Shared Editing

Michel Beaudouin-Lafon

mbl@lisn.fr
Université Paris-Saclay

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

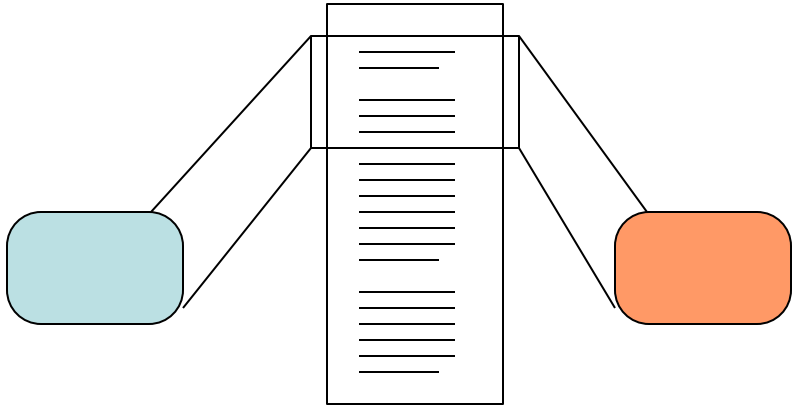
Time of display congruence

When changes are seen by other users

Subgroup congruence

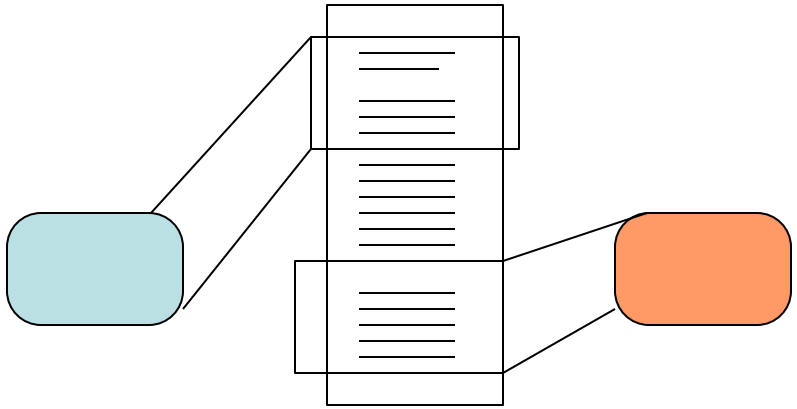
Users who see the changes

WYSIWIS / WYSIAWIS



WYSIWIS

Strict view congruence



WYSIAWIS

Relaxed view congruence

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

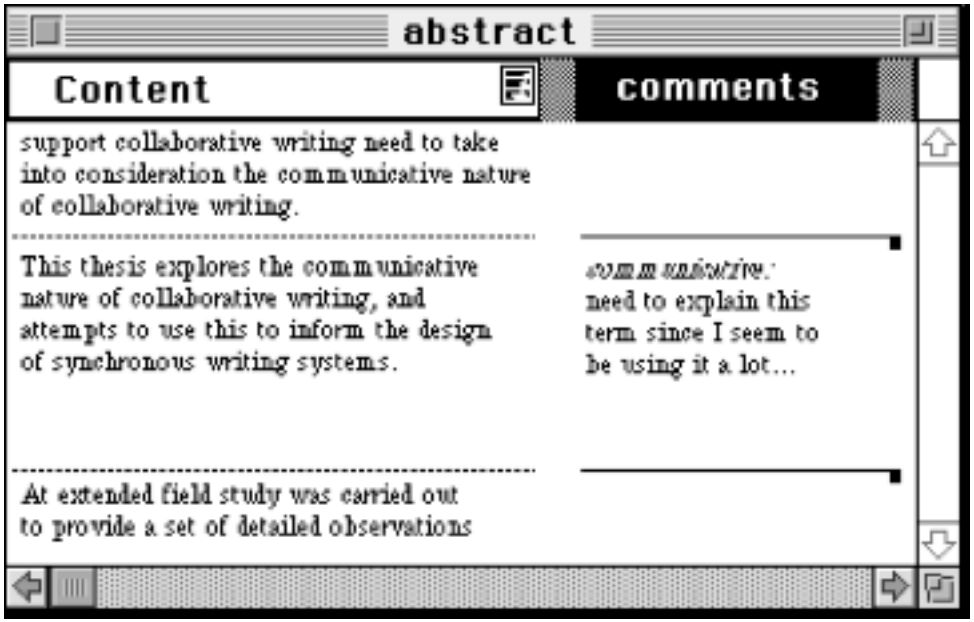
Ellis et al., 1989

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

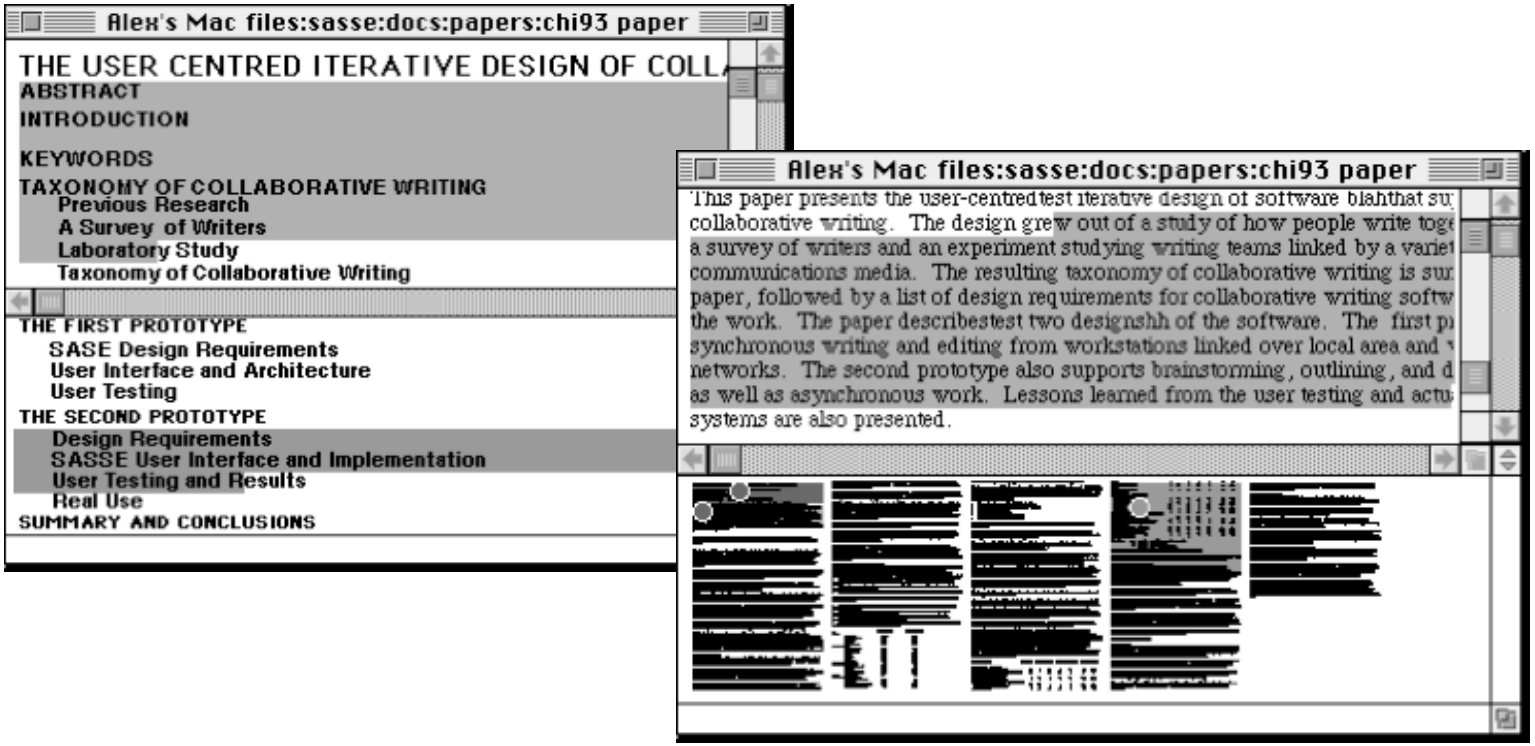


Real-time text editor: Sasse

Baecker et al., 1993

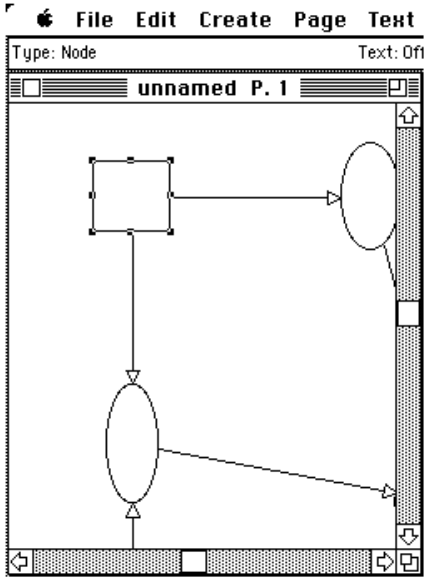
Group-awareness widgets

- Scrollbars
- Radar view

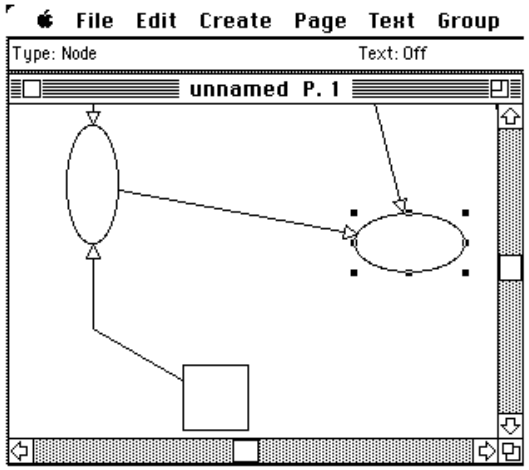


Real-time graphics: GroupDesign

Karsenty, 1992

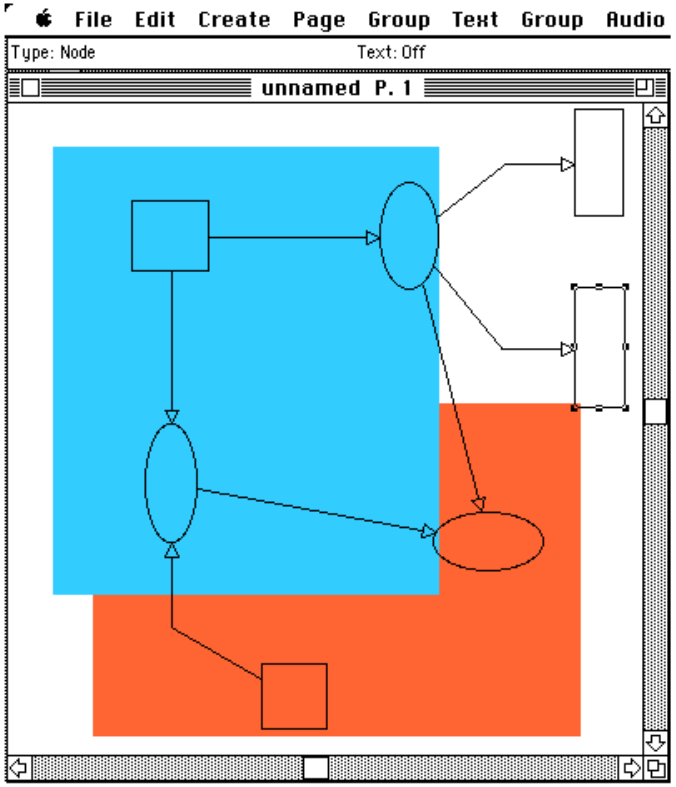


Alain



Michel

Christophe
Alain
Michel



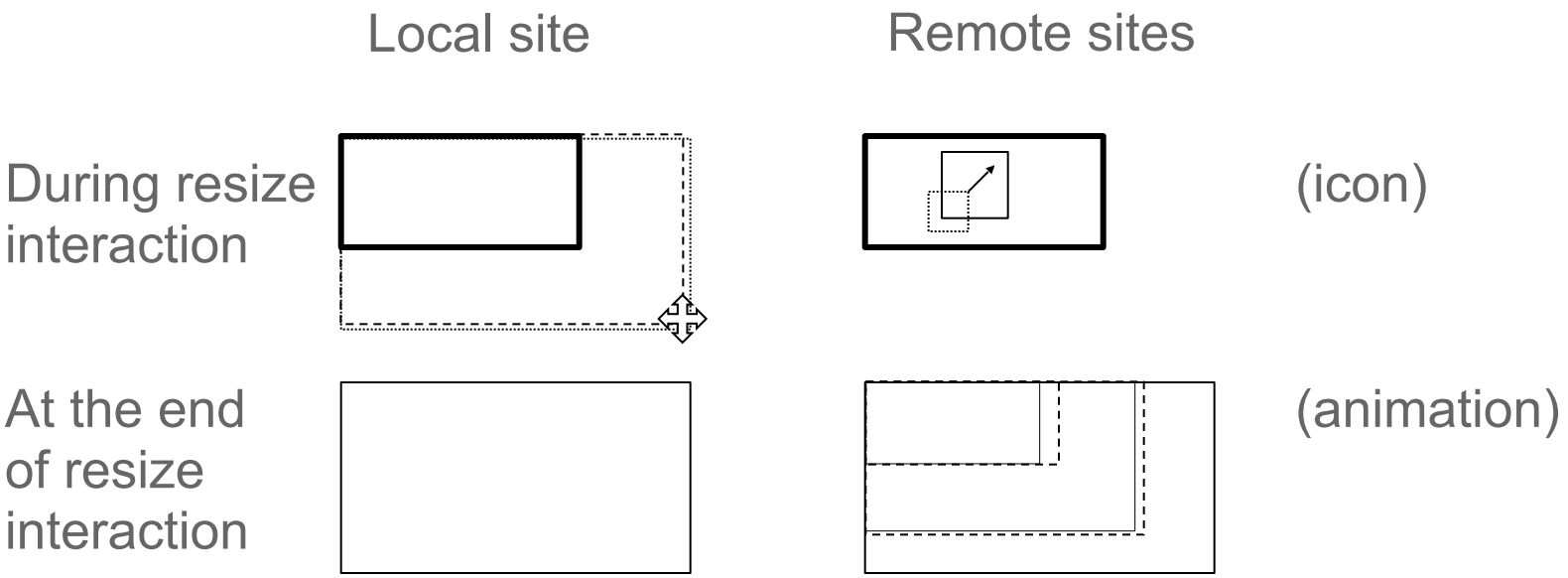
Christophe

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

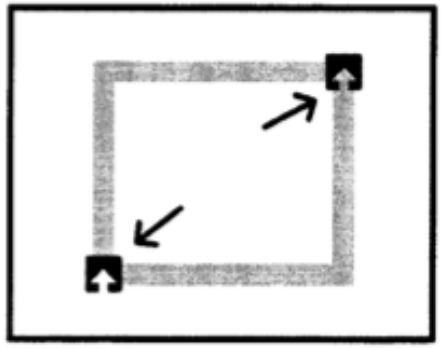
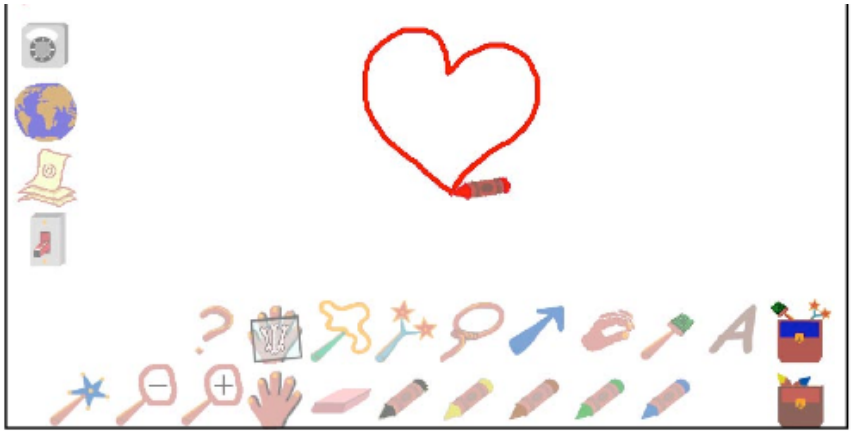


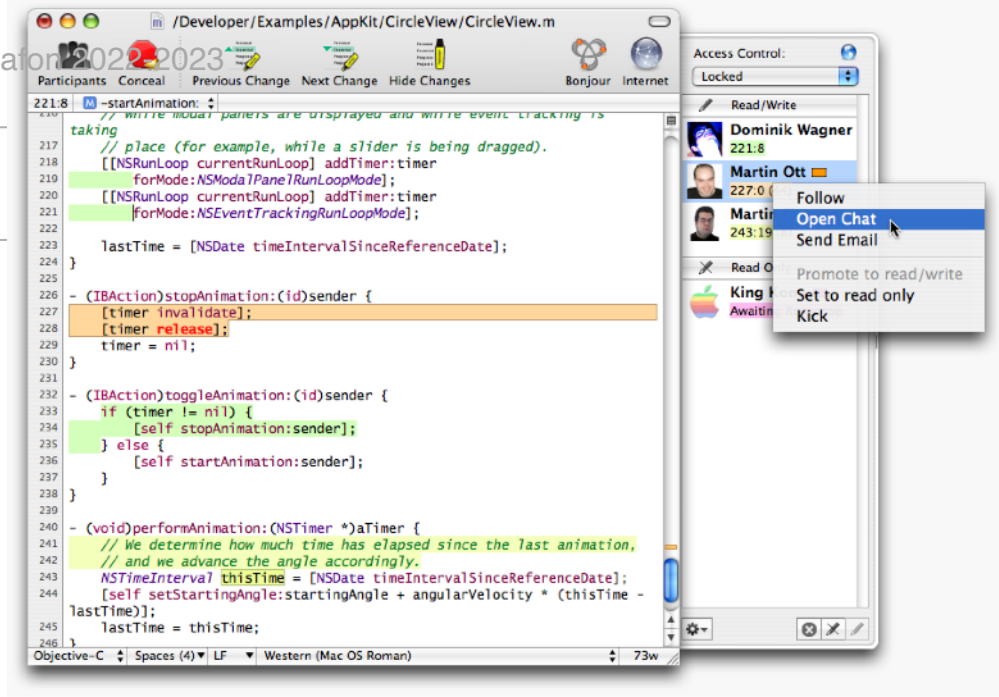
Figure 4. Two users stretching a rectangle at the same time.

KidPad (Druin et al.): local tools

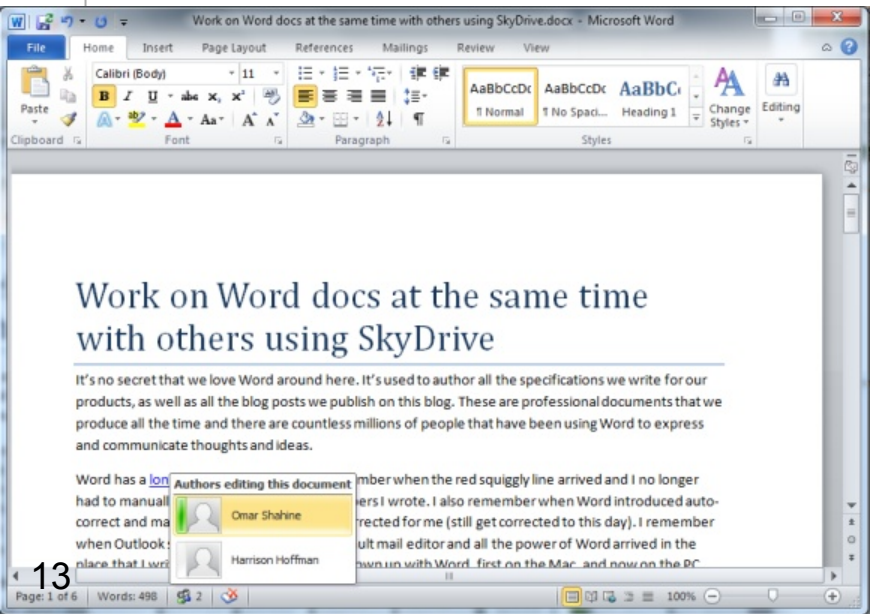


Modern systems

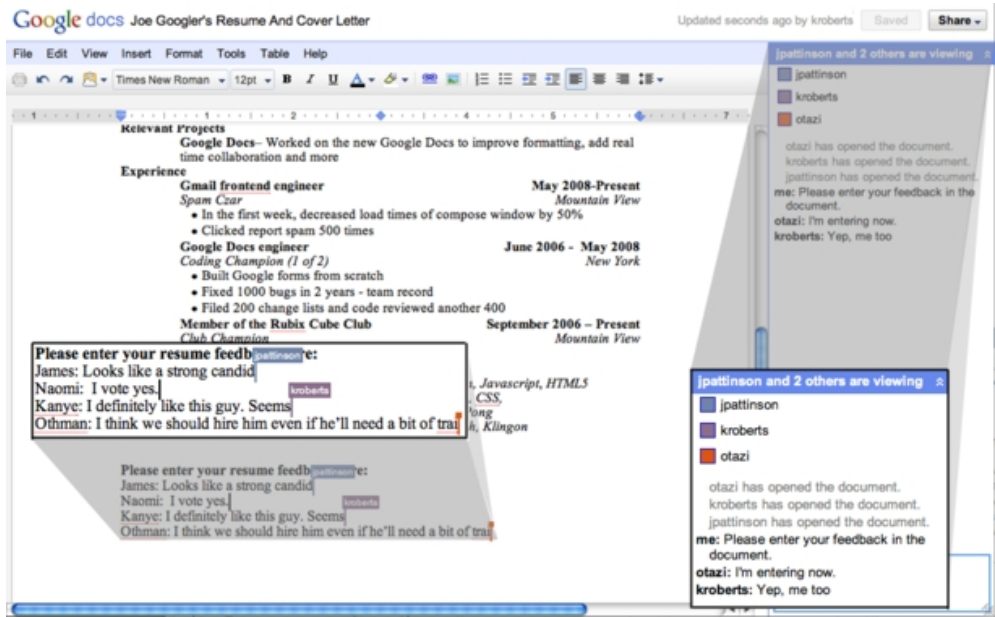
SubethaEdit



Microsoft Office



Google docs



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?

Support a **single level of *coupling***

- Strong coupling: pure WYSIWIS (ex: VNC)
- Loose coupling: WYSIAWIS (ex: Google docs)
- Very loose coupling: asynchronous (ex: GitHub)

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

Groupware vs. operating systems & databases

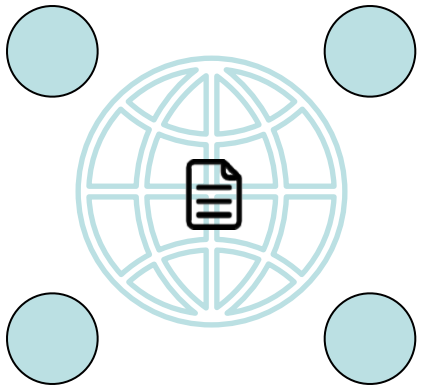
Some similarities with operating systems and databases:

- 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

Distributed real-time groupware



Replicated copies

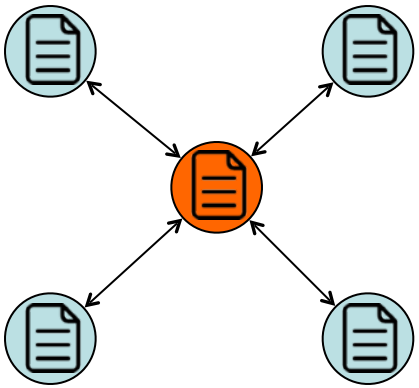
A copy of the document
(a “replica”)
resides at each node

Synchronized copies

Changes must be propagated
to all replicas

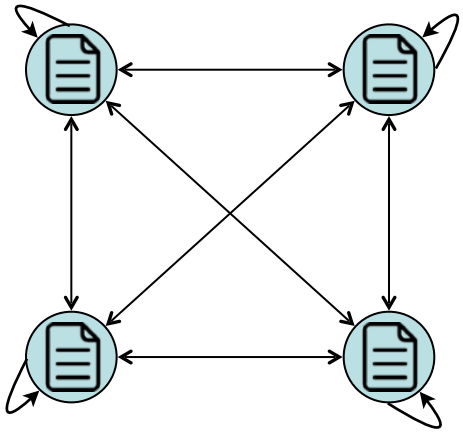
But beware of conflicts!

Distributed architectures



Centralised

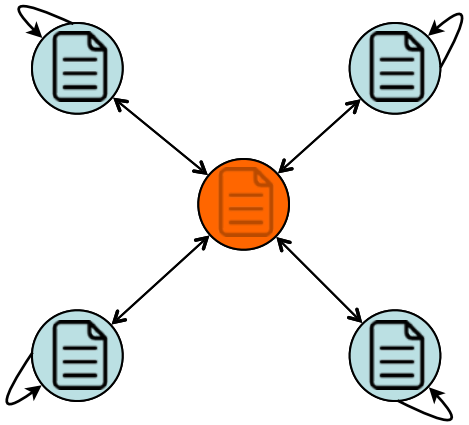
Send events to server
to serialize them
+ easy to implement
– low response time
– brittle



Fully replicated (peer-to-peer)

Handle events locally and
send them to everyone else
+ good response time
+ good notification time
+ robust
– complex to implement

Distributed architectures



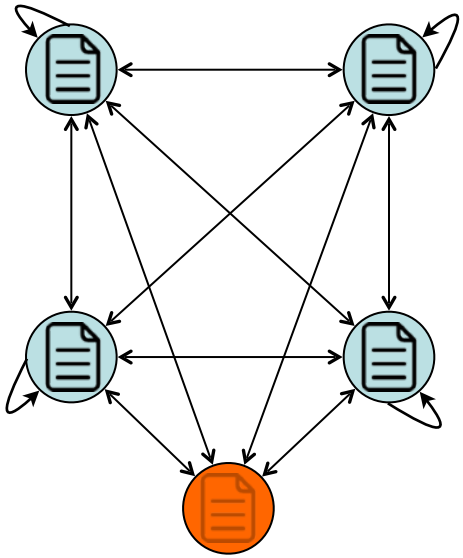
Hybrid

Handle events locally and send them to server

+ good response time

~ less brittle

~ harder to implement



Hybrid

Replicated with some centralized functions

+ good response time

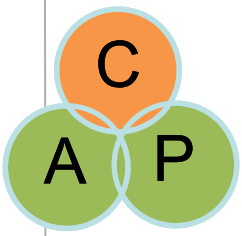
+ good notification time

~ easier to implement

Fundamental problem: the CAP theorem

Theory of distributed systems:

A distributed store can only provide 2 of these 3 properties:



Consistency: every read gets the most recent write

Availability: every request receives a response

Partition tolerance: the system works despite messages being dropped or delayed

In groupware, we typically want **Partition** tolerance, so we have to choose between **Consistency** and **Availability**

Most groupware systems choose **Availability**

Therefore the state is **not always consistent** across nodes

Managing conflicts

Problem: ***eventual*** 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 eventually consistent across sites

Two classes of algorithms

- pessimistic (locks)
- optimistic (events)

Optimistic algorithms:

- optimized undo/redo, e.g. ORESTE (GroupDesign)
- operational transformation, e.g. dOpt (GROVE)
- Conflict-free Replicated Data Types - CRDTs

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 > localClock **then** localClock <- t +1

Timestamp defines a partial order of events

Turned into a global order with an ordering of sites

(t1, s1) < (t2, s2) **iff** t1 < t2 or (t1 = t2 and s1 < s2)

Undo-redo algorithm

Principles

- Every operation op must have an inverse op^{-1}
- Each site maintains a history of events

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

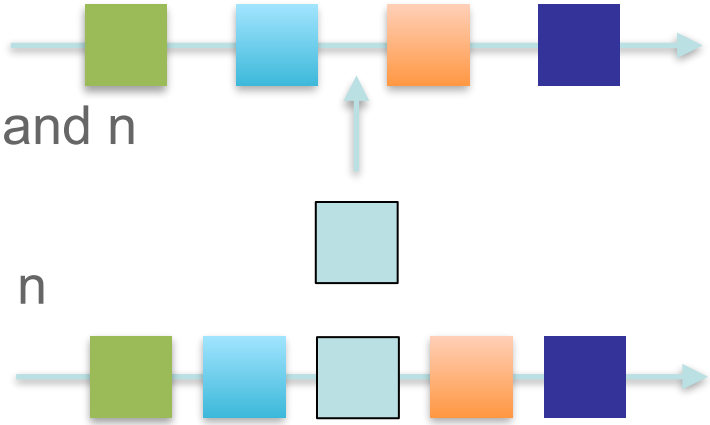
- When an event arrives out of sync

  (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

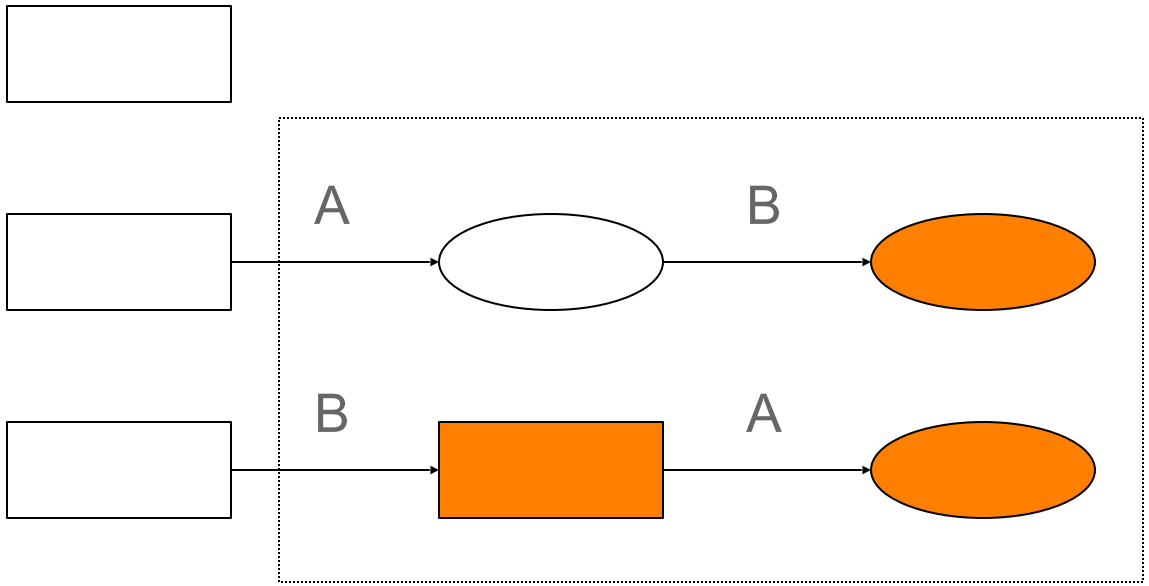
- Concept of *compatible order*
- Take advantage of *commutativity* and *masking* between operations
- Use total order in case of a conflict

ORESTE : commutativity

A changes the shape to an ellipse

B changes the color to orange

Total order is A then B



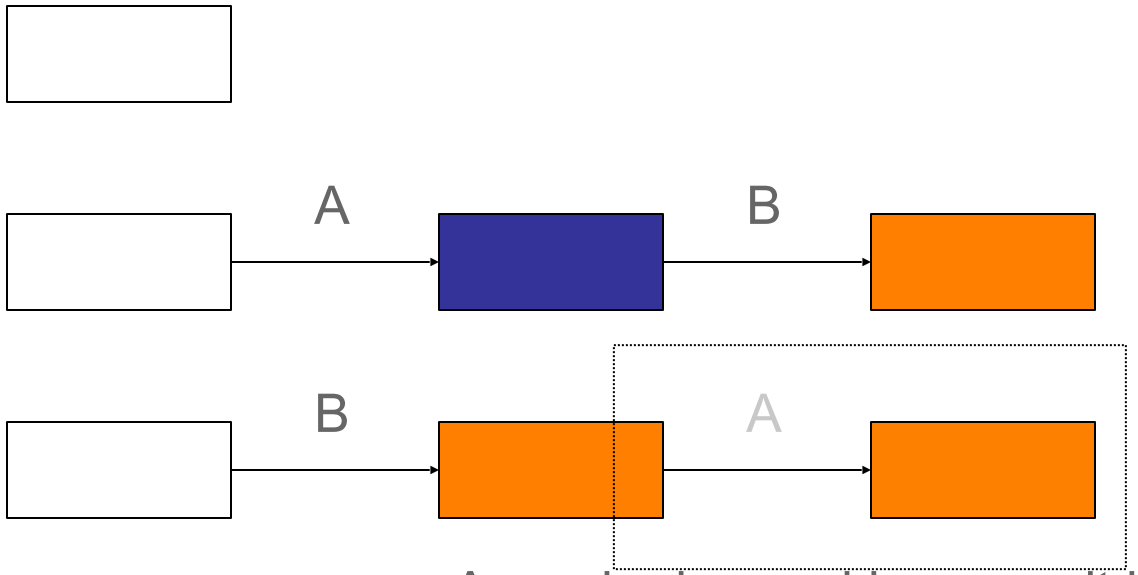
A and B commute

ORESTE : masking

A changes color to blue

B changes color to orange

Total order is A then B



A can be ignored because it is masked by B

Text editing: problem

Text is a sequence of characters

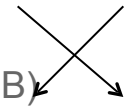
Each user represented by the offset of his/her cursor

Basic operations:

- Move cursor forward, backward
- Insert character
- Delete character

Problem:

Site A	Site B	
Hello |w|orld	Hello |w|orld	
Hello m||world	Hello ||orld	(B deletes character)
Hello |w|orld	Hello |m|orld	(B receives insert m at A)
(A receives delete B)		



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

Site A		Site B	
Hello |w|orld		Hello |w|orld	
Hello m|w|orld	(A inserts m)	Hello ||orld	(B deletes character)
Hello m||orld	(A receives delete B)	Hello m||orld	(B receives insert m at A)

Is this sufficient?

Unfortunately not:

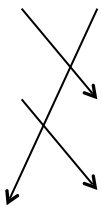
Problems occur when users move their cursors

Text editing: not solving the problem

Cursor motion includes *relative* information

This leads to different results depending on event order

Site A	Site B
Hello w|o|rld	Hello w|o|rld
Hello wo||rld (A moves right)	Hello w||rld (B deletes character)
Hello w||rld (A deletes character)	Hello w|r|ld (B receives move A right)
Hello ||rld (A receives delete B)	Hello w||ld (B receives delete A)



The resulting text is 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 op1, op2,
where op2 arrived after op1 but occurred before it,
we need a transformation T(op1, op2) = op'2 so that

op'2(op1(text)) = op1(op2(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 op2 into op1

T(insert(p1, c1, ts1), insert(p2, c2, ts2)):

if (p1 < p2) or (p1 = p2 and ts1 < ts2)

then return ins (p1, c1, ts1)

else return ins (p1+1, c1, ts1)

Backward transformation: exclude impact of op2 from op1

T^{-1}(insert(p1, c1, ts1), insert(p2, c2, ts2)):

if (p1 < p2) or (p1 = p2 and ts1 < ts2)

then return ins (p1, c1, ts1)

else return ins (p1-1, c1, ts1)

Operational Transform: example

Basic operations include the character position

Cursor positions are only managed locally

Site A	Site B
Hello |world	Hello |world
Hello a|world (A inserts a at pos 6)	Hello b|world (B inserts b at pos 6)
A receives insert b at pos 6 in order	B receives insert a at pos 6 out of order
Hello ba|world (A inserts b at pos 6)	Hello b|aworld (B inserts a **at pos 7**)

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

Operational transform

Writing the transformations is hard

Proving that they are correct is even harder (in fact, most are not!)

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: ShareJS - https://sharejs.org

Other libraries exist for other languages

Other approach: CRDTs

Shapiro et al., 2011

Conflict-Free [or Convergent] Replicated Data Types

Idea: data types designed such that independent replicas
can be updated without coordination,
and it is always possible to resolve inconsistencies

Two types: Operation-based and State-based CRDTs

We focus on **operation-based CRDTs**
We assume that the transport layer
does not lose nor duplicate messages
We use logical clocks to tag messages

Javascript library: Yjs https://yjs.dev

Example CRDT: LWW-Record

Last-Write-Wins Record represents an object mapping property names to their value

Operations:

Set(obj, prop, value)

Get(obj, prop)

obj	updated
p_1: v_1	ts_1
p_2: v_2	ts_2
…	…
p_n: v_n	ts_n

Implementation:

Each property has a value and logical clock of last update

op("set", ts, obj, prop, val):

if (obj.updated(prop) < ts) obj[prop] = val; obj.updated(prop) = ts

op("get", ts, obj, prop):

return obj[prop]

Example CRDT: OR-Set

Observed-Remove Set: set of values
Each value can be in the set at most once
If it has been removed, it cannot be added again

Operations: add(set, elem), remove(set, elem), has(set, elem)

Implementation:
Each element has a list of add-tags and remove-tags

op("add", ts, set, elem): add "ts" to the set of add-tags of elem

op("remove", ts, set, elem): copy the tags in the add-tags list of elem to the remove-tags list of elem => Tombstone

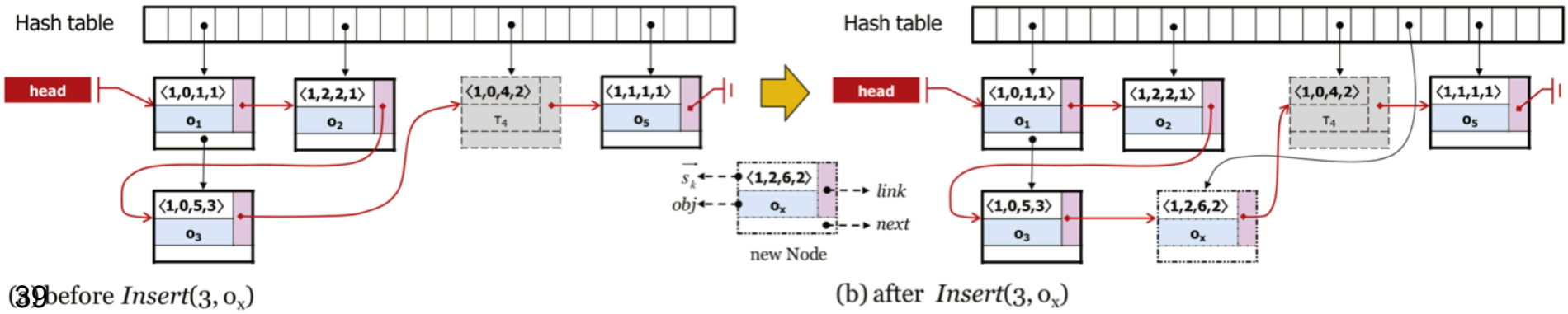
op("has", ts, set, elem): return true if add-tags - remove-tags is non empty, false otherwise

Example CRDT: RGA

Replicated Growable Array: an ordered sequence of values
where values can be inserted and deleted

Operations: insert(seq, afterElem, elem), remove(seq, elem)

Implementation: complex!
uses a hash table and a vector of timestamps
must keep tombstones for elements that have been removed



(a) before $Insert(3, o_x)$

(b) after $Insert(3, o_x)$

Summary

Real-time shared editing is complex!

Maintaining consistency among replicas depends on many factors:

- What type of network? (reliabillity)
- How many prospective simultaneous users? (scalability)
- Connected vs. disconnected mode? (resilience)
- What type of data to be shared?

The algorithms are complex and sometimes wrong!

Trust existing libraries rather than build your own

Groupware toolkits

Groupware toolkits

Provide *groupware widgets* to support group awareness

Embed concurrency algorithms (sometimes)

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)
- **WebStrates (Klokmose et al., 2015)**

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

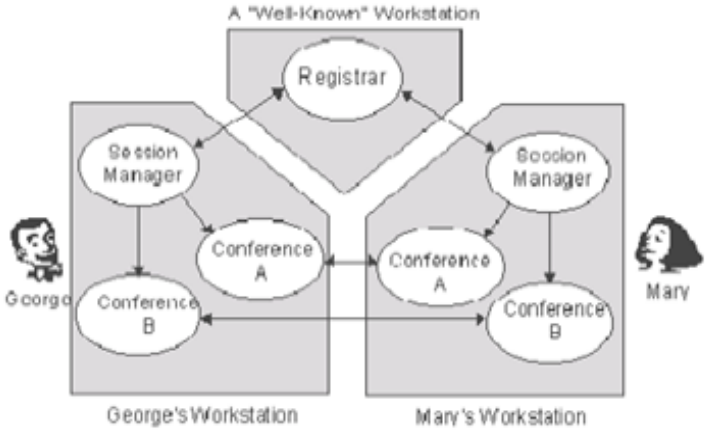
Available: *www.groupkit.org*

GroupKit : architecture

Registrar : centralized process accessible by all computers

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

Conference : replicated process managing a single conference

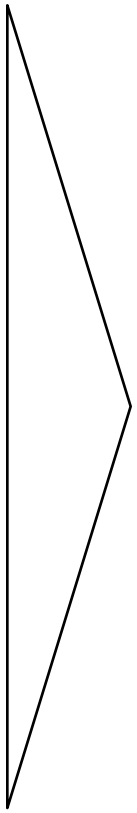


GroupKit : awareness widgets

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

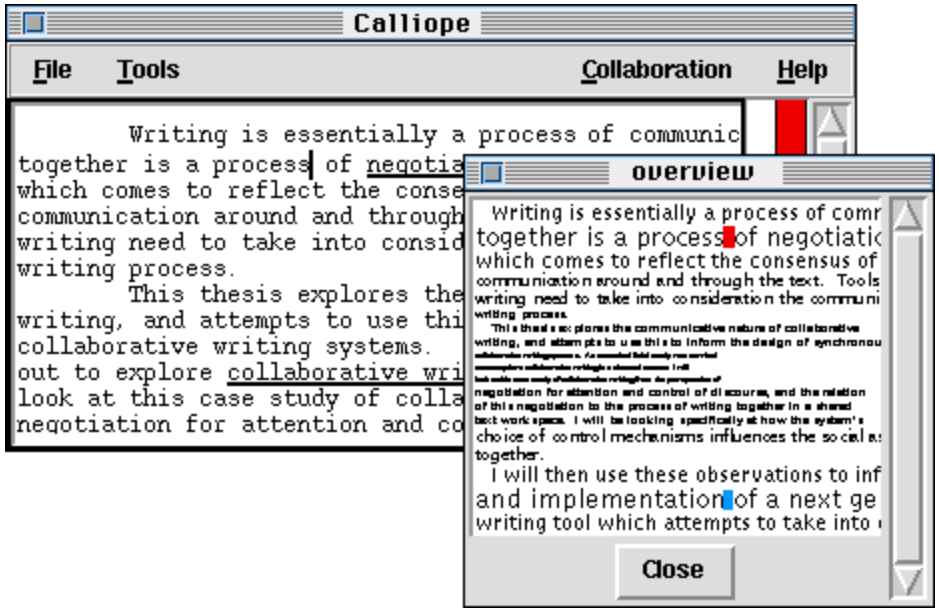
What is their activity level?
What do they do?
What do they need?

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

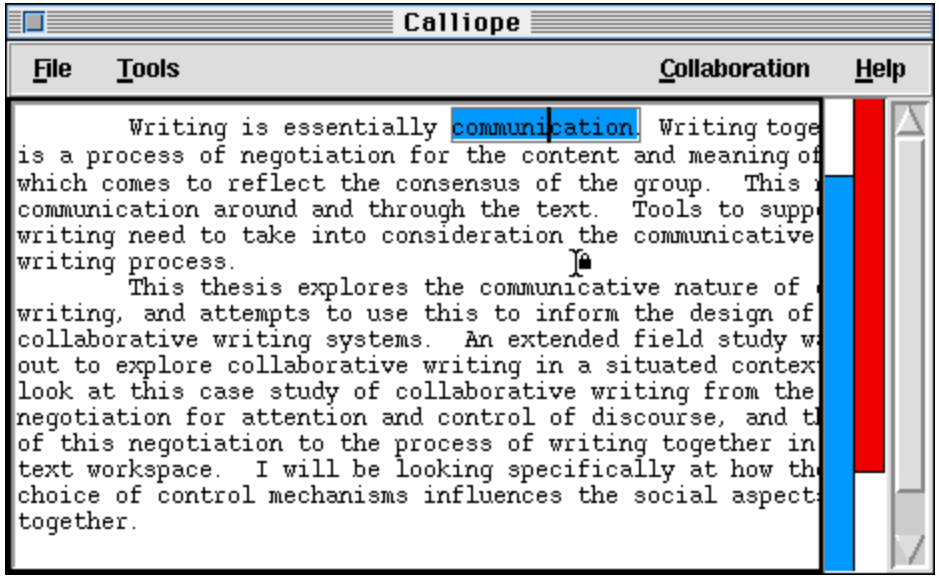


Telepointers
Multi-scrollbars
Radar views
Fish-eye views

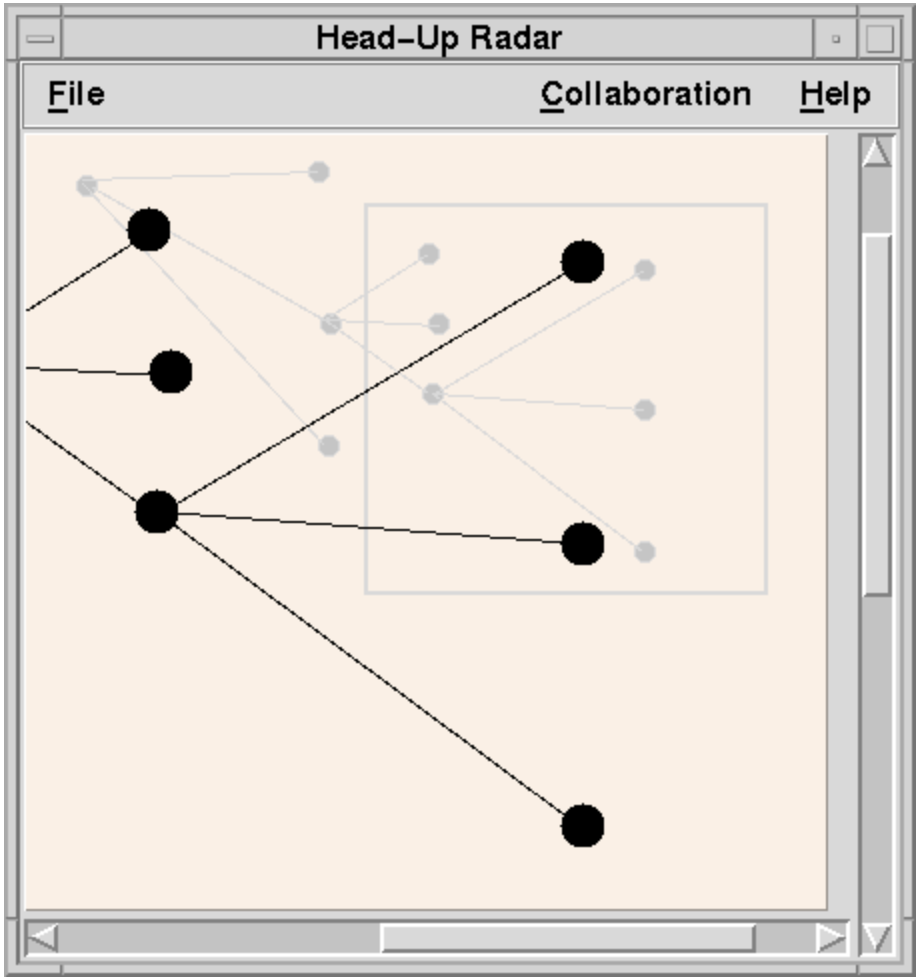
Telepointers



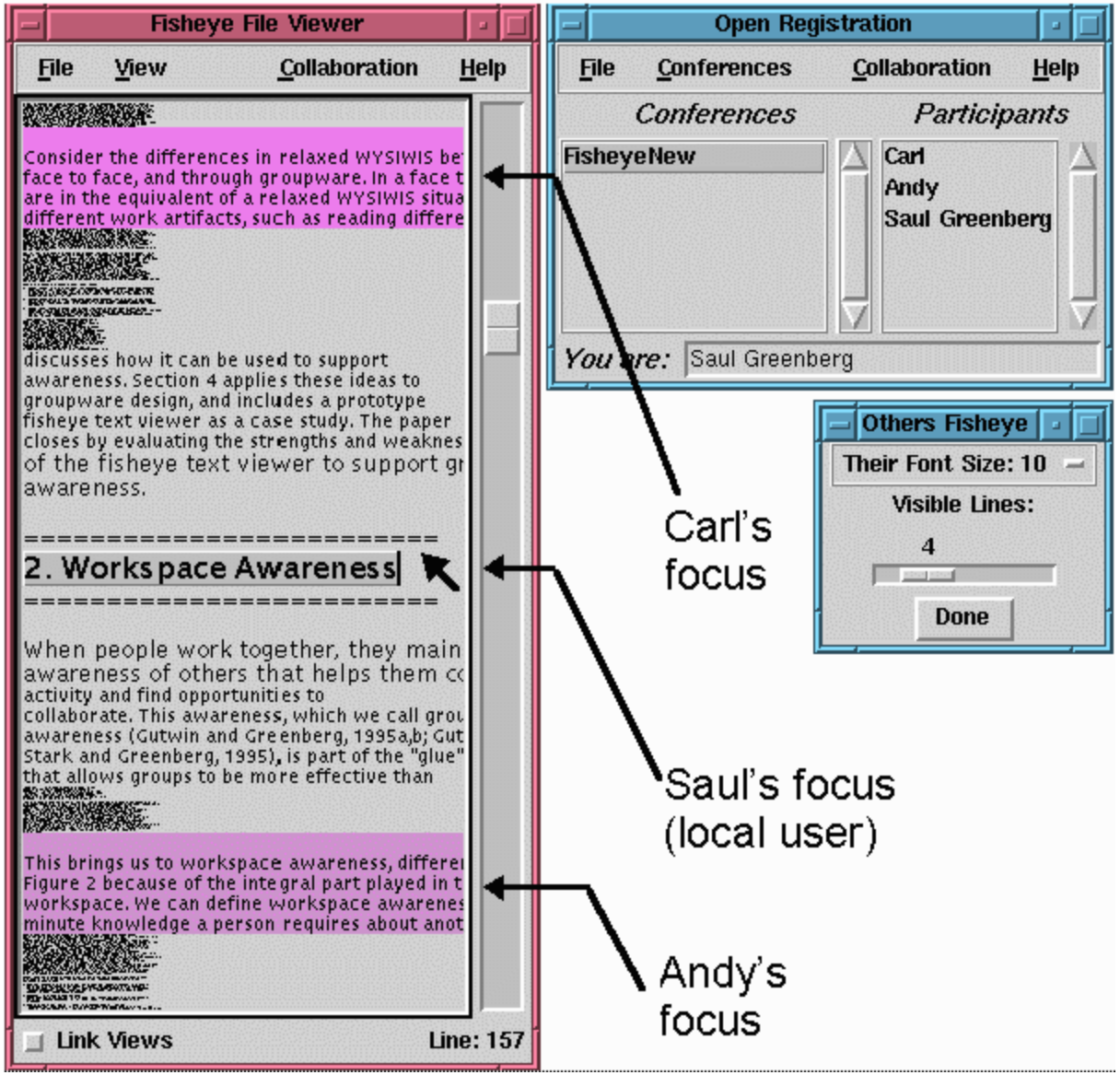
Multi-scrollbars



Radar view

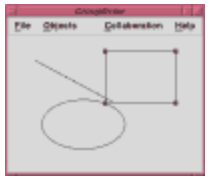


Fish-eye view



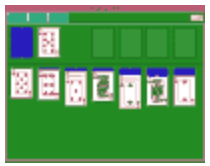
GroupKit : applications

Brainstorming
Text chat



Drawing (bitmaps or vectors)
Graph editing

File browsers
Text editors



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

Webstrates

Klokmose et al., 2015

http://webstrates.net

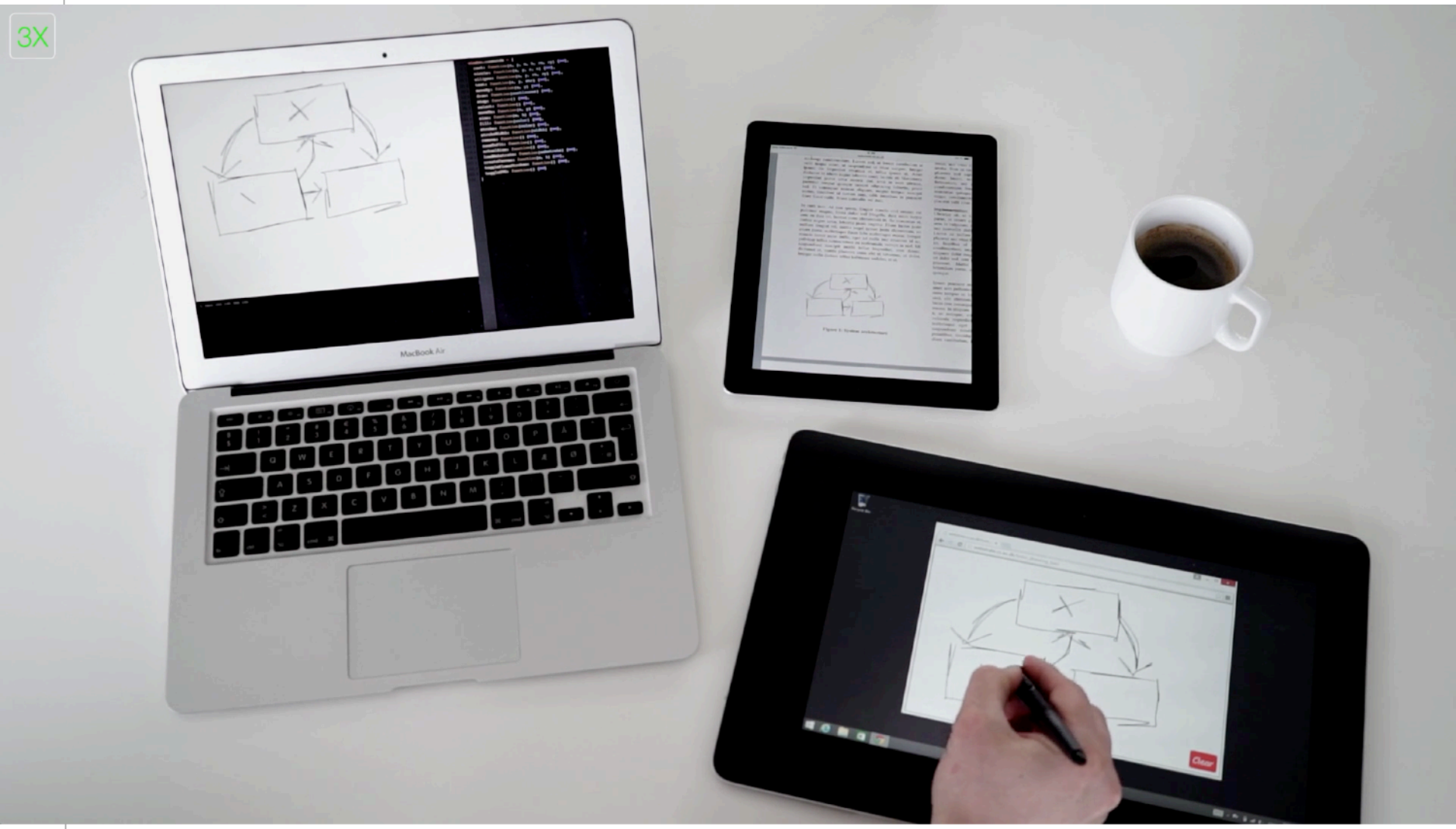
Shareable Dynamic Media

- Content that is inherently shared
- No distinction between application and document
- Collaboration, sharing and distribution across devices

Webstrate = Web Substrate

- Web document served by a webstrate server
- Any change to the DOM of a document is replicated on any client of that document
- Transclusion: a document can be embedded into another document, and is still editable

Webstrates

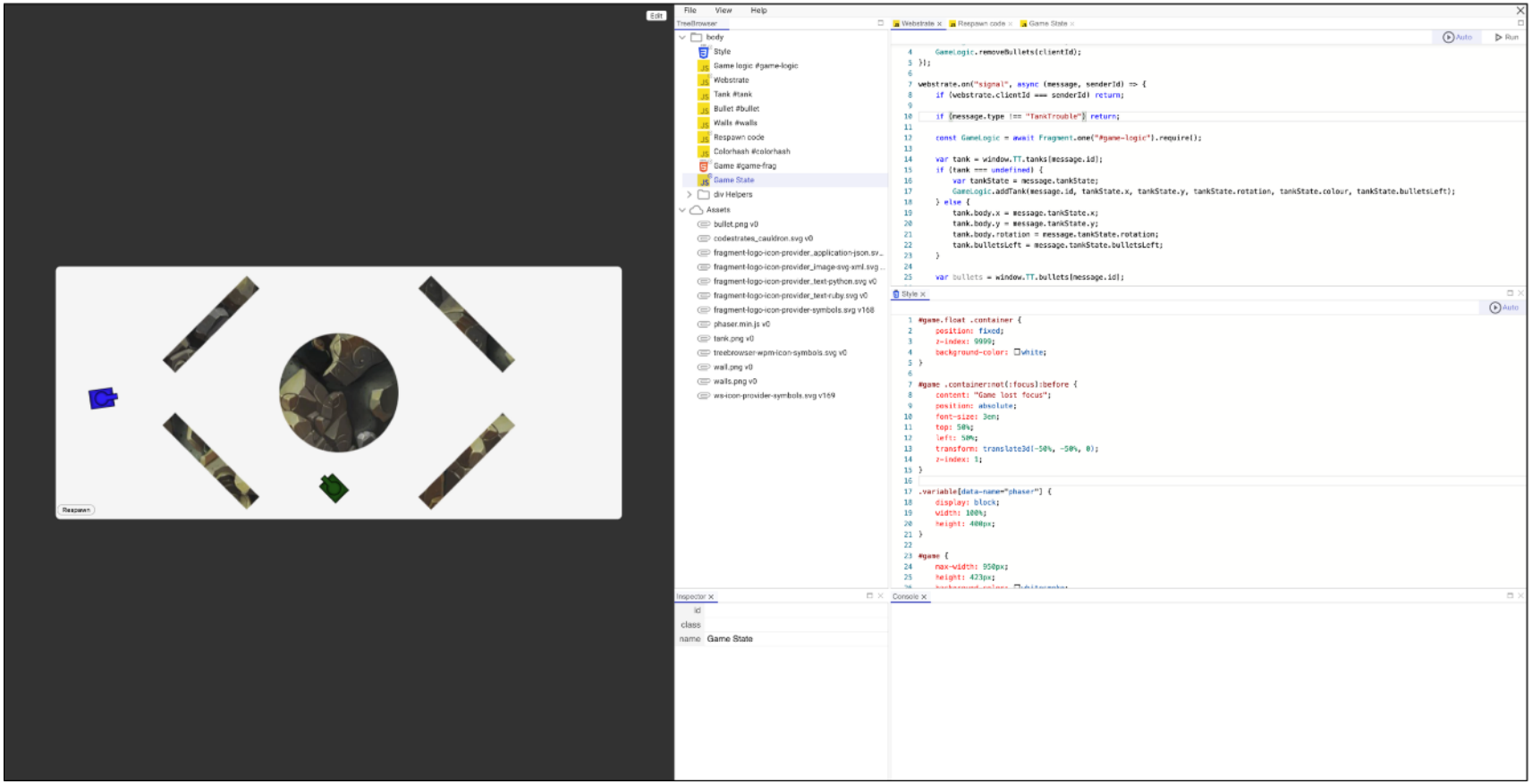


Codestrates

Borowski et al., 2022

https://codestrates.projects.cavi.au.dk/

Development environment for Webstrates



Conclusion

Shouldn’t shared editing be part of every content-editing application?

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