Duplex: A Distributed Collaborative Editing Environment in Large Scale*

François Pacull *Alain Sandoz* *André Schiper*

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)
E-mail: {pacull, sandoz, schiper}@lse.epfl.ch

ABSTRACT

DUPLEX is a distributed collaborative editor for users connected through the Internet. Large scale implies heterogeneity, unpredictable communication delays, and failures, and leads to inefficient implementations of techniques traditionally used for collaborative editing in local area networks. To cope with these unfavorable conditions, DUPLEX proposes a model based on splitting the document into independent parts, maintained individually and replicated by a kernel. Users act on document parts and interact with co-authors using a local environment providing a safe store and recovery mechanisms against failures or divergence with co-authors. Communication is reduced to a minimum, allowing disconnected operation. Atomicity, concurrency, and replica control are confined to a manageable small context.

KEYWORDS: Collaborative editing, distributed groupware, large scale networks, concurrency control

INTRODUCTION

The past ten years have seen the number of interconnected computers and networks increase in a considerable manner. As a consequence there is a growing interest in the design of distributed groupware, i.e. distributed computer applications which support cooperative work. Among these applications, collaborative editing is of great interest. It is concerned with many issues in groupware design as it addresses a fundamental necessity encountered by groups of workers: to formulate and communicate information inside or outside their community.

This paper presents the DUPLEX environment for collaborative editing, a groupware application designed to fulfill this need in a large scale distributed environment. DUPLEX furnishes a collaborative editing tool for groups of users distributed in distant locations, such as the co-authors of a scientific paper resident at different universities, or the partners of a multinational research project.

Collaborative editing involves both writing activities and communication between co-authors. A collaborative editing environment is composed of three separate facilities[18]:

- a *collaborative editor* enables to share and to maintain the state of a document written by several co-authors.
- a *direct communication facility* helps co-authors exchange information about the collaboration. Recipients of a message are selected by the sender.
- a *subjects based communication facility* enables to distribute data on specific subjects such as the modifications brought to part of the document. Recipients are the users possibly concerned by the subject.

These facilities furnish the means of interaction between co-authors. Each facility can be designed to provide either *synchronous* or *asynchronous* interaction [22]. With synchronous interaction, modifications to the document context (the body of all data such as text, figures, messages, directives, etc. produced using the three facilities) can be observed in real-time by all members of the collaboration. For example, several systems (GROVE[6], DistEdit[12], Mule[21]) propose a synchronous collaborative editor.

In contrast, the DUPLEX environment incorporates a collaborative editor with asynchronous interaction between users through a shared kernel which maintains the document context. Synchronous interaction, if required, is achieved using the two other basic facilities. For example, the initial document outlining phase and the final polishing phase require intensive direct communication rather than real-time editing[1].

*Research supported by the Swiss FNS and OFES under contract number 21-32210.91, as part of European ESPRIT Basic Research Project Number 6360 (BROADCAST).

[1] Such communication in DUPLEX does not require any kind of strong consistency.

The results of these phases are merged into the document in a second step using the collaborative editor. By restricting the collaborative editor to asynchronous interaction, a powerful environment can be designed for workers not affected by the inherent inefficiency of communication and distributed computations in large scale.

In asynchronous editors, sharing has been realized either through a central store (Quilt[7], MultimETH[16], Prep[19], Mule[21]), or through *local* replication of the data users are involved with (IRIS[2], CES[8]). DUPLEX proposes a hybrid approach. Sharing is based (1) on a replicated kernel which maintains shared information independently from users' locations and actions, and (2) on a set of policies for replica management and local caching designed to optimize concurrency control and response time.

Control of concurrent access to shared data is of course an important issue in collaborative editing. Three approaches have been proposed in the literature: (1) system enforced control based on an appropriate document model which defines the unit of concurrency[2, 8, 16]; (2) system enforced control based on the attribution of responsibility for document parts to users (roles [14]); and (3) relying on self-discipline and trusting users' perception of potentially conflicting behavior[6]. The DUPLEX approach integrates all three in a flexible manner. The document is decomposed into independently editable parts as in IRIS[2], CES[8], and MultimETH[16]. Decomposition is based on document structure, is dynamic, and reflects both document state and each author's current responsibility and involvement on different parts. The dynamic aspect enables adaptive concurrency control using a set of consistency policies tailored to collaborative editing. Authors are allowed to choose the type of control (exclusive, pessimistic, optimistic, etc.) they wish on the document parts they are concerned with.

The paper is structured as follows. We first present the basic DUPLEX model and the document decomposition. Then we present the structure and implementation of the kernel and finally the user's environment. A discussion concludes the paper.

DUPLEX MODEL

This section is an overview of the distributed collaborative editing environment. It presents the underlying distributed system, the basic DUPLEX model, and the implications of the basic model on fault-tolerance, concurrency control and integration in a large scale environment.

Distributed System Model

The distributed system is composed of a set of nodes in a wide area network. The set of nodes evolves dynamically as users join or leave the collaboration, and parts of a document are added or removed. We consider only the nodes involved with the edition of one document. Nodes for different documents can be disjoint or overlap, but are managed as independent distributed systems. A node is the workstation of an author, a data repository for parts of the document, or both. The failure semantics of nodes is crash-failure [4], i.e. all activities of a node, including communication, stop upon failure.

Communication between nodes is point-to-point, asynchronous (there are no bounded transmission delays) and unreliable. In particular, links can fail and the network can be partitioned into several mutually unreachable sub-networks. More precisely, the nodes are connected through the Internet, which provides the widest possible large scale integration in the current state of technology. This implies that two nodes might be unable to communicate, though both can communicate with a common destination.

Large scale means that nodes might be scattered over large distances, and in this case, are connected using several overlapping physical networks [3]. This implies high communication latency and prevents the efficient implementation of distributed algorithms between even small subsets of actively cooperating nodes. In the context of cooperative work, interactions between users working in different time zones suggest an asynchronous mode of operation.

Three major problems have an influence on high availability and functionality of the system: (1) managing node and communication failures; (2) controlling concurrency of operations on shared objects; and (3) integrating the system into a large scale environment.

Duplex Basic Model

The DUPLEX model incorporates three basic concepts.

The first concept is *document decomposition*, which specifies a set of rules to partition a document into *independent* parts. This reduces the granularity of concurrency between users and the effects of system failures which could lead to unavailability of document state. The document is partitioned into disjoint pieces according to authors' (self attributed or agreed upon) responsibilities in the collaborative edition, which reduces the probability that several authors work on the same data concurrently. Decomposition conforms to document structure, is dynamic and recursive, and enables to split, merge, and destroy document parts in a globally consistent manner (see Section Document Decomposition).

The second basic concept is the existence of a *kernel*, shared by all members of the collaboration, which provides persistence and availability of the most recent document state. The kernel is composed of *objects* (at any time one object for each existing independent document part). Kernel objects are replicated to ensure availability and quick access (see Section Kernel). Several operations are defined on a kernel object o_i (for example, read o_i, update o_i, or break o_i into smaller parts).

The third concept is the *local user environment* which provides a personal safe store and autonomous workspace for each author. Operations on kernel objects are invoked from within users' local environments, which are temporarily *connected* to the kernel object upon invocation. The operations themselves consist only of state transfer between the kernel object and the user environment, or the other way around. Once the operation completes, the user environment is disconnected from the kernel object.

A user environment contains a set of copies of kernel objects, which compose the user's local *view* of the document. These copies are local and not maintained as kernel object replicas. The user works locally on this view, disconnected from the kernel, until either (1) a copy of an object is required to continue work, and the user copies the object from the kernel, or (2) the user considers that an object of the local view can become shared, and updates the corresponding kernel object. Users work independently in their local environment and interact through the kernel.

This is illustrated in figures 1 and 2. User u_1 has a view of the document composed of a copy of objects o_1, o_2, o_4. The view is incomplete because there is no copy of o_3. User u_2 who wishes to work on object o_1, reads the object into his local view (fig.1).

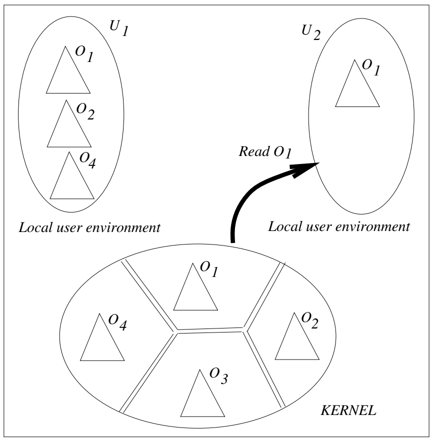


Figure 1: Kernel and user environments

In figure 2, user u_2 has modified object o_1 and broken it up locally into two local objects o_1 and $o_{1,1}$. User u_2 updates o_1 in the kernel, making the two objects shared. User u_1's view of object o_1 becomes obsolete.

As a consequence of the DUPLEX model, a user's view of the document can differ in several ways from the shared replicas maintained by the kernel. First, a local view can be incomplete in terms of the objects contained (like user u_1's). Second, a user can modify copies of objects or the document partition within the local view (like user u_2 after copying object o_1 and before writing it back). Finally, a user (u_2) can

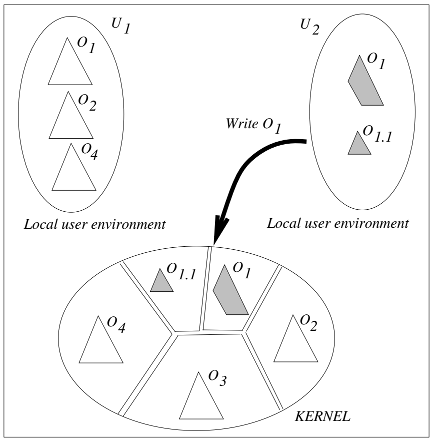


Figure 2: Updating the kernel from a user environment

modify the kernel state, making local views of other users (u_1) obsolete. Allowing the kernel state to diverge from local views is adapted to the asynchronous working interactions between users in a large scale environment where mutual consistency is impossible to achieve. A kernel object can be modified, unknown to a user which had been working previously on that part of the document. This user's previous views (stored and maintained locally) will enable to conduct adaptive recovery from within his/her environment in collaboration with the author of the modifications. The user's view is saved locally on disk either explicitly on the user's request, or periodically, and also when the user ends his/her session.

Managing the differences between local views and the kernel is realized using tools both at the kernel level (e.g. concurrency control, bulletin boards, journalization) and in the local environment (e.g. storing of views, visualization). These aspects are described in the related sections below.

Implications of the Basic Model

The concepts introduced so far respond to the three major problems described at the end of the Distributed System Model section.

Fault-tolerance. Thanks to the two kernel and user levels, failures in the distributed system can be neatly classified and their impact on the collaborative editor application confined. Kernel objects are maintained independently from user environments and replicated to increase fault tolerance. There are thus two types of nodes. *Kernel nodes*, which maintain the replica of at least one kernel object, and *user nodes*, which maintain the local environment of at least one author. A node can be both kernel and user, but the two functions are independent.

- *Failure of a kernel node.* The failure of a kernel node affects the availability of the kernel objects of which a replica is located. The kernel nodes of an object are maintained as a group [1, 15] and replication policies are implemented using group concepts. In particular, enforcing the atomicity of kernel operations and the replica set partition problem are treated within the group context [25, 26].

- *Communication failure between kernel and user node.* Two types of failures can occur: failure during connected operation and partition of a user node from some kernel objects currently disconnected from the user. The former are treated by enforcing atomicity of kernel operations. The second type of failure implies that, at some later time, a user might be unable to connect to some partitioned kernel objects. This prevents neither working on recently stored copies of those objects, nor accessing other reachable kernel objects.
- *Failure of a user node.* This occurs for example when an author's node crashes, local area network breaks down, or file server is unavailable. If the crash occurs during connected operation with the kernel, fault-tolerance implies enforcing atomicity of the operation. Else the effects are confined to the user's workspace. In particular, the crash of one user environment does not affect work of other members of the collaboration.

Concurrency control. The data consistency problem arises when several users make a request on the same kernel object concurrently. This situation is inherent to all distributed applications using shared data and raises a large number of issues [27], especially in the replicated context [23]. Solving the consistency problem requires (1) to define a consistency criterion for the concurrent application, (2) to adapt the criterion to the replicated context (i.e. define one-copy equivalent behavior), and finally, (3) to implement concurrency control and replica management policies accordingly. Depending on the application and the underlying system's failure characteristics, different types of policies can be considered (pessimistic vs. optimistic, or syntactic vs. adapted to application semantics [5, 13]).

In DUPLEX, the consistency problem is addressed by introducing document decomposition to reduce the occurrence of conflicting operations. A consistency criterion is defined such as to order operations on all kernel objects into a meaningful sequence. Finally, different concurrency control policies tailored to cooperative editing are proposed to the user, enabling him or her to use the type of control (pessimistic or optimistic) most appropriate to the situation at hand.

Large scale integration. Large scale integration poses two problems: (1) long communication delays for connected operation between distant nodes; and, (2) embedding the small document context into a large distributed system.

The first problem is curtailed by maintaining a rich working environment locally for each author, allowing him/her to work mostly in disconnected mode, reducing the need for communication and the duration of connected operations. Moreover, installing kernel objects on well chosen sets of nodes can speed up both communication and remote operations.

The second problem is more delicate. A name server is necessary to manage the set of kernel objects distributed over the large heterogeneous network. This set is small (typically a few dozen elements) and composed of objects whose names are contextual to the document. Replicas are however maintained in heterogeneous file systems with no global naming scheme consistent with the name space of the collaboration. This requires a dedicated name service designed to solve this problem in large scale distributed applications [17]. Combined with judicious document decomposition and kernel naming rules, the name service (see Section Kernel) enables to integrate the kernel in a large scale heterogeneous network environment.

DOCUMENT DECOMPOSITION

Document decomposition reduces the number of conflicts between operations on kernel objects, and hence the amount of control (expensive in a large scale system) needed to manage consistency. Informally, if the document is maintained as one object, then every pair of operations potentially conflict. For N operations, one has $N(N-1)/2$ conflicts to manage. Whereas, if the document is split in, say k parts ($k \ll N$), roughly equal in the sense that the probability of access to any one part by an operation is uniformly distributed among parts, then the number of conflicts drops to $N(N-k)/2k$. Document decomposition in DUPLEX establishes rules for splitting the document in order to realize these favorable conditions.

Authors themselves are probably the most knowledgeable on where conflicts might arise and whether they should, or on the contrary cannot, be avoided. So decomposition is designed both to enable consistent control of referencing, access, and concurrency by the kernel, and to allow user-driven, dynamic document partitioning.

The consistency problem arises from concurrent activities in the system (just as in a database maintained as a set of independent items), and *not* from distribution (the influence of distribution pertains to the implementation of atomicity). So one can consider the problem of document splitting as if the document were maintained at one centralized abstract location. In this respect, one can reason about the document at a given instant as if there existed a unique draft version which authors could handle as a whole (defining the consistency criterion in Section Kernel will clarify this notion).

Decomposition is based on the nested hierarchical structure of the document draft (e.g. a book is composed of chapters, composed of sections, sub-sections, paragraphs, figures, etc...). Consider a draft of the document from the hierarchical point of view: the draft is a *level-0* segment composed of several *level-1* segments (e.g. Introduction, DuplexModel, DocumentDecomposition, etc.). The draft can thus be modelized as a well formed bracketed expression[2]:

$\{_0\{_1\text{Introduction}\}\{_1\text{DuplexModel}\}$
$\{_1\text{DocumentDecomposition}\}\ldots$
$\{_1\text{Discussion}\}\}$

which partitions the complete document into disjoint segments of *level-1*. In turn, each *level-1* segment is itself a well formed sequence of *level-2* segments, and so on recursively. Finally, segments of the document without any structure (e.g. a sentence or an unnumbered paragraph) are called *elementary* segments.

In DUPLEX, any *level-k* segment ($k \geq 0$) can be defined as an *independent* document segment, and therefore maintained as a kernel object. Elementary segments can be independent, but cannot be further decomposed.

An example of decomposition of the present paper is presented in figure 3. Each box represents a segment. Independent sub-segments of *level-k* are represented in the corresponding columns. Dependent sub-segments are represented as boxes nested in their parent segment.

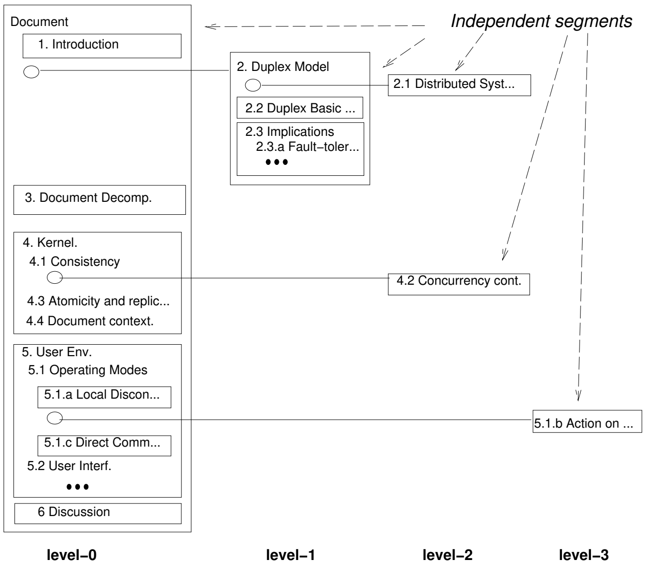


Figure 3: Example of decomposition

Consider at a given instant a partition of the document $\Pi = \{S_i\}$ along this type of structural boundaries. A complete description of independent segment S_i of level k is furnished by: (1) a complete description of sub-segments of S_i which are not independent; (2) the relative position in S_i of dependent and independent sub-segments; and (3) references to the independent sub-segments. The latter can be of level $> k+1$ as shown in figure 3 (e.g. for independent sub-segment 2.1.DistributedSystem at *level-2* relatively to sub-segment 2.DuplexModel at *level-1*).

Independent segment S_i is maintained as a kernel object. Any two operations on S_i represent potential concurrency and are thus subject to concurrency control enforced by this kernel object. Operations on S_i do not conflict with operations on independent sub-segments of S_i or on any other independent document segment, and are executed without concurrency control. Extracting an independent sub-segment from S_i, merging an independent sub-segment into S_i (thus making it dependent), and adding a dependent sub-segment anywhere in S_i are atomic operations which affect only S_i together with the considered sub-segment (and not any of its independent sub-segments).

KERNEL

With time, independent segments are extracted from the document, and consequently kernel objects are created in the network. The function of the kernel is (1) to maintain global consistency of the decomposed document, and (2) to ensure continued access to the document context. The first function requires the definition of a consistency criterion adapted to collaborative editing. The second requires replication of kernel objects. Both functions together require to adapt the consistency criterion to the replicated context, and to implement kernel operations accordingly and efficiently in the large scale system.

Consistency

The basic idea beneath the consistency criterion is the notion of draft version suggested in the preceding section. Suppose co-authors worked on a unique paper draft of the document passed along from one to another as requests to read, update or annotate the document arose. This mutually exclusive protocol would enable the document (considered as an object accessible concurrently) to always remain consistent, as all updates and annotations would be at hand when an author possessed the draft[3].

The ensuing consistency criterion, which relies on the existence of an ordering of all updates to document context, and of all reads, into a sequence consistent with *causal order* [24] in the system, is the one adopted in DUPLEX. If document state is distributed among a set of concurrent objects, this sequence is called a *linearization* [10] of the system. Linearizability is a criterion especially well suited to large systems, because it is a *local* criterion, meaning that if each object is linearizable individually, then the whole system is linearizable. Thus concurrency control need only be implemented on each object

[2] The opening bracket index represents the segment level in the document hierarchy

[3] This is not to say that the *contents* of the document would be consistent, since this is the responsibility of each author as part of a mutual contract and is independent of the editing environment and its concurrency control mechanisms!

independently to ensure global consistency of the document.

In [20] the linearizability criterion has been adapted to the replicated context. In the case of simple objects, like segments of text, adapting linearizability to replication for fault-tolerance is relatively simple and preserves the locality property, thus enabling efficient implementations of concurrency control.

Concurrency Control

Locality of the consistency criterion has a second useful property: the mechanisms used to enforce linearizability of different kernel objects can be chosen on a per-object basis, and moreover changed at any time.

Using this remark, DUPLEX offers for each kernel object a set of concurrency control policies (e.g. pessimistic based on capabilities, or optimistic based on a straightforward implementation of linearizable replicated registers), from which the user can chose according to the degree of concurrency he/she is ready to allow on a document segment he/she is currently involved with. Exclusive access offers the best guarantee that the latest updated view of an object is actually the one maintained by the kernel. However, the less expensive optimistic policies are more advantageous if decomposition has been used properly to isolate a segment of interest. Moreover, in case of a conflicting update overwriting an object, the object can still be recovered from within the local environment of the user having made the lost update (see Section User Environment). Finally agreement between authors in conflict over some independent segment can be reached using the direct or subject based communication facilities.

Atomicity and replication

Atomicity of operations on non-replicated objects is inherent to the synchronous mode of invocation in linearizable systems, and well suited to the state transfer based DUPLEX operations. However, in the replicated context, atomicity requires reliable multicast communication. In DUPLEX this is realized by managing the replica set as a *group* of processes in the virtually synchronous model [1, 26]. Localization of group members (replicas) is realized using a dedicated light weight name server for the document. The name server links logical document segments (known to authors by a common shared name established upon creating the independent segment) to the set of replica processes running in the distributed system.

Document context

A final point is to describe what information on document context a kernel replica maintains. This information is composed of four separate items:

- **Textual information** provides a complete description of the document segment maintained by the kernel object (e.g. a sequence of characters, or the bitmap of a figure).
- **Structural information** describes the decomposition of the segment into dependent and independent sub-segments, and maintains logical references (by name) on independent sub-segments. The references are translated into addresses in the distributed system by the light weight name server, upon invocation of a kernel operation on the segment.
- **A bulletin board** supports the subject-based communication facility for the segment. This is a list of messages which can be either of limited validity in time or persistent. The list is causally ordered and consistent with the interleaving of discussions and segment updates.
- **A journal** contains information describing all the sensitive operations (e.g. update, split, delete, etc.) performed on the segment during its active lifetime. The journal is persistent until the end of the collaboration, i.e. when the document is finished, and can be consulted to help recovery by a user using views stored in the local environment.

All operations on a kernel replica atomically modify or read one or several of these items.

USER ENVIRONMENT

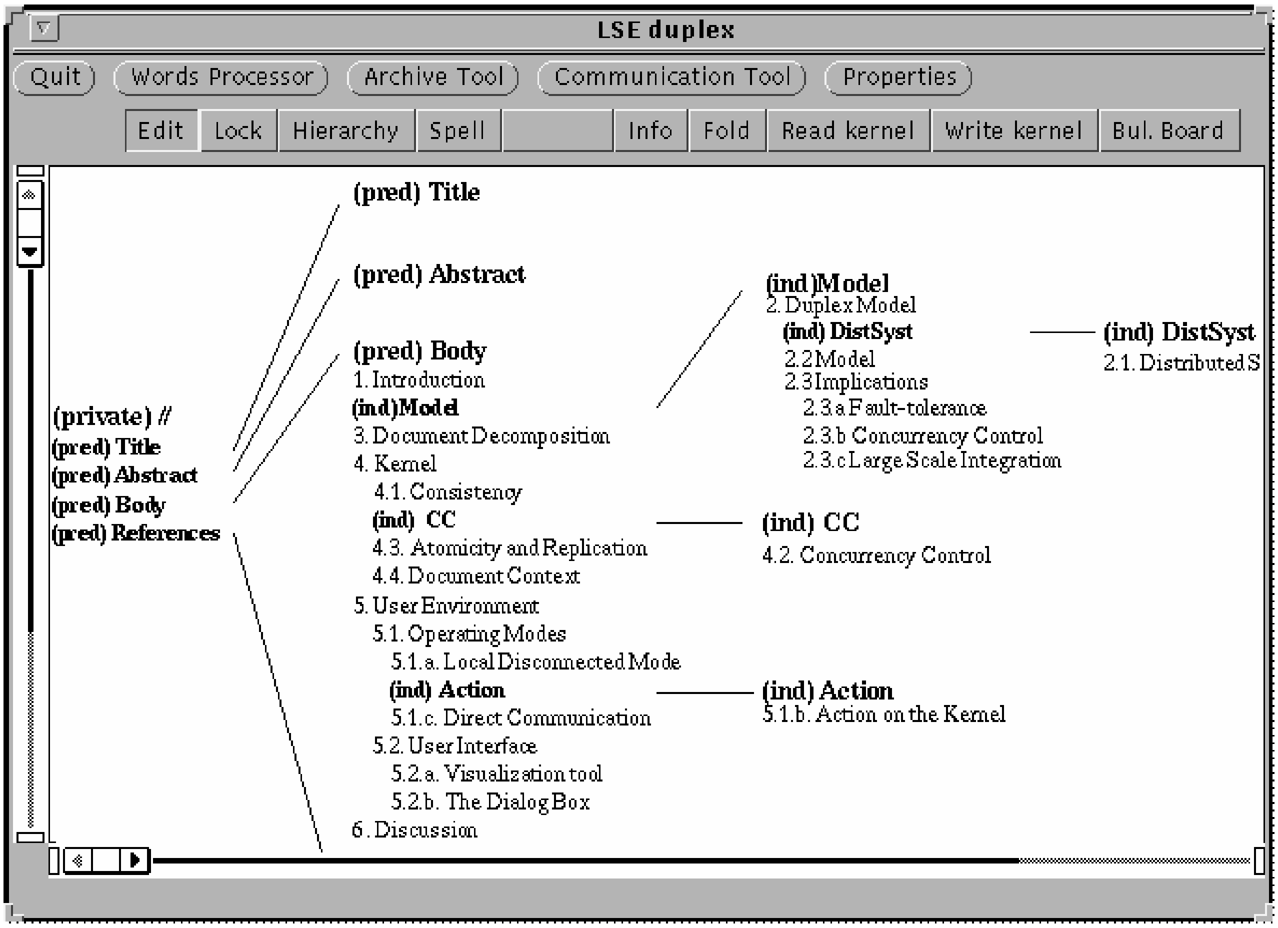
The basic function of the local user environment in DUPLEX is to enable the user to perform *local disconnected work* every time it is possible. Nevertheless "collaborative editing" requires interaction between users in order to accomplish the common task. This can be performed either as *actions on the kernel* or as *direct communication between users*.

Operating Modes

Local Disconnected Mode. All activities that do not require an action on kernel objects or interaction with co-authors is done in local disconnected mode. It is by far the most common mode of operation, since all text editing and previewing is done in this mode. Since by definition disconnected mode does not require a network connection, a user can work in the local environment though it is physically disconnected from the rest of the world, e.g. because of a failure or because the user is working on a laptop during travel (what is commonly referred to as *disconnected operation* [9, 11]). Two types of operations are performed within this mode.

The first type is editing, performed on local copies using the author's preferred text or graphic editor (DUPLEX supports all editors commonly used with UNIX, e.g. emacs, textedit, idraw, etc.).

The second type concerns archiving and retrieving document context in the user environment. Archiving consists in storing the successive views of the document an author constructs by reading kernel objects and making modifications to contents and structure of the document. It enables the user to keep a



Round buttons define tools: Archiving, Communication facilities, etc...

Square buttons define operating mode: Edit, Lock, Info, transfert from/to kernel, Bulletin board, etc ...

Figure 4: User interface

causal trace of his/her work even if it has never been transmitted to the kernel or later overwritten by a co-author. The main difference from classical approaches is that archiving is performed in the user environment and not by the shared kernel. This responds to the user's interest in retrieving information he/she knows about (i.e. has already seen) and decreases dependency on the kernel.

Action on the kernel. In connected mode, the user interacts with the kernel, or more precisely with a kernel object. There are three types of interaction with the kernel: (1) *object access* concerns state transfer between kernel object and user environment, and modifications to segment structure; (2) *journal lookup* enables the user to acknowledge important operations performed on the object by other users, such as decompose, update or delete; finally (3) *bulletin board consultation* enables the user to generate, filter, and read messages related to the segment maintained by the kernel object.

Direct Communication. Using communication with the bulletin board, the recipient of a message cannot be defined by the sender. Another facility is required to provide a direct (and private) communication channel. The direct communication facilities (currently based on e-mail within the user environment), provides both synchronous and asynchronous communication between users or groups of users.

User Interface

Visualization tool. The main component of the user interface is the visualization of the user's current local view of the document. This visualization represents the document's structure into dependent and independent segments, as can be inferred from all local information present in the user environment. In particular, the decomposition rules enable to infer a consistent numbering of segments in the local view. The document structure is presented as a tree whose nodes are the segments. From this representation, the user can apply different customary tools (editor, word processor, speller, etc.) and the DUPLEX tools (archiving, kernel operation, or communication) to selected segments, using the mouse. The user interface is reproduced in figure 4. In particular, edges in the representation are links to independent objects.

The dialog box. Interaction with the kernel, when requested on a segment selected from the document representation, is realized through a dialog box. The dialog box is the specialized interaction means between the kernel and the user workspace, during the working session, and for this particular object. Once a connection has been established, interaction can be initiated by the user (e.g. to transfer state to and from the kernel) or by the kernel (e.g. to notify the user of new messages on the segment's bulletin board). Communication between a kernel object and the dialog box is realized using unreliable datagrams to prevent long connected sessions.

DISCUSSION

The objective of DUPLEX is to provide a collaborative editing environment on a large network. Large scale implies heterogeneity, long communication delays, link failures, and node crashes. It prevents using traditional techniques for collaborative editing which would work in a local area network.

Our approach explores two directions to make the system scalable. The first concerns cooperative work paradigms. DUPLEX uses document decomposition together with a consistency criterion adapted to collaborative editing in order to limit conflicts between authors and enable recovery using subject based or direct communication. Decomposition enables users to clearly define the segments of the document they are most interested in and to apply the concurrency control they wish on these segments. Thus concurrency control is *adaptive*, in the sense that it remains relatively loose during the phase of document edition where users most work independently, but later becomes more restrictive when independent segments are merged and concurrency control is applied to larger pieces of the document. This first direction emphasizes liberty of action and choice of concurrency control mechanisms as a means to help the users progress on the collaborative edition. It has been put to trial while writing this paper and has proven to be satisfactory, though this should not be surprising since it reflects our own perception of cooperative work.

The second direction is more system oriented and concerns working in large scale and heterogeneous environments. Regarding the latter point, DUPLEX is a set of library routines running on top of UNIX and using X11 and TCP/IP. It is independent of any particular hardware. Concerning large scale, making document segments independent in the distributed system provides much flexibility and enables to implement the kernel/local environment concept in a scalable manner. This should make the system both reliable and efficient in large networks. Our next planned development is to test this assertion with several partners of European projects we are involved in.

REFERENCES

1. K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37-53, December 1993.
2. U.M. Borghoff and G. Tegge. Application of collaborative editing to software-engineering projects. *ACM SIGSOFT*, 18(3):56–64, July 1993.
3. D.E. Comer. *Internetworking With TCP/IP: Principles, Protocols, Architecture*. Prentice Hall, Stevenage, 1988.
4. F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.
5. S.B. Davidson, H. GarciaMolina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.
6. C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware - some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.
7. R.S. Fish and M.D.P. Leland R.E. Kraut. Quilt: A collaborative tool for cooperative writing. In *Proceedings of ACM Int. Conf. on Office Information Systems*, volume 9, pages 30–37, March 1988.
8. I. Greif, R. Seliger, and W. Weihl. A case study of CES: A distributed collaborative editing system implemented with Argus. *IEEE Transactions on Software Engineering*, 18(9):827–839, September 1992.
9. J.S. Heidemann, T.W. Page, R.G. Guy, and G.J. Popek. Primarily disconnected operation: Experiences with Ficus. In *Proceedings of the 2nd Workshop on the Management of Replicated Data*, December 1992.
10. M. Herlihy and J. Wing. Linearizability a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12:463–492, July 1990.
11. J.J. Kistler and M. Sartanarayanan. Disconnected operation in the coda file system. *ACM SIGOPS*, 25(5):13–16, October 1991.
12. M.J. Knister and A. Prakash. DistEdit: A distributed toolkit for supporting multiple group editors. In *Proceedings of the ACM Int. Conf. on Computer-Supported Cooperative Work (CSCW '90)*, pages 343–355, October 1990.
13. R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. *Operating Systems Review*, 25(1):49–55, January 1991.
14. M.D.P. Leland, R.S. Fish, and R.E. Kraut. Collaborative document production using Quilt. In *Proceedings of the ACM Int. Conference on Computer-Supported Cooperative Work (CSCW '88)*, pages 206–215, September 1988.
15. L. Liang, S.T Chanson, and G.W. Neufeld. Process groups and group communications: Classifications and requirements. *IEEE Computer*, 23(2):56–65, February 1990.

16. H. Lubich and B. Plattner. A proposed model and functionality definition for a collaborative editing and conferencing system. In *Proceedings of IFIP WG 8.4 Conf. on Multi-User Interfaces and Applications*, pages 215–232. North-Holland, September 1990.
17. J.-C. Lugeon and A. Sandoz. Sharing a small domain in a large distributed file system. Technical report, Swiss Institute of Technology of Lausanne (anonymous FTP ftp-lse.epfl.ch:/pub/TechReports/DILSE-6-93.ps), 1993.
18. V.C. Miles, J.C. McCarthy, A.J. Dix, M.D. Harrison, and A.F. Monk. Computer Supported Cooperative Writing (chapter 8: Reviewing designs for a synchronous-asynchronous group editing environment). In *Computer Supported Cooperative Work*, pages 137–160. Springer-Verlag, 1993.
19. C.M. Neuwirth, D.S. Kaufer, R. Chandhok, and J. Morris. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the ACM Int. Conf. on Computer-Supported Cooperative Work (CSCW '90)*, 1990.
20. F. Pacull and A. Sandoz. R-linearizability: An extension of linearizability to replicated objects. In *Proceedings of the 4th IEEE Workshop of Future Trends of Computing Systems*, 1993.
21. M.O. Pendergast and D. Vogel. Design and implementation of a PC/LAN-based multi-user text editor. In *Proceedings of IFIP WG 8.4 Conf. on Multi-User Interfaces and Applications*, pages 195–206. North-Holland, September 1990.
22. I.R. Posner and R.M. Baeker. How people write together. In *Proceedings of the 25 th Hawaii International Conference on System Sciences*, volume IV, pages 127–138, January 1992.
23. M. Raynal and M. Mizuno. How to find his way in the jungle of consistency criteria for distributed shared memories. In *Proceedings of the 4th IEEE Workshop on Future Trends of Computing Systems*, 1993.
24. A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proceedings of the 3rd Workshop on Distributed Algorithms*, 1989.
25. A. Schiper and A. Ricciardi. Virtually synchronous communication based on a weak failure suspector. In *Proceedings of the 23rd IEEE Int. Conf. on Fault Tolerant Computing Systems*, 1993.
26. A. Schiper and A. Sandoz. Uniform reliable multicast in a virtuallly synchronous environment. In *Proceedings of 13th IEEE Int. Conference on Distributed Computing Systems*, May 1993.
27. A. Sheth and M. Rusienkiewicz. Management of interdependent data: specifying dependency. *Proceedings of the 1st Workshop on the Management Of Replicated Data*, November 1990.