussed in further detail below. Thus, the basic collaboration issues that OpenDoc supports are: *Which version is being worked on? How do you know whether a document has been changed or not? Who made the last changes?* The design ideas behind OpenDoc's collaboration technologies address these questions through three features: drafts, annotation, and authentication and authorization.

Managing Versions with Drafts

Which version of this document am I working on? How many versions are there, who owns them, and is there a way to compare them to the version I'm working on now?

Requirements

As we gathered data, we found that users have two basic models for collaboration. The first model is linear: a user creates an artifact, such as a written document, and it is versioned in a linear sequence with contributions by various other collaborators. The second model is non-linear: a user creates an artifact, and other collaborators create their own branching versions of it, building a number of alternatives from which a "winner" is selected for further work. These two models were further elaborated on during the design phase of OpenDoc, and the first option is currently implemented as OpenDoc's Drafts feature.

While a branching drafts interface might more realistically accommodate the sometimes non-linear process of how people change and exchange contents in multiple drafts, we found in early user studies that such a non-linear history of n-dimensional changes was difficult for users to track, and they had difficulty thinking recursively.

Implemented Feature

At any time, a user may need to view an earlier draft of a document, or send a draft to collaborators for review and annotation. Some of the common problems of collaborative editing involve determining which version is current, deciding on a naming convention for version control, securing versions, and determining read-write privileges. OpenDoc 1.0 uses a drafts metaphor to allow users to save a history of the document within its file.

Consistent with our approach to floor protocol, we minimize collaboration's "interface overhead" by providing a menu command that creates a draft, which is a "snapshot" of the current state of the document. Once created, drafts are numbered and dated, and "invisibly" stored within the document, though they can be easily opened and viewed. Early user studies showed that users liked the idea of saving document drafts on demand within the same file. As drafts are created, they are numbered sequentially and users may create as many drafts as they like, and store comments about the draft, including a field for who created that particular draft. The Draft dialog reflects the list of drafts in chronological order, and displays the user's comments. Drafts are stored in an "economical" format: each draft is saved as the "set of changes that occurred since the last draft was saved." This takes up very little disk space and is all

OpenDoc needs to reconstruct a complete draft when the user chooses to open one. The stored draft *may not* be edited *after* it has been saved, since that would corrupt the accuracy of the draft history. In order to edit an older draft, the user must open and save a copy of the draft, thus creating an entirely new document with its own draft history.

OpenDoc does not currently provide support for reconciling multiple drafts, because there are many alternatives for implementing draft reconciliation, and reconciliation can be required at different levels of granularity. However, it is possible for OpenDoc developers to utilize the OpenDoc draft mechanism to build an appropriate model for reconciling or merging multiple drafts of a document.

Annotation

I want to mark up this copy; how do I attach notes to it, and how can I tell who wrote which notes? How are the notes implemented, as layers on top of content, or as part of the content itself?

Requirements

Another topic we discovered in the collaboration research was annotation styles. User studies have shown that users like the idea of attaching annotations to a section of a document, perhaps to describe the reasons for their actions. We learned that users depend on sticky notes and markers to make annotations on documents; the need for annotation is apparent. The "Pink" project built a prototype of annotation and conducted user studies to gauge response to on-screen sound-and-text notes. This resulted in the incorporation of an electronic analog of sticky notes which could contain various kinds of content, including sound, text, sketches, etc. This concept was further developed by allowing annotation *of* any kind of content *by* any kind of content, and helped us to deduce the concept of annotation as "simple content with a property of location." In this respect, annotation is fundamental to OpenDoc because any content could be perceived as an annotation of any other piece of content (based on the user's perception of the relationship between parts). This is especially important when working with parts in a multimedia document. For example, a user who is annotating a film score document should be able to annotate using sound, animation, video, charts, drawings, or even a simple text note — whatever is appropriate to properly convey their ideas.

Our prototypes of annotations allowed the user to display annotations using different views, so that the annotations could be viewed as icons and displayed to the side of the content, or viewed in full *in situ*. The transition from the open and closed states of notes was accompanied by an in-transition animation image ("zoom-recs"). The current design of annotation is discussed below.

Implemented Feature

OpenDoc does not prescribe annotation; instead, it delivers the capability to make annotations. The current design requires the document to provide the mechanism to make annotations visible; there are no system distinctions for

annotations because there are too many potential visual and interaction models for notes. Any OpenDoc part may be dragged and dropped into any other OpenDoc part, and depending on the content model of the containing part, the added content can be merged or embedded. The user can, however, indicate that the part should not merge, and should instead retain its own “part-ness.” An annotation can occur at any grain in OpenDoc, and can consist of any kind of content available on the user's system.

Depending on the part editor the user chooses, such embedded parts can look very similar to the sticky-notes used in the real world. OpenDoc may in the future provide some basic interaction, such as transparent layering, to support specific organizations of annotations. An alternative method to annotate content is to drop a “flag” part into a part to serve as an indicator that an annotation is enclosed within it. The behavior of annotations is dependent on the functionality offered by the editor which creates the part, and the editor which creates the annotation. The authentication/authorization process allows the user to indicate who “owns” or created the annotation. Annotation is fully supported in OpenDoc 1.0; authentication and authorization is under design and will be available in a future release.

Authentication/Authorization

How can I be sure that this document is signed off and approved? Is the entire document signed, or just a part of it? How do I control who can make changes to it? And how do I know whether I can make changes to someone else's document?

Requirements

A third topic we uncovered in our research was the need for authentication and authorization, including electronic signatures. Our marketing results implied that there was a great need for authentication and authorization in workflow applications. Thus, we thought of ways to provide authorization to the level of individual parts within a document.

Implemented Feature

Authentication and authorization are not implemented in OpenDoc 1.0, but are planned for OpenDoc 2.0. OpenDoc will, in the future, allow users to set authorizations by part or at the document level. Every part has properties that are documented in Part Info, a dialog accessible through the Edit menu. Basic part properties include name, whether the part is a document or stationery, the “kind” (data type) of the part, which editor created it, and how the user prefers to view the part (several options are available). There is also a By field that is not modifiable, which is a text string

identifying the user who caused the last modification to the part. The permissions feature is not yet designed in the interface of OpenDoc 1.0.

OpenDoc authentication identifies the user who is attempting to interact with a document and, once the user's identity is known, the authorization system grants the user a set of authorizations to interact with that document and its parts. For example, as the creator of a document, you may decide that only a particular set of users may read that document, and no one may modify it but you. Alternatively, you may allow users to annotate the document, but prohibit them from editing the original content. Thus, if a document reviewer wishes to make a private comment to you, that reviewer may set the authorization of one of their annotations so that only you and they can read it. As the owner of a document review process, a user can control which reviewers may see and make comments in the document.

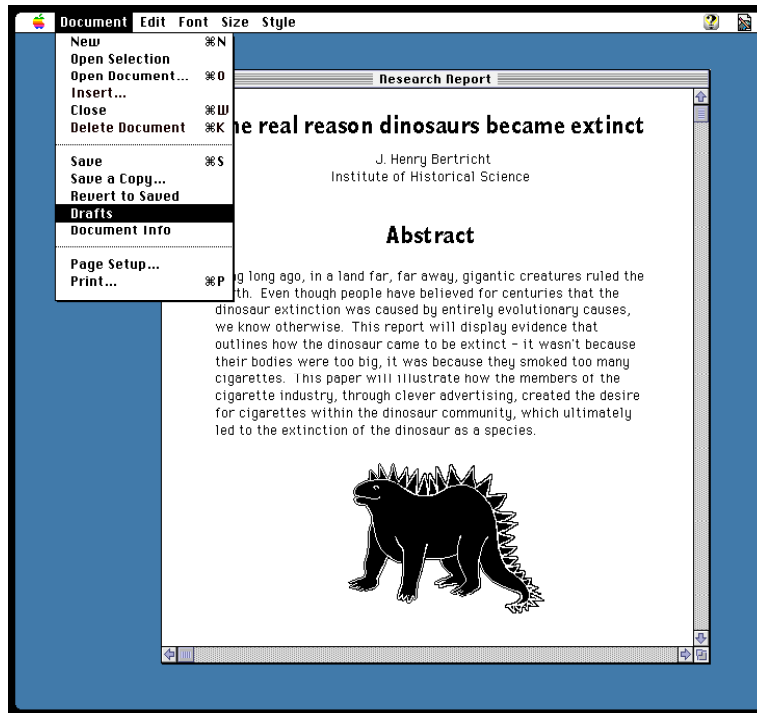
An electronic signature is a different kind of annotation, applied to an entire part, which may or may not always be visible to the user. For example, a signature annotation can be used to assert that a part (or a document) has been “signed” and the part's content has not been changed since the signature was applied. A signature is created based on the stability of content and a public encryption key. This way, an electronic signature could be checked for validity (which would be confirmed when it is confirmed that the part's content has not been changed).

Applying an electronic signature to a traditional digital document is straightforward; applying one to a compound document is more interesting. In a signed document, the signature is applied at the document level, which assumes that all of the embedded parts are equally signed. Because compound documents may contain unlimited levels of embedding, a part may actually carry many signatures. Signatures applied at the document level will include each embedded part, along with any signatures assigned to individual parts. This opens up a world of possibilities for sequenced activities that require signoffs, such as budget approvals, production quality checks, or basic work flow applications.

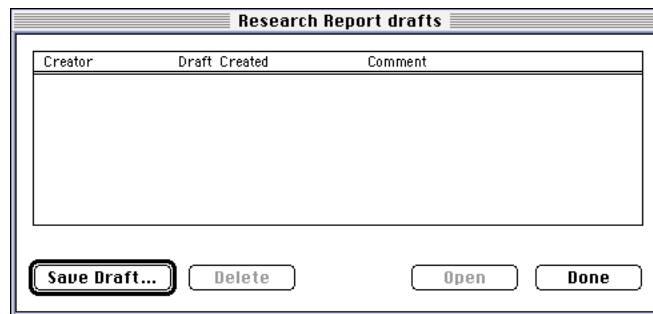
USER EXPERIENCE

To illustrate how users can make annotations and save drafts, we offer the following user experience scenario. In this asynchronous collaboration scenario, the user has prepared an article using OpenDoc and is ready for it to be reviewed by some team mates. Each reviewer is given access to the electronic version of the article. Here's how the drafts feature works:

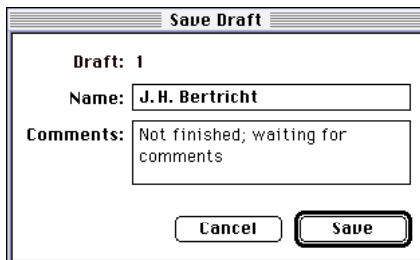
1. Use the **Drafts** menu command to save a draft of the article before the addition of any annotations. First select the **Drafts** command.



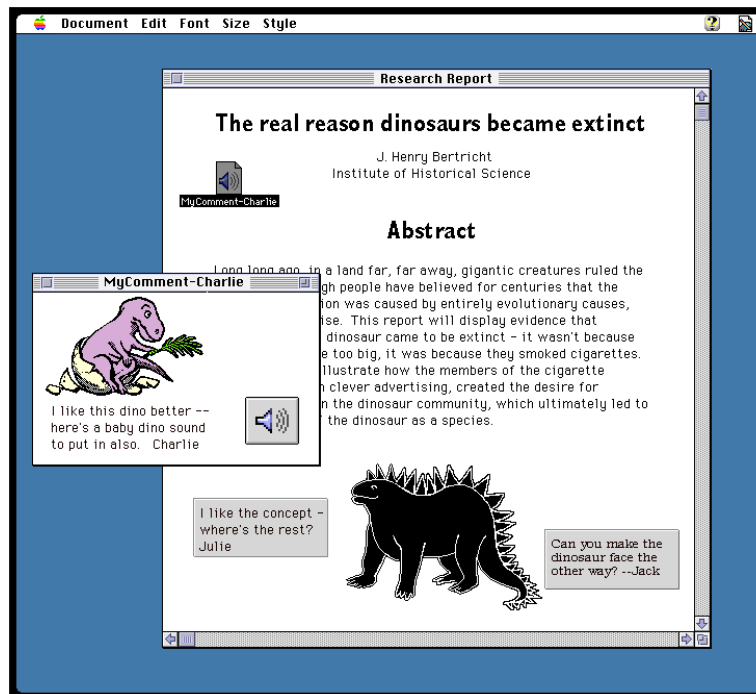
The Drafts dialog will be displayed. Click the **Save Draft...** button.



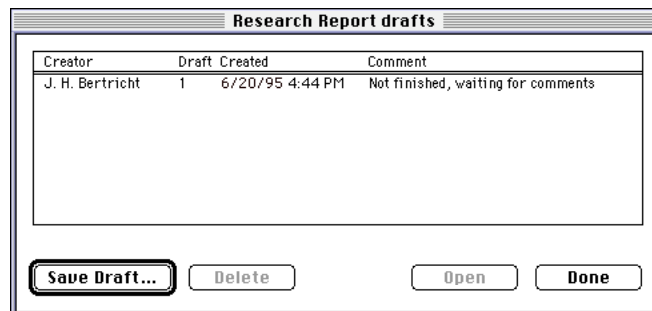
2. This brings up the Save Draft dialog. Here you may input your name and any comments you'd like to note.



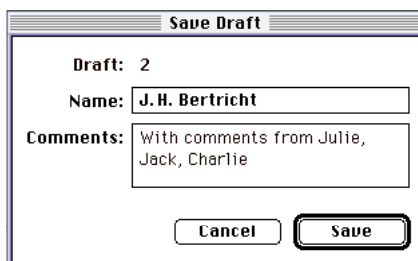
3. Your team mates use OpenDoc parts to add their comments and criticisms. The document now contains annotations from each reviewer. Because OpenDoc is component technology, there are typically no restrictions on the kind of content the reviewers may use for annotation.



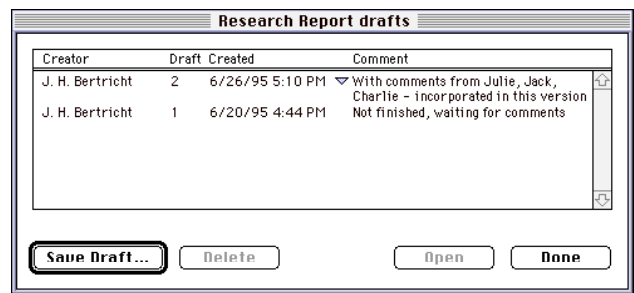
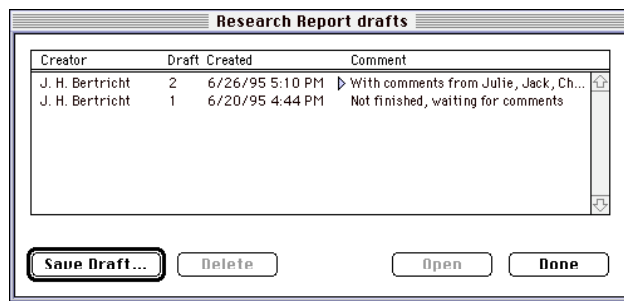
4. This is a good time to save the document as a draft again, in order to preserve this state of life as an article with comments. Note that the drafts dialog now records that there is already a draft #1; this is the draft you previously saved.



Click the **Save Draft...** button to create your second draft, and enter some comments.



5. After looking at the various annotations on the document, you can make any edits necessary. When you are finished, you may want to save another draft. Notice that as you begin the Draft process again, the drafts dialog shows you the two previous drafts.



You can access either of these by selecting the entry and clicking the Open button. This will allow you to review the contents of a draft, but will not allow you to make further changes to the saved draft. If you'd like to edit a draft, you must save a copy, thereby creating a new document. The new document will then have its own separate history.

CONCLUSION

OpenDoc provides a great collaboration foundation for both developers and users. And the component nature of OpenDoc allows part editor developers to leverage this foundation to provide additional specialized collaboration parts, which support more complex collaboration techniques. Various parts from different sources may be used together, even if some of these parts were not specifically built for use in collaborative environments. At this writing, over 120,000 copies of OpenDoc have been shipped to developers. Over 200 parts have been created in workshops with Apple's support. The best part is that each of these can work together in the same document.

ACKNOWLEDGMENTS

We would like to acknowledge David Austin, Paulien Strijland, George Corrick, Kerry Ortega, and Dave Smith for their comments and support.

REFERENCES

1. Curbow, D. and Dykstra-Erickson, E.A. The OpenDoc User Experience, *develop* (Apple Computer, Inc.: Cupertino, CA), Vol. 22, 1995, pp. 83-93.
2. Dykstra, E.A. and Carasik, R.P. Structure and support in cooperative environments: the Amsterdam Conversation Environment, *International Journal of Man-Machine Studies* Vol. 34, 1991, pp. 419-434.

3. Grudin, J. and Poltrock, S.E. Computer supported cooperative work and groupware, Tutorial Notes, *CSCW '92: Sharing Perspectives, Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work* (Toronto, Canada, October 31-November 4, 1992), ACM Press.
4. Johansen, R. User approaches to computer-supported teams, in *Technological Support for Work Group Collaboration*. (M.H. Olson, Ed., Hillsdale, NJ: Lawrence Erlbaum Associates), 1989, pp. 1-32.
5. Piersol, K. A Close-Up of OpenDoc, *BYTE Magazine*, March 1994.
6. Smith, D.C., Irby, C. Kimball, R. Verplank, B. & Harslem, E. "Designing the STAR User Interface", *Byte* 7(4), 1982, pp. 242-282. Reprinted in Baecker, R.M. & Buxton, W.A.S., "Readings in Human-Computer Interaction: A Multidisciplinary Approach", San Mateo, CA: Morgan Kaufmann), 1987, pp.653-661.

OpenDoc® is a registered trademark of Apple Computer, Inc., and is used by permission.