2

CoCoDoc: a framework for collaborative compound document editing based on OpenDoc and CORBA

G.H. ter Hofte, H.J. van der Lugt
Telematics Research Centre
P.O. Box 589, 7500 AN, Enschede, the Netherlands
Phone: +31-53-4850485, Fax: +31-53-4850400
E-mail: {H.terHofte, H.vanderLugt}@trc.nl

Abstract

We propose collaborative compound document editing as a new paradigm for editing environments and describe the design and implementation of CoCoDoc, a framework based on OpenDoc and CORBA. CoCoDoc supports reuse of existing editors as simple collaborative editors and supports development of new collaborative compound part editors with flexible collaboration facilities, thus facilitating a gradual migration towards collaborative editing environments that are both rich in editing support and rich in collaboration support.

Keywords

CSCW, collaborative editing, component software, compound documents, OpenDoc, CORBA, couplable object groups, multicast ORB

1 INTRODUCTION

Until recently, conventional editors and word processors primarily assisted an individual in producing polished final copy and provided little support for collaborative editing. Yet, many documents result from a collaborative effort. In some fields of science, for example, 65% of articles are written by two or more authors (Fish et al., 1988).

Increased penetration of networks enlarged the potential for richer computer support of collaborative work in general (CSCW) and for collaboratively editing a shared document in particular. However, developing collaborative editors is a complex and challenging task: developers are faced with many distributed systems issues, such as distribution, replication, consistency, concurrency and communication protocols.

J. Rolia et al. (eds.), *Open Distributed Processing and Distributed Platforms*

Most developers of collaborative editors either focused on editing functions (i.e. provide rich support for media types and editing operations on media) or focused on collaboration functions (i.e. provide rich support for coupling of operations and control over coupling). In this paper, we propose 'collaborative compound document editing' as a new paradigm that facilitates the development of collaborative editing environments that are both rich in editing support and rich in collaboration support. The paradigm is inspired by a specific combination of emerging distributed computing platforms and compound document editing frameworks, viz. CORBA (Orfali et al., 1996) and OpenDoc (Nelson, 1995; MacBride et al., 1996). In particular, we present CoCoDoc, a framework that supports developers in constructing collaborative compound document editing systems. Work on CoCoDoc was part of the Platinum project, a joint research project on multimedia CSCW applications over broadband networks (Ouibrahim et al., 1995), in which Lucent Technologies, the Telematics Research Centre, the University of Twente and Deutsche Telekom participated. CoCoDoc is part of a larger platform for collaborative multimedia applications that was developed in the Platinum project. The platform is based on Co^4, a generic model for groupware functionality (Ter Hofte et al., 1996b), which distinguishes four broad areas of groupware functionality:

- *conference management*: creating, terminating, and modifying conferences: associations between users, shared workspaces, conversation channels and coordination policies;
- *cooperation support* in shared workspaces: joint manipulation of shared artifacts;
- *conversation support*: conversation channels for direct communication between users, such as audio, video and textual chat channels;
- *coordination support*: mechanisms and policies, e.g., floor control policies and workflows.

CoCoDoc provides the platform's support for shared workspaces.

The body of this paper is structured as follows. First, we introduce collaborative editing and compound document editing. Then, we describe the advantages of combining these into collaborative compound document editing. Subsequently, we describe the design and implementation of the CoCoDoc framework. Finally, we describe how CoCoDoc supports the implementation of a simple collaborative compound outline editor.

2 COLLABORATIVE EDITING

A collaborative editing system can be modeled as a distributed computer system with multiple user interfaces: one for each user. Due to different actions by different users, the user interfaces *diverge*, and sooner or later, the collaborative editing system must *synchronize* the user interfaces so a user interface not only reflects a user's own actions (i.e. give feedback), but also the actions of other users (i.e. give feedthrough) (Dourish, 1995). Major functions of collaborative editors are synchronizing the user interfaces and thus keeping the representations of the document 'consistent' to some degree, providing awareness of other user's actions, as well as coordinating actions when it is desirable that not all users can perform all actions at any moment (e.g. avoid unwanted conflicts when two users want to edit the same document(part)). Other functions of collaborative editors, such as conference management and conversation support, are not discussed in this paper.

From analysis (Ter Hofte, 1996) of existing collaborative editing prototypes and commercial products (Koch, 1994), three typical architectures for collaborative editing systems emerged:

- the *shared file system* software architecture;
- the *collaboration-aware* software architecture;
- the *shared user interface* software architecture.

In each architecture, a different part of a typical single-user software architecture (consisting of a file system, an application and a user interface) is made collaboration-aware. The *shared file system* and *shared user interface* approach both employ collaboration-unaware editors (i.e. editors that were not designed to be used by multiple collaborating users) and combine them with application-independent collaboration services. This approach allows for existing editors to be used in collaborative settings.

2.1 Shared file system architecture

In the *shared file system* architecture, an unmodified single-user editor application (e.g., MS Word) is combined with a shared file system (e.g. NFS, NetWare), as illustrated in Figure 1.

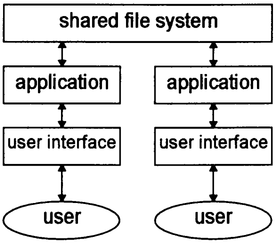


Figure 1 Shared file system software architecture.

As soon as one user saves the document, the shared file gets updated and other users can observe the updated document. *Exclusive shared file editing* is a form of coordination that allows one user at a time to get the exclusive permission (the write lock) to update the document, in order to avoid conflicting updates or the emergence of different versions of the document. *Versioned shared file editing* allows multiple versions to emerge and provides rudimentary version navigation and/or version merging services.

2.2 Shared user interface architecture

In the *shared user interface* architecture, an unmodified single-user editor application (e.g., Xemacs) is combined with a shared user interface system (e.g., SharedX), as illustrated in Figure 2. In fact, this architecture allows sharing the user interface of any application, not only editors.

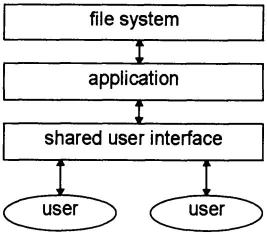


Figure 2 Shared user interface software architecture.

With a shared user interface, as soon as one user performs an action, the user interface state gets updated and all users can observe the updated user interface. This property is also known as *WYSIWIS (What You See Is What I See)* (Stefik et al., 1987). Examples of shared user interface systems are screen sharing systems such as NLS (Engelbart, 1975) and shared window systems (e.g. X windows-based systems (Baldeschwieler et al., 1993).

Floor-controlled shared user interface editing is a form of coordination over shared user interface editing that allows only one user at a time to get an exclusive permission to perform actions (that user holds the 'floor') and that provides mechanisms to pass the floor.

2.3 Collaboration-aware editors

A collaboration-aware editor, i.e. an editor that is designed for concurrent use by multiple collaborating users, can offer more forms of collaboration support. For example, a collaboration-aware text editor may allow simultaneous access to a document, e.g. different users may simultaneously edit different sentences in the same document, or different users may simultaneously edit the same sentence. Additionally, collaboration-aware editors can provide users with awareness of who is editing what, who has modified what, at what time, etc. Figure 3 illustrates a collaboration-aware software architecture.

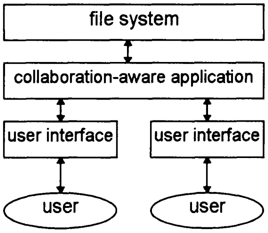


Figure 3 A collaboration-aware software architecture.

However, development of such applications is complex. In fact, researchers approached the complexity of the problem from either the shared file system approach or from the shared user interface approach.

- The academic collaborative editing tools Quilt (Fish et al, 1988; Leland, 1988) and PREP (Neuwirth et al.,1990) are typical examples of early collaboration-aware editors that exploit shared file systems. They offer support for annotations, for communicating comments and intentions, and revisions, as well as role-based access. They can be regarded as 'early' collaborative editing tools and use standard shared file systems or shared databases, they notify users only of committed changes made by other users, while carefully avoiding the complexities involved in providing simultaneous access to documents and records. Broadly speaking, such tools are generally referred to as '*asynchronous*' collaborative editors.
- Another approach was taken by GROVE (Ellis et al., 1990) and DistEdit (Knister et al., 1990). These are examples of collaborative editing tools that support sessions of simultaneous access to a shared document as well as 'immediate' awareness of interactions of other users with the shared document. Broadly speaking, these tools are referred to as '*synchronous*' collaborative editors.
- Only recently, 'multi-synchronous' collaborative editing tools have emerged that support combinations of synchronous and asynchronous collaborative editing and transitions between these modes (e.g., SEPIA (Haake et al., 1992)).

2.4 Coupling levels: the zipper model

The above architectures are variants of the typical single-user software architecture (consisting of three levels: file system, application and user interface). Other approaches show that additional levels are possible, e.g. by separating the application level into an edit level and a view level, as applied in GROVE and DistEdit. If state at a level is coupled, then state updates (caused by operations) at that level are observable for all users. If state at a level is uncoupled, then state updates at that level are unrelated, i.e. each user has its own state. Inspired by this possibility to couple and uncouple more or less levels, (Patterson, 1995) coined this the collaborative 'zipper' architecture, as illustrated in Figure 8.

An example of levels and associated operations is presented in Table 1, which classifies user operations of a collaborative outline editor in four levels, ranging from those operations only affecting the user interface to operations affecting the persistently stored document.

Table 1 A four-level zipper: state and operations per level

Coupling Level	*State*	*Operations affecting state*
file coupling	stored state of document	`Externalize`[*]

edit coupling	editing state of document	`InsertUnit`, `DeleteUnit`
		`EditUnit`, `MoveUnit`, `RenameUnit`

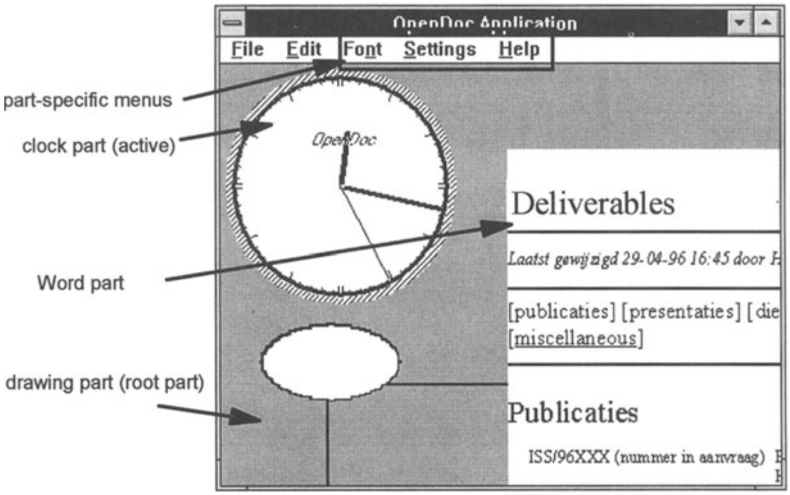
view coupling	view control state (e.g., expand/collapse state)	`CollapseUnit`, `ExpandUnit`
user interface coupling	user interface control state (e.g., mouse position, scrollbar state)	Scroll content, Move mouse pointer Push button

[*] We use monospaced text to indicate method names and class names.

Later, collaborative zipper architectures were introduced with even more levels (Dewan, 1996; Karsenty et al., 1995).

3 COMPOUND DOCUMENT EDITING

In compound document editing approaches such as OpenDoc and OLE2, a *compound document* consists of a hierarchy of *parts*** that may embed other parts. Each part has *intrinsic content* of a particular *part kind* (e.g. an abstract data type, such as text, audio, video, spreadsheet data, or any other editable abstract data type). In addition to having intrinsic content, a part may embed other parts. The part at the top of a compound document hierarchy is called the *root part*. A *frame* maintains the relation between an embedding part ('parent') and an embedded part ('child'). A frame becomes visible as space that the (presentation of the) embedded part has in the (presentation of the) embedding part. That is, an embedding part contains a frame, and the frame contains the embedded part. Each part is handled (i.e. displayed, edited, stored, printed) by its own *part handler*, a sort of mini-application. When a user embeds a picture into a text document, it is the picture *part handler* that gets 'embedded' into the text part handler. That is, once the picture is embedded in the document, the picture part handler still is responsible for handling (i.e. displaying, editing, storing, printing) the embedded picture. Another crucial user interface characteristic of compound document editing is *in-place editing*: embedded parts can be completely handled at the location in which they are inserted in the document, not in a separate window. Figure 4 illustrates a typical user interface of a compound document editor.



Deliverables

Laatst gewijzigd 29-04-96 16:45 door F

[publicaties] [presentaties] [die
[miscellaneous]

Publicaties

ISS/96XXX (nummer in aanvraag)

Figure 4 A compound document editing screen.

** We will use OpenDoc terminology here, given our choice to use OpenDoc as a basis for implementation, as explained later.

Thus, in a compound document editing session, different part handlers cooperate to create the illusion of one seamless compound document being edited by one seamlessly integrated application. This requires a single, well-defined inter-part handler interface, across which part handlers can negotiate about screen space, share menu bars, share file storage, etc.

In the traditional editing approach, the inter-application interface is of a very different nature, viz. exporting, importing and conversion of data between applications. Each data type has its own application and there is typically one application that edits the entire final document. This application takes care of displaying, printing, storing and (limited) editing of embedded (or 'contained') data, that is usually cut, converted and pasted into the application. A problem with this approach is the numerous conversion filters that are to be included in applications, resulting in increasingly large applications. Moreover, updates in contained data require dynamic update mechanisms, such as DDE.

The benefits of the compound document approach, compared to the traditional editing approach are clear: less need for conversion filters, no loss of fidelity (each abstract data type is edited by its native application) and *polymorphic embedding*: once a part handler can embed another part handler, it can embed *all* part handlers, including future part handlers.

4 COLLABORATIVE COMPOUND DOCUMENT EDITING

Although both the benefits of collaborative editing and the benefits of compound document editing are clear, the combination of the two editing paradigms has hardly been studied up to now. To our knowledge only Taligent (Orfali et al, 1996, p. 307]) and the Technical University of Munich (Koch, 1995; Schlichte, 1996) have done work in this area.

A collaborative compound document editing environment consists of a collection of different collaborative part editors, each with its own media editing capabilities and collaborative editing capabilities. Within the same editing environment, collaboration-unaware and collaboration-aware part editors can peacefully coexist. Even in the context of a single document, both collaboration-unaware part editors and collaboration-aware part editors could be active editing their parts. Collaboratively editing a document with MS-Word in file coupling mode (see Table 1), with an embedded picture that is collaboratively edited in view coupling mode (see Table 1), is perfectly conceivable and technically feasible. Each part of a document has its own zipper, e.g. a rather inflexible zipper with two coupling modes as indicated in Figure 1 and Figure 2, or a more flexible zipper with more coupling modes similar to Figure 8.

As a jump-start for constructing a collaborative compound document editing environment, many existing and collaboration-unaware single user compound document editors (e.g. OLE2 based applications) can be used as a collaborative part editor. When such part editors are combined with facilities such as a shared user interface system (see Figure 2) or a shared file system (see Figure 1), these part editors can provide support for simple forms of collaborative editing such as WYSIWIS editing and shared file editing. The environment can evolve over time, by adding newly developed or upgraded versions of collaborative part editors, thus extending the environment with new media editing capabilities and/or collaborative editing capabilities. Figure 5 illustrates these two dimensions.

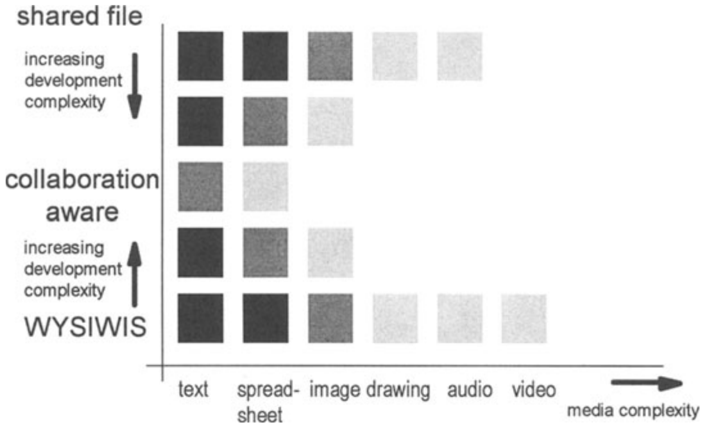


Figure 5 Migration of a collaborative compound document editing environment: adding new collaborative part editors (indicated by lighter squares) with new editing facilities (i.e. part kinds) and/or new collaboration facilities.

For developers, the collaborative compound document editing paradigm offers the following benefits:

- *Reduced development effort*: Developers can concentrate on providing editing functionality in small, manageable chunks, without having to provide an entire editing environment to have a marketable product. For example, developers could concentrate on a collaborative editor of a specific part kind in which they happen to be good (e.g. a spreadsheet editor), leaving editing of other parts in a user's editing environment to others.
- *Decide on collaborative features on a part-by-part basis*: The needs for collaborative features often depend on the part kind. Decisions on collaboration-awareness or collaboration-unawareness can be taken on a part-by-part basis.

For a group of collaborating users, the collaborative compound document editing paradigm offers the following benefits:

- *Coupling setting per part*: With more advanced collaborative part handlers, users could decide to collaborate more tightly over some parts (e.g. view coupling in a drawing), and to collaborate more loosely over other parts (e.g. file coupling for other parts).
- *Less steep learning curve*: Collaborative editing requires learning to use new features related to collaboration. A collaborative compound document editing environment can start with familiar editors. When collaborative features are incrementally added, learning can also be done incrementally.
- *Increased composability and extensibility*: The collaborative compound document editing environment can be tailored to the group's needs, by selecting only a subset of available part editors.
- *Reduced vendor lock-in by use of open standards*: Combined use of different part editors from different vendors within a collaborative compound document editing environment is

possible if they conform to an open standard for inter-part editor interaction, such as the Distributed Document Component Facility (DDCF) standard of OMG (Apple Computer et al., 1995). Moreover, open standards for part kinds (persistent data formats) would allow for the emergence of a software components market, from which users can choose for the most suitable part editor on the market for a particular part kind. Finally, open standards for intra-part editor collaboration protocols (i.e. between the local peer part editors) would allow each user in a group of collaborating users to choose a different local peer editor implementation, thus further reducing vendor lock-in.

5 THE COCODOC FRAMEWORK

CoCoDoc is a framework that supports the development of collaborative compound document editing systems. It has been shaped considerably by our decision to design and implement it as a collaborative extension of an existing compound document framework, rather than developing it from scratch, or as a compound document editing extension of a collaborative document editing framework or toolkit (such as DistEdit (Knister et al., 1990), GroupIE (Rüdebusch, 1995) or IRIS (Koch, 1995)). We considered both alternatives to be inefficient, as the development of compound document editing document facilities requires a significant amount of time without providing new insights compared to using existing compound document frameworks. We also rejected the alternative of extending Taligent's implementation of a rudimentary collaborative compound document editing framework, because it was not easily available for R&D purposes and because it was not sufficiently in line with emerging compound document editing standards.

In sections 5.1 and 5.2, we present the support CoCoDoc provides for developers of collaborative component editors. In section 5.1, we focus on compound document editing support and in section 5.2, we focus on collaborative document editing support. Subsequently, in section 5.3, we briefly describe the current implementation of CoCoDoc.

5.1 Compound document editing support

A compound document editing system consists of a number of part editors, each devoted to editing a particular document part (e.g., a text part or a spreadsheet part), in close cooperation with other part editors, as illustrated in Figure 6.

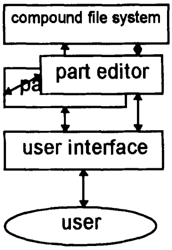


Figure 6 Non-distributed view: inter-part relations.

The *compound* document editing support that CoCoDoc needs to provide is primarily concerned with inter-part relations, i.e. relations between different part editors involved in editing a document.

- *Embedding relations between parts.* The CoCoDoc platform needs to support embedding relations between parts (and consequently: between part editors); this involves creation, and deletion of frames and negotiation about screen space.
- *Compound document storage.* Parts need to be supported in sharing document storage with each other (e.g. sharing a file that stores a compound document).
- *Compound document user interface*: Parts need to be supported in sharing a document editing user interface with each other. This involves part activation and dispatching input to parts, and sharing user interface objects such as menus (e.g. making sure that the menu bar always is the menu bar of the active part).

As a basis for the design and implementation of compound document editing support in CoCoDoc, we decided to use OpenDoc (Nelson, 1995; Orfali et al., 1996), a new industry standard for compound document editing, which was recently adopted as the basis for the OMG standard for the CORBA Distributed Document Component Facility (DDCF) (Apple Computer et al., 1995). In OpenDoc, part developers must implement a subclass of the OpenDoc pure virtual baseclass `ODPart`, by overriding a minimum of 6 and a maximum of 60 methods defined in the interface of `ODPart` (Orfali et al., 1996, p. 346-350). These methods are called by the OpenDoc framework and cover areas such as embedding, user interface event handling, initialization and termination (constructors/destructors), frames, and imaging and linking (Orfali et al., 1996).

5.2 Collaborative document editing support

A collaborative part editor typically consists of a number of peer part editors, each devoted to supporting an individual user, in close cooperation with its peer part editors, as illustrated in Figure 7.

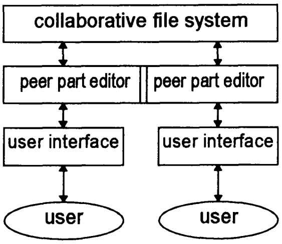


Figure 7 Distributed view: intra-part relations.

The collaborative document editing support that CoCoDoc needs to provide is primarily concerned with intra-part relations, i.e. relations between the different peer part editors:

- *Collaborative part life cycle support.* When a collaborative part editor is instantiated, multiple peer part editors need to be instantiated (one for each user) and relations between these peers need to be maintained. Also, when a new users joins or leaves, a new peer part editor for that user needs to be instantiated or deleted, respectively.

- *Shared file editing support for parts* (similar to Figure 1). The peer part editors must share storage for the part they are editing. In addition to 'traditional' shared file editing, the support may provide collaborative storage services such as change notifications, versioning and version merging.
- *Shared user interface editing (WYSIWIS) support for parts* (similar to Figure 2).
- *Support for multiple levels of coupling in a part, and part coupling control* (similar to Figure 8).
- *Document-wide coupling control.* It may not always be desirable that each part editor is coupled at its own level. Users must have the ability to set the coupling level for the entire document. In CoCoDoc, four document-wide coupling levels are identified, which must be supported by all collaboration-aware CoCoDoc-compliant part editors: file coupling, edit coupling, view coupling and user interface coupling. An example of these levels was given in Table 1.
- *Document-wide outline control.* The CoCoDoc framework should allow each part in an embedding hierarchy to contain one or more outline levels and to be able to present an outline view. The outline level should be controlled document-wide.

Collaborative part editor development in CoCoDoc is very similar to part editor development in OpenDoc. In CoCoDoc, part developers must implement a subclass of the CoCoDoc pure virtual baseclass **CoCoDocPart**, which is described below, in the section entitled 'CoCoDocPart baseclass'. Support for multiple levels of coupling in a part, and part coupling control is provided by *couplable object groups*, as described below, in the section entitled 'Couplable object groups and the Multicast Object Request Broker'. With these, CoCoDoc covers most of the collaborative requirements as listed above, with the exception of versioning, version merging and shared user interface editing (WYSIWIS) for parts, which are not yet supported in the present version of CoCoDoc.

CoCoDocPart baseclass

The methods that **CoCoDocPart** subclass developers must override are very similar to those of **ODPart**; except for a few methods which have been replaced by **CCD_** variants and a few additional **CCD_** methods which must be implemented, in the areas of (I) collaborative part life cycle support, (II) shared file editing support for parts, (III) document-wide coupling control and (IV) document-wide outline control, as indicated in Table 2. For the implementation of these methods, the CoCoDoc framework provides developers with some additional **CoCoDocPart** methods that are prefixed by '**CoCoDoc_**'.

Table 2 Overview of `CoCoDocPart` methods

	CoCoDocPart method that needs to be implemented	*When called; support provided by CoCoDoc*

I	`CCD_InitPart,`
	`CCD_InitPartFromStorage`
		called at part creation: makes sure storage units are coupled; framework takes care of interaction with conference management facilities

	`CCD_createObjectGroupMember`	called when the part needs to act as factory for an object group member
II	`CCD_storageUnitChanged`	called when coupled storage unit is updated
III	`CCD_setCouplingLevel`	called recursively over all parts when a user selects a document-wide coupling level from the standard CoCoDoc menu, which can be installed with the function `CoCoDoc_CopyBaseMenuBar`
IV	`CCD_setOutline`	called recursively over all parts when a user selects a document-wide outline level from the standard CoCoDoc menu

Couplable object groups and the Multicast Object Request Broker

The possibilities for coupling and coupling control depend on the physical distribution of state over the user sites. Broadly, there are three options for physical distribution:

- *centralization*: the state in a coupling level is only present at one user site. It is accessed and updated by the next lower layer in the collaborative zipper architecture.
- *replication*: the state in a coupling level is present at all user sites. All physical copies of the logical state (replicas) must be kept in the same state, if coupled.
- *distribution*: there is one logical state, parts of which may be located at different locations; distribution has both characteristics of centralization and replication; this option will not be considered in this paper.

The type of distribution determines which options are available for coupling of operations. Centralization at a particular level implies coupling at that level and all higher (see Figure 8) levels. For example, if part view control status is centralized, then only at the level of the user interface there is an option concerning coupling. In this situation, it makes no sense to have two document representations which may be coupled. Thus, in order to allow a high flexibility in choice of coupling level, replication should be possible at many levels. Figure 8 illustrates a situation where the part file state is centralized, while the other three levels are replicated and may be coupled and uncoupled dynamically.

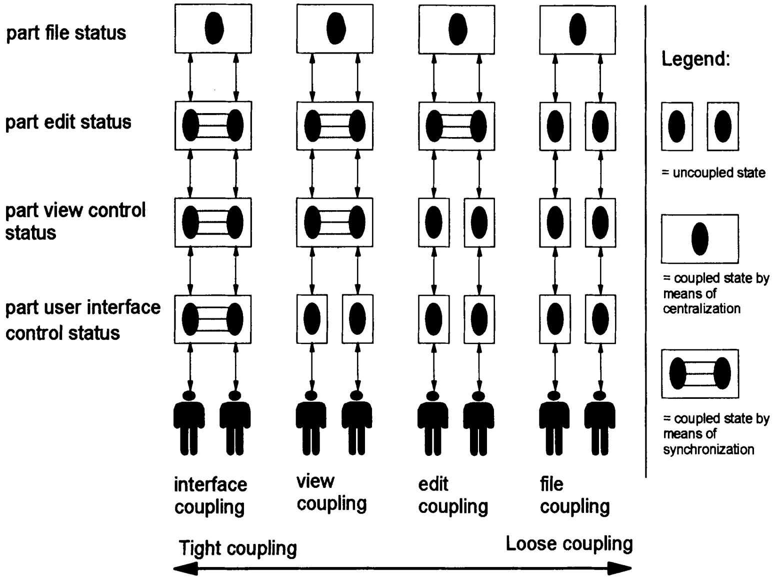


Figure 8 A collaborative zipper with four coupling levels (rows); the distribution choices at each level allow users to choose between four kinds of coupling (columns).

CoCoDoc supports implementation of this kind of collaborative systems (i.e. highly replicated, with flexible coupling possibilities) by means of couplable object groups, which are an extension of object groups (ISIS, 1993), a concept originating from fault tolerant computing. An object group consists of a number of replicas: the member objects. To the world outside the object group, an object group behaves as if it were a single object. Furthermore, implementing an object group is largely similar to implementing a single object. A method invocation on an object group is invoked on all replicas. This ensures state changes are applied to all object replicas. Thus, a high availability of an object's state can be achieved, even if some members are not available due to faults such as system or network crashes. In a *couplable* object group, coupling between replicas can be dynamically switched on or off during its lifetime, with the methods `StartCouple` and `StopCouple`, respectively.

A typical pattern of interaction occurring in collaborative compound document editing systems, as illustrated in Figure 9, is an invocation (e.g. a call on the method `CollapseUnit` of a view object `CoCoTreeUnitView` of a collaborative outline editor), triggered by a user action, that needs to be invoked 'simultaneously' on a number of objects, each located at a user's site, containing a replica of the coupled object, in order to change the state of the shared artifact globally and keep the coupling intact.

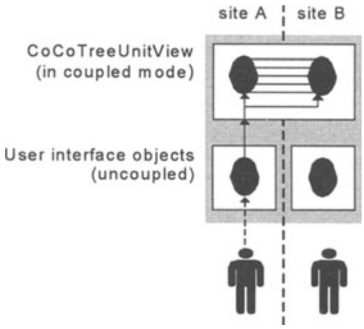


Figure 9 Typical use of object groups.

CoCoDoc provides support for such couplable object groups with a Multicast Object Request Broker (MORB), an extension of current CORBA ORBs and an associated mixing baseclass `CouplableObject`. These facilities provide generic support for multicasting object invocations, multipoint ordering services, response collation, object group membership management and state transfer for newcomers, similar to the object group support provided in the Multiware platform (Costa et al., 1996). In addition to these services from the fault-tolerant domain, it provides coupling and uncoupling of couplable object groups.

5.3 CoCoDoc prototype implementation

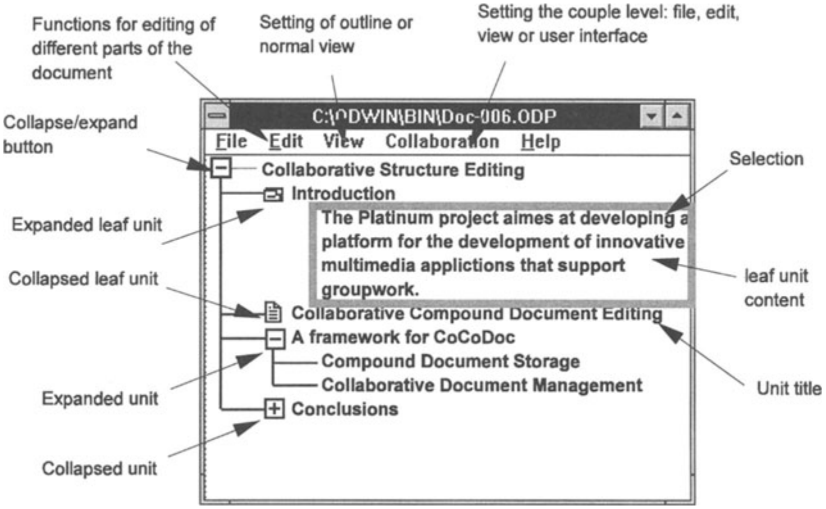
Our current prototype implementation of CoCoDoc is based on the Windows 1.1 version of the OpenDoc developers kit, which is not fully CORBA/DDCF-compliant. No changes were made to OpenDoc baseclasses; the `CoCoDocPart` baseclass was implemented as a subclass of `ODPart` and `CouplableObject`. Furthermore, no changes were made to the OpenDoc shell. This arrangement required a relatively complex implementation of initiation of a collaborative compound document editing conference as a number of related local OpenDoc shells. CoCoDoc is embedded in the Platinum platform that was developed in the Platinum project, which provided facilities for conference management and basic multicasting facilities for the implementation of the MORB.

To achieve minimal additional learning efforts from OpenDoc programmers to use CoCoDoc, the `CoCoDocPart` interface closely resembles the `ODPart` interface. For example, the implementation requirements for `CCD_` methods such as `CCD_InitPart` closely resemble those of their OpenDoc relatives, i.e. `InitPart`. Moreover, other interface extensions (such as those for document-wide coupling and outline control) are kept to a minimum.

6 DEVELOPING A SIMPLE COLLABORATIVE OUTLINE EDITOR

This section briefly describes the process of developing a simple collaborative outline part editor with the CoCoDoc framework (see Figure 10, for a snapshot of its user interface; for a more extensive methodological discussion, we refer to (Moelaert-El Hadidy et al, 1996)).

First, we define (in CORBA IDL) a subclass of `CoCoDocPart`, say `CoCoTreePart`. Implementing `CoCoTreePart` is quite similar to implementing a normal subclass of `ODPart` (Orfali et al., 1996, p. 346-350). Since CoCoTreeParts must be able to embed other parts (e.g. other CoCoTreeParts, since we decide that each `CoCoTreePart` handles only one level of an outline, and delegates the handling of sub-levels to other parts), the optional `ODPart` methods related to embedding must be implemented. The most significant differences compared to OpenDoc programming are the implementation of the additional methods `CCD_createObjectGroupMember`, `CCD_storageUnitChanged`, `CCD_setOutline` and `CCD_setCouplingLevel` on the one hand and the implementation of *part coupling* by using the couplable object group paradigm on the other hand. We will focus primarily on the latter.


Functions for editing of different parts of the document
Setting of outline or normal view
Setting the couple level: file, edit, view or user interface
Collapse/expand button
Selection
Expanded leaf unit
Collapsed leaf unit
leaf unit content
Expanded unit
Unit title
Collapsed unit
C:\ODWIN\BIN\Doc-006.ODP
File
Edit
View
Collaboration
Help
Collaborative Structure Editing
Introduction
Collaborative Compound Document Editing
A framework for CoCoDoc
Compound Document Storage
Collaborative Document Management
Conclusions

The Platinum project aimes at developing a platform for the development of innovative multimedia applictions that support groupwork.

Figure 10 User interface snapshot of a collaborative outline editor based on `CoCoTreePart`.

First, we decide what object (state) can be independently coupled and uncoupled for individual CoCoTreeParts and define separate object groups for these. Since we are implementing a collaboration-aware CoCoDoc-compliant part editor, we need to distinguish at least the three kinds of state as identified in Table 1, associated with the standard document-wide coupling levels: file coupling, edit coupling and view coupling (user interface coupling is currently not supported). There is no need to define a separate object group for file coupling; this is provided by the CoCoDoc framework in the form of coupled storage units (which cannot be uncoupled). For `CoCoTreePart`, no part coupling levels are defined other than the standard levels. So, we have to define a couplable object group (i.e. subclass of the baseclass `CouplableObject`) that encapsulates the editing state of the `CoCoTreePart`, say `CoCoTreeUnitData`, and a couplable object group that encapsulates the view control state of the `CoCoTreePart`, say `CoCoTreeUnitView`. For each couplable object group, we define methods that can update its state (see the second and third row of the last column in Table 1). These methods define the minimum granularity of updates and feedthrough. For example, defining a method `RenameUnit` of `CoCoTreeUnitData` implies that other users, if coupled at edit level, can only observe the result of a complete rename operation; they cannot observe updates to individual letters in the unit title. All the newly defined methods of `CoCoTreeUnitData` and `CoCoTreeUnitView` must be implemented, including overridden baseclass methods from `CouplableObject`.

The `CoCoTreePart` is responsible for creation of the object groups `CoCoTreeUnitData` and `CoCoTreeUnitView`; this must be implemented in `CCD_InitPart` and `CCD_InitPartFromStorage` by instantiating the proper subclass and calling its `createObjectGroup` method, while indicating in a parameter the object group that serves as factory (itself, in this case). This will result in a call to `CCD_createObjectGroupMember` at all peer factories (its peer parts, in this case).

For part coupling, we need to add code to implement additional menu options under the 'Collaboration' menu, such as 'part edit coupling' and 'part view coupling'. Handling these menu selections for part coupling as well as handling `CCD_setCouplingLevel` is implemented in the same way, viz. with calls to `StartCouple` and `StopCouple` on the appropriate couplable object groups.

Being able to embed other OpenDoc parts as unit content requires only a minimal implementation effort; most of the embedding code is already implemented for embedding outline sub-trees. Since OLE2 applications can be embedded in OpenDoc parts as if they were OpenDoc parts, we can use a large variety of content from existing applications (e.g. MS Word) as unit content of our simple collaborative outline editor.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed collaborative compound document editing as new paradigm for editing environments. We also presented the design and implementation of CoCoDoc, a framework based on OpenDoc and CORBA that supports developers in constructing collaborative compound document editing systems. CoCoDoc contributes to various research and development communities.

- To the research and development of collaborative editing environments, CoCoDoc provides a means to reuse existing single user editors in such environments and supports the development of new collaborative part editors. Thus, CoCoDoc facilitates a gradual migration towards collaborative editing environments that are both rich in editing support and rich in collaboration support. Moreover, CoCoDoc and its associated MORB provide couplable object groups as an effective and efficient means to design and implement CSCW applications based on component software.
- To compound document editing in general an OpenDoc in particular, CoCoDoc contributes a standardizable way towards collaborative editing, which requires only minimal extensions to existing interfaces and minimal additional implementation effort from part developers.
- In the area of distributed computing platforms in general and CORBA in particular, CoCoDoc and its associated MORB demonstrate how a distributed computing platform could be extended with couplable object groups to support the needs of an important class of applications of distributed systems, viz. CSCW applications in general and collaborative editors in particular (see also (Ter Hofte et al., 1996a)).

- And last but not least, for end-users, CoCoDoc enables collaborative compound document editing environments that provide a flexible coupling setting per part, less steep learning curves, a means to tailor (i.e. compose and extend) collaborative editing environments to user needs and the emergence of a component software market for collaborative part editors based on open standards.

We implemented a research prototype of CoCoDoc based on the Windows 1.1 version the of the OpenDoc developers kit and implemented a simple collaborative outline part editor on top of it.

Future research and development on CoCoDoc will include making a fully CORBA/DDCF-compliant implementation of CoCoDoc (depending on availability of CORBA/DDCF-compliant framework implementations); enabling WYSIWIS user interface sharing for parts; developing additional collaborative part editors; and extending CoCoDoc with support for timeframes as a means to enable editing of multimedia documents with temporal relations between parts, e.g., to facilitate collaborative timeline editing of multimedia documents.

Acknowledgements

This work was partially funded by the Dutch Ministry of Economic Affairs. The authors wish to thank all Platinum project participants, in particular, Harm Bakker, Raymond Otte, Ferial Moelaert, Wouter Teeuw and Maurice Houtsma.

8 REFERENCES

Apple Computer, Component Integration Laboratories, IBM, and Novell. (1995) *OMG RFP submission : Compound presentation and compound interchange facilities* (OMG document No. 95-12-30). Object Management Group, Framingham, MA, USA. http://www.omg.org/docs/1995/95-12-30.ps.

Baldeschwieler, J.E., Gutekunst, T., and Plattner, B. (1993, April) A survey of X protocol multiplexors. *ACM SIGCOMM Computer Communication Review*, **23**(2), 16-24.

Costa, F.M., and Madeirea, E.R.M. (1996). An object model and its implementation to support cooperative applications on CORBA. In *Distributed Platforms: IFIP/IEEE International Conference on Distributed Platforms, Dresden, February 1996* (eds. A. Schill, C. Mittasch, O. Spaniol, and C. Popien), (p. 213-228). Chapman & Hall, London.

Dewan, P. (1996) Multiuser architectures. In *Engineering for HCI: IFIP WG2.7 Working Conference on Engineering for Human-Computer Interaction, Grand Targhee Resort, Wyoming, USA 1995, August 14-18*, (eds. C. Unger, and L.J. Bass), (p. 247-270). Chapman & Hall, London. ftp://ftp.cs.unc.edu/pub/users/dewan/papers/arch.ps.Z.

Dourish, P. (1995) The parting of the ways : Divergence, data management and collaborative work. In *ECSCW'95 : Proceedings of the fourth European conference on computer-supported cooperative work, 10-14 September 1995, Stockholm, Sweden* (eds. H. Marmolin, Y. Sundblad, and K. Schmidt), (p. 215-230). Kluwer Academic, Dordrecht, the Netherlands. ftp://parcftp.xerox.com/pub/europarc/jpd/ecscw95-divergence.ps.

Ellis, C.A., Gibbs, S.J., and Rein, G.L. (1990) Design and use of a group editor. In *Engineering for human-computer interaction : proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction, Napa-Valley, USA, August 1989* (ed. G. Cockton), (p. 13-28). North-Holland, Amsterdam, the Netherlands.

Engelbart, D.C. (1975) NLS teleconferencing features : The journal, and shared-screen telephoning. In *CompCon75 Digest, September 9-11, 1975* (p. 173-176). IEEE, http://www.bootstrap.org/NLS.ps.

Fish, R.S., Kraut, R.E., Leland, M.D.P., and Cohen, M. (1988) Quilt : A collaborative tool for cooperative writing. In *SIGOIS bulletin: 9(2&3). Conf. on office information systems, March 1988, Palo Alto, USA* (ed. R.B. Allen), (p. 30-37). ACM Press, New York.

Haake, J.M., and Wilson, B. (1992) Supporting collaborative writing of hyperdocuments in SEPIA. In *CSCW'92 : Proceedings of the conference on computer-supported cooperative work, October 31 to November 4 1992, Toronto, Canada* (eds. J. Turner, and R.E. Kraut) ,(p. 138-146). ACM Press, New York.

Ter Hofte, G.H. (1996) *Generic service features of CSCW applications : An analysis of co-authoring tools* (Platinum Deliverable D2.2/011: PLATINUM/N008/V00). Telematics Research Centre, Enschede, the Netherlands.

Ter Hofte, G.H., van der Lugt, H.J., and Bakker, H. (1996a). A CORBA platform for component groupware. In *OzCHI'96 Workshop on the Next Generation of CSCW Systems, Hamilton, New Zealand 1996, November 25*, [Working Paper 96/26] (ed. J.C. Grundy), (p. 31-36). University of Waikato, Hamilton, New Zealand. http://www.trc.nl/publicat/ozchi96.zip.

Ter Hofte, G.H., van der Lugt, H.J., and Houtsma, M.A.W. (1996b) Co^4, a comprehensive model for groupware functionality. In *Telematics in a multimedia environment: Euromedia 96, London, United Kingdom 1996, December 19-21*, (ed. A. Verbraeck), (p. 231-238). Society for Computer Simulation International, Ghent, Belgium.

ISIS Distributed Systems. (1993) *Object groups : A response to the ORB 2.0 RFI* (OMG document No. 94-03-01). Object Management Group, Framingham, MA, USA. http://www.omg.org/docs/1993/93-04-11.ps.

Karsenty, A., and Beaudouin-Lafon, M. (1995) Slice: A logical model for shared editors. In *Groupware for real time drawing: A designer's guide* (eds. S. Greenberg, S. Hayne, and R. Rada), (p. 156-173). McGraw-Hill, New York. http://www.lri.fr/~ak/publis/cscw-book.ps.

Knister, M.J., and Prakash, A. (1990) DistEdit : A distributed toolkit for supporting multiple group editors. *CSCW'90 : Proceedings of the conference on computer-supported cooperative work, October 7-10, 1990, Los Angeles, CA, USA:* , (p. 343-354). Association for Computing Machinery, New York.

Koch, M. (1994) *The unOfficial Yellow Pages of CSCW : Enhanced interface*. WWW-document. http://www11.informatik.tu-muenchen.de/cscw/yp/.

Koch, M. (1995, April 24) *The collaborative multi-user editor project IRIS* (Technical Report, TUM-I9524). Technische Universität München, München, Germany. ftp://hpschlichter18.informatik.tu-muenchen.de/pub/papers/koch95.ps.gz.

Leland, M.D.P., Fish, R.S., and Kraut, R.E. (1988) Collaborative document production using Quilt. In *CSCW 88 : Proceedings of the conference on computer-supported cooperative work, September 26-29, 1988, Portland, Oregon* (p. 206-215). ACM Press, New York.

MacBride, A., and Susser, J. (1996) *Byte guide to OpenDoc*. Osborne McGraw-Hill, Berkeley, CA, USA.

Moelaert-El Hadidy, F., Teeuw, W.B., & Bakker, H. (1996) *An innovative approach for designing collaborative applications using OpenDoc: from theory to practice*. To be published in the proceedings of SEE'97, Cottbus, Germany, April 8-9, 1997.

Nelson, C. (1995, January) OpenDoc and its architecture. *The X resource*, 107-126.

Neuwirth, C.M., Kaufer, D.S., Chandhok, R., and Morris, J.H. (1990) Issues in the design of computer support for co-authoring and commenting. In *CSCW'90 : Proceedings of the conference on computer-supported cooperative work, October 7-10, 1990, Los Angeles, CA, USA* (p. 183-195). ACM Press, New York.

Orfali, R., Harkey, D., and Edwards, J. (1996) *The essential distributed objects survival guide*. Wiley, New York.

Ouibrahim, H., and Schot, J. (1995) Tele-teaching and the electronic superhighway : Towards a vertical approach. In *IDC'95 : First International Distributed Conference on High-Performance Networking for Teleteaching, Madeira, Portugal; Madrid, Spain; Sophia Antipolis, France; Brussels, Belgium, 1995, November 16-17.*

Patterson, J.F. (1995, April) A taxonomy of architectures for synchronous groupware applications. *SIGOIS Bulletin*, **15**, 27-29.

Rüdebusch, T. (1995) Cooperation support. In *Cooperative computer-aided authoring and learning : A systems approach* (ed. M. Mühlhäuser), (p. 249-272). Kluwer Academic, Dordrecht, the Netherlands.

Schlichte, M. (1996) *Mehrbenutzerfähige Verbunddokument-Architekturen [Multi-user compound document architectures]*. M.Sc. Thesis (in German), Technische Universität München, Munich, Germany, ftp://ftp11.informatik.tu-muenchen.de/pub/papers/da-schlichte96.ps.gz.

Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tatar, D. (1987, April) WYSIWIS revisited : Early experiences with multiuser interfaces. *ACM transactions on office information systems*, **5**(2), 147-167.

9 BIOGRAPHY

G. Henri ter Hofte has been an associate member of scientific staff at the Telematics Research Centre since 1993. His professional interests include CSCW, distributed object computing, telework, software engineering and human factors. His Ph.D. research concerns platform support for component-based groupware applications. He holds a cum laude masters degree in Computer Science of the University of Twente, the Netherlands, where he graduated on distribution aspects of graphical user interface software.

Hermen J. van der Lugt is Member of the Scientific Staff of the Telematics Research Centre. He received his Ph.D. in Experimental Physics at the National Institute for Nuclear Physics and High Energy Physics (NIKHEF) in Amsterdam. Topic of his thesis was the design and implementation of a real-time distributed computing network for the data-acquisition of the ZEUS experiment at a electron-proton collider in Hamburg, Germany. He currently coordinates the TRC CSCW research, focusing on modeling, organizational and infrastructure aspects of CSCW applications in organizations, and is involved in several TRC projects in this area.