

Shared Editing

Michel Beaudouin-Lafon

Université Paris-Sud

Concept

Collaborative creation and editing of shared computer artifacts
- Typically a shared *document*
- All users have the illusion that they edit the *same* document

Notion of *group awareness*
- Knowing what the others are doing
-> different from, e.g., a multi-user database

Notion of *collaborative task*
- Users work towards the same goal
- Implicit or explicit coordination of their actions

Types of shared editors

Different document types: text, graphics, spreadsheet, etc.

Synchronous: Changes immediately visible to all

Asynchronous: Changes visible to others at a later time

Homogeneous: All users must use the same software

Heterogeneous: Users can use different software

Collaboration-aware: Include group awareness features

Collaboration-transparent: No group awareness features

The notion of congruence

Stefik et al., 1987

View congruence
Part of the document being viewed

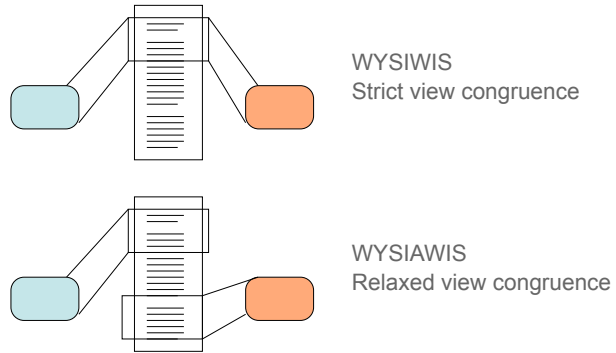
Display space congruence
Organization of the windows

Time of display congruence
When changes are seen by other users

Subgroup congruence
Users who see the changes



WYSIWIS / WYSIAWIS



Sample shared editors (historical)

Text, asynchronous (different time)

- Quilt (Leland, Fish & Kraut, 1988)
- Prep (Neuwirth et al., 1989)

Text, synchronous (real-time)

- Grove (Ellis, Gibbs & Rein, 1989)
- ShrEdit (McGuffin & Olson, 1992)
- SASSE (Baecker et al., 1993)

Graphics, synchronous (real-time)

- GroupDesign (Karsenty & Beaudouin-Lafon, 1992)

Real-time text editor: GROVE

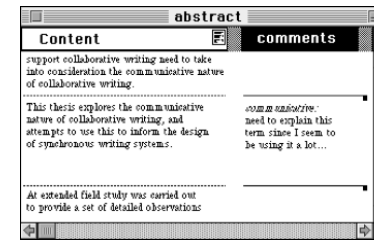
Ellis et al., 1989

Group Outline Viewing Editor

- Concurrent editing at the character level
- Private, Shared and Public views
- Clouds to show the activity of other users
- Aging text: blue at first, then progressively black

Asynchronous text editor: Prep

Neuwirth et al., 1992

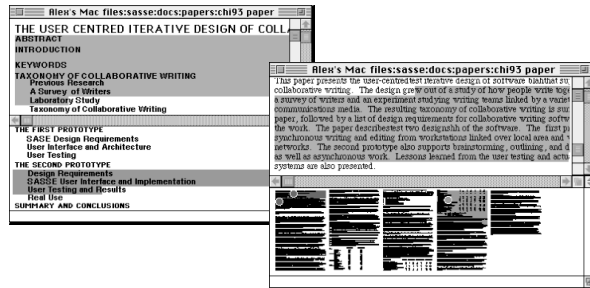


Real-time text editor: Sasse

Baecker et al., 1993

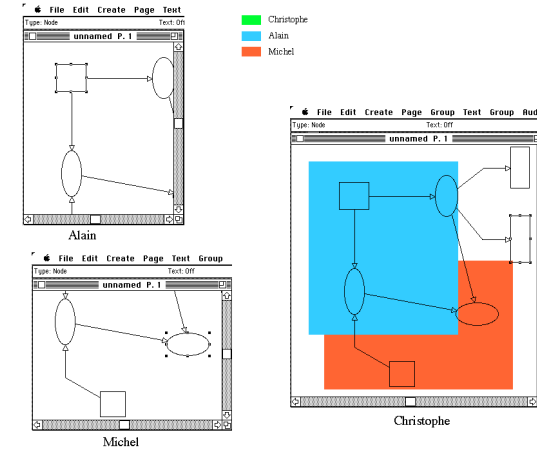
Group-awareness widgets

- Scrollbars
- Radar view



Real-time graphics: GroupDesign

Karsenty, 1992

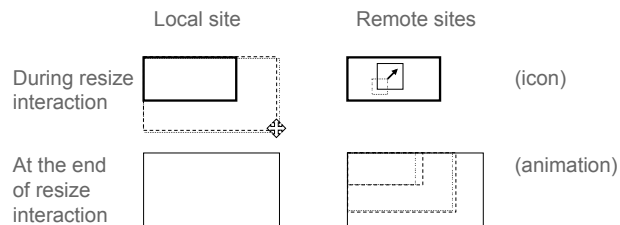


GroupDesign

Karsenty, 1992

Group-awareness features:

- Show participants as colors
- Immediate feedback of commands for the local user
- *Echo* of the command for the other users, until completed



Single-display groupware

Connect multiple input devices to a single computer+display
 Useful in colocated situations, e.g. classrooms and meetings
 Also applies to tabletop interfaces

MMM (Bier & Freeman, 1991)
 fine-grain shared editing

KidPad (Druin et al.): local tools

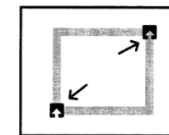
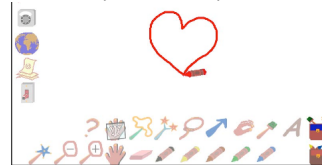
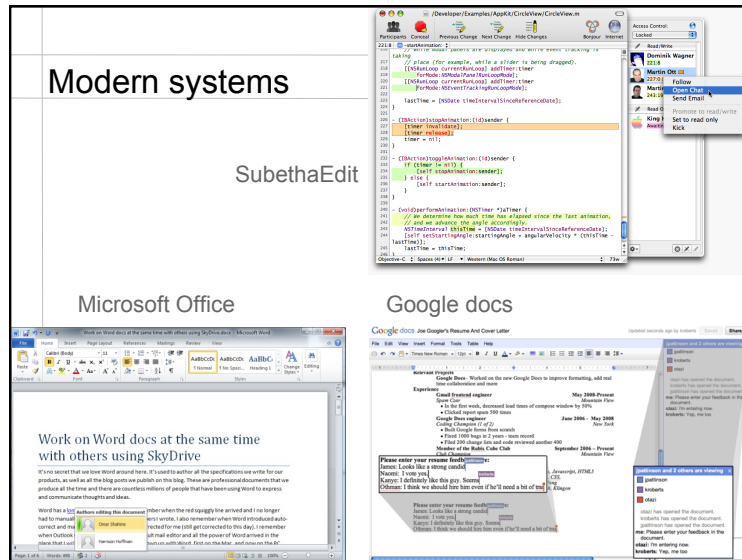


Figure 4. Two users stretching a rectangle at the same time.





Modern systems

SubethaEdit

Microsoft Office

Google docs

Work on Word docs at the same time with others using SkyDrive

If you've seen that we have Word documents here it's used in a number of the specifications we write for our products, as well as all the blog posts we publish on this blog. These are professional documents that we produce all the time and there are countless millions of people that have been using Word to express and communicate thoughts and ideas.

Word has a lot of features calling this document... when the red squiggly line arrived and no longer had to reformat... Dear Steve... I'm not sure if you've noticed this today, I remember... I'll send you a link to the document... I'll send you a link to the document...

Problems of modern systems

Homogeneous

All users must use the same application

Mostly cloud-based

Who owns your documents and where are they?

What if you do not have network access?

Do not support different levels of *coupling*

Strong coupling: pure WYSIWIS

Loose coupling: WYSIAWIS

Very loose coupling: asynchronous

Implementation of real-time groupware

Approaches

Collaboration-transparent system

- Wrapping a single-user application
- Screen and window sharing
- Turn taking
- Example: VNC

Collaboration-aware system

- Designed from the start for collaborative work
- Consistency of distributed copies
- Robustness: a failure of a distant network or computer should not affect the local user
- Example: *Google Docs*

Some vocabulary

Participant: a user in a session
 Session: one or more documents, edited by one or more users
 Invitation: giving a user access to a session
 Floor control: policy for managing input from multiple users
 Turn-taking: Floor control where one user can edit at a time
 Telepointer: visualization of one's cursor on other users' screens

Coupling: how local actions are tied to remote actions
 Response time: time for an action to be executed locally
 Notification time: time for an action to be executed remotely
 Replication: transparently managing multiple copies of a document
 Robustness: sensitivity to remote faults

Implementation

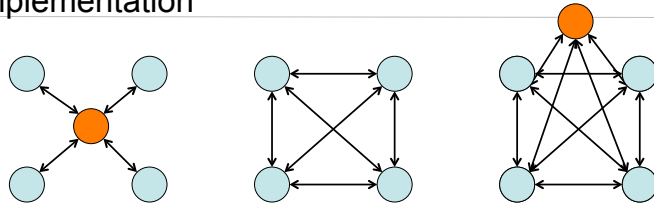
Some similarities with operating systems and databases:

- Several users, geographical distribution, concurrent access, replication, faults...
- BUT groupware tries to be *transparent*, i.e. not hide users

Specific issues:

- Group awareness
 - View congruence (WYSIWIS, WYSIAWIS)
 - Feedthrough (telling other users what I am doing)
- Latecomers
 - Getting users that arrive during the session up to speed

Implementation



Three main types of software architecture:

- Centralised: simple, but low response time, brittle
- Replicated: good response/notification time, but complex
- Hybrid: mostly replicated with some centralized functions

Development tools:
 screen sharing, shared window systems, groupware toolkits

Managing conflicts

Problem: consistency of distributed data
 Each site generates events and sends them to other sites
 Each site must execute the events so that the result is consistent across sites

Two classes of algorithms

- pessimistic (locks)
- optimistic (events + undo)

Optimistic algorithms:

- operational transformation, e.g. dOpt (GROVE)
- optimized undo/redo, e.g. ORESTE (GroupDesign)

Causality and logical clocks

Strong notion of causality

If A happened before B, then A must be executed before B
(because A may have influenced B)

Total ordering of events: Lamport's logical clocks

One logical clock per site (counter)

Incremented for each local event, Sent with each event

When an event arrives with a timestamp t

if $t > \text{localclock}$ then $\text{localclock} \leftarrow t + 1$

Timestamp defines a partial order of events

Turned into a global order with an ordering of sites

$(t_1, s_1) < (t_2, s_2)$ iff $t_1 < t_2$ or $(t_1 = t_2 \ \& \ s_1 < s_2)$

Undo-redo algorithm

Principles

Every operation op must have an inverse op^{-1}

Each site maintains a history of events

$(op_1, t_1, s_1) \dots (op_n, t_n, s_n)$

When an event arrives out of sync

(op_i, t_i, s_i) with $(t_i, s_i) < (t_n, s_n)$

Undo the operations between i and n

Execute op_i

Redo operations between i and n

ORESTE

Karsenty & Beaudouin-Lafon, 1993

Principle

- Consistent state when the system is quiescent (all sent messages have been received and processed)
- Uses Lamport timestamps for total ordering
- Undo/redo when a message arrives out of order

Optimizing undo/redo

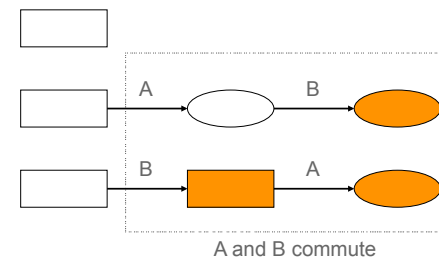
- Concept of *compatible order*
- Take advantage of *commutativity* and *masking* between operations
- Use total order in case of a conflict

ORESTE : commutativity

A changes the shape to an ellipse

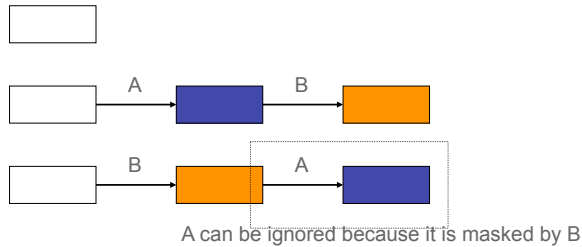
B changes the color to orange

Total order is A then B



ORESTE : masking

A changes color to blue
 B changes color to orange
 Total order is A then B



Text editing: problem

Text is a sequence of characters
 Each user represented by the offset of his/her cursor

Basic operations:

Move cursor forward, backward

Insert character

Delete character

Problem:



Text editing: solving the problem?

When inserting at C, cursors after it should move to the right
 When deleting at C, cursors after it should move to the left



Is this sufficient?

Unfortunately not:

Problems occur when users move their cursors

Text editing: not solving the problem

Cursor motion includes *relative* information
 This leads to different results depending on event order



The resulting text is the different,
 and the cursors are in different positions!

Solution: Operational Transform (OT) Ellis et al., 1989

Total ordering of operations (Lamport timestamps)
 When an operation arrives out of order, it is *transformed*:
 It is modified to take into account the effects of the operations that have occurred since it was issued

For each pair of operations op_1, op_2 ,
 where op_2 arrived after op_1 but occurred before it,
 we need a transformation $T(op_1, op_2) = op'_2$ so that
 $op'_2(op_1(\text{text})) = op_1(op_2(\text{text}))$

When an operation arrives, it is transformed by those that
 have occurred since then
 Note: this requires a potentially unbounded history buffer

Operational Transform: example

Forward transformation: include impact of op_2 into op_1
 $T(\text{insert}(p_1, c_1, ts_1), \text{insert}(p_2, c_2, ts_2))$
 if $(p_1 < p_2)$ or $(p_1 = p_2 \text{ and } ts_1 < ts_2)$
 then return $\text{ins}(p_1, c_1, ts_1)$
 else return $\text{ins}(p_1+1, c_1, ts_1)$

Backward transformation: exclude impact of op_2 from op_1
 $T^{-1}(\text{insert}(p_1, c_1, ts_1), \text{insert}(p_2, c_2, ts_2))$
 if $(p_1 < p_2)$ or $(p_1 = p_2 \text{ and } ts_1 < ts_2)$
 then return $\text{ins}(p_1, c_1, ts_1)$
 else return $\text{ins}(p_1-1, c_1, ts_1)$

Operational Transform: example

Basic operations include the character position
 Cursor positions are only managed locally



Note that each cursor is after the character inserted by the user.

Operational transform

Writing the transformations is hard
 Proving that they work is even harder (in fact, most don't!)

Properties:
Causality preservation: operations that depend on each other are executed in the same order at each site
Convergence: same state at each site when all messages have been processed
Intention preservation: matching what the user meant

A free Javascript library: www.sharejs.org
 Other libraries exist for other languages

Groupware toolkits

Groupware toolkits

Embed concurrency algorithms into a library

Provide *groupware widgets* to support group awareness

Examples:

DistEdit (Prakash, 1990)

Suite (Dewan, 1990)

Rendez Vous (Patterson et al., 1990)

GroupKit (Roseman & Greenberg, 1992)

MEAD (Bentley et al., 1994)

Prospero (Dourish, 1996)

DAC (Tronche, 1998)

GroupKit

Developed at the University of Calgary GroupLab

Toolkit developed in Tcl/Tk

- Prototyping and development of shared real-time applications
- Research and teaching about CSCW

Features

- Session management (participants joining and leaving)
- Supports data distribution (1:1, 1:n)
- Specific widgets for collaborative interaction

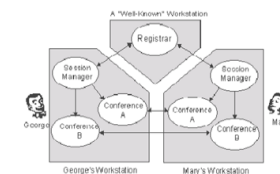
Available: www.groupkit.org

GroupKit : architecture

Registrar : centralized process accessible by all computers

Session manager : process managing conferences and access control for one participant

Conference : replicated process managing a single conference



GroupKit : awareness widgets

Who is participating?
Where are they?
What can they see?

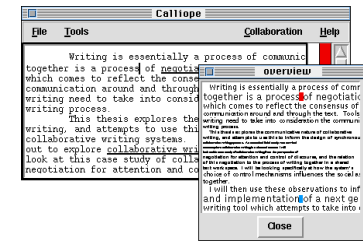
What is their activity level?
What do they do?
What do they need?

What are they going to do?
What can they do?

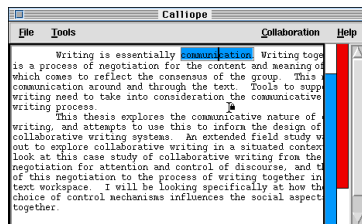


Telepointers
Multi-scrollbars
Radar views
Fish-eye views

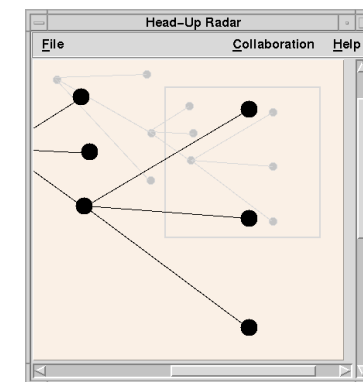
Telepointers



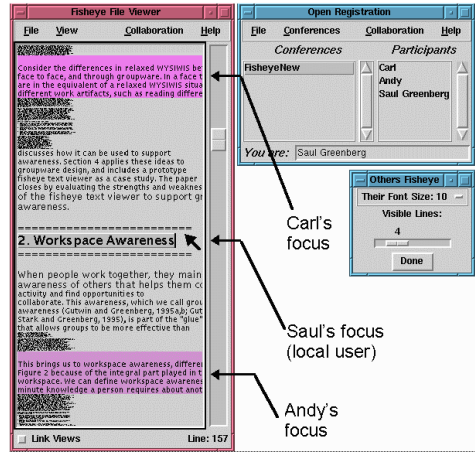
Multi-scrollbars



Radar view



Fish-eye view



GroupKit : applications



Brainstorming
Text chat



Drawing (bitmaps or vectors)
Graph editing



File browsers
Text editors



Games (tic-tac-toe, cards, tetriminos)

Conclusion

Shouldn't shared editing be part of every software application?

Is the move towards cloud-based applications a good thing?