Master Informatique Paris-Saclay - (c) Michel Beaudouin-Lafon 2012-2021

Shared Editing

Michel Beaudouin-Lafon

mbl@lri.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

Subgroup congruence Users who see the changes

Organization of the windows

Display space congruence

Time of display congruence

View congruence

Part of the document being viewed

When changes are seen by other users

The notion of congruence



4

Stefik et al., 1987

Master Informatique Paris-Saclay - (c) Michel Beaudouin-Lafon 2012-2021

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

abstr	act	
Content	comments	
support collaborative writing need to take into consideration the communicative natu of collaborative writing.	re	<u></u>
This thesis explores the communicative nature of collaborative writing, and attempts to use this to inform the design of synchronous writing systems.	commaniatrie: need to explain this term since I seem to be using it a lot	
At extended field study was carried out to provide a set of detailed observations		■ ⊽
4		90

Real-time text editor: Sasse

Baecker et al., 1993

Group-awareness widgets

- Scrollbars
- Radar view

Alex's Mac files:sasse:docs:papers:chi93 pape	er 📃 💷	
THE USER CENTRED ITERATIVE DESIGN OF CO ABSTRACT INTRODUCTION		
KEYWORDS	Alou's Mar filos:sassordors:nanors:chi03 nanor	ΠĒ
TAXONOMY OF COLLABORATIVE WRITING Previous Research	This paper presents the user-centred test iterative design of software blahthat su	1日
A Survey of Writers	collaborative writing. The design grew out of a study of how people write toge	
Laboratory Study	a survey of writers and an experiment studying writing teams linked by a variet 💻	
Taxonomy of Collaborative Writing	communications media. The resulting taxonomy of collaborative writing is sur	
	paper, followed by a list of design requirements for collaborative writing softw the work. The paper describestest two designship of the software. The first p	
SASE Design Dequirements	synchronous writing and editing from workstations linked over local area and y	
User Interface and Architecture	networks. The second prototype also supports brainstorming, outlining, and d	
User Testing	as well as asynchronous work. Lessons learned from the user testing and actu	
THE SECOND PROTOTYPE	systems are also presented.	-
Design Requirements SASSE User Interface and Implementation		\$
User Testing and Results	المستقدين فتقصفنان فتقتصف المتعادية	
SUMMARY AND CONCLUSIONS		
		Ri

Karsenty, 1992

Real-time graphics: GroupDesign



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.



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?

Do not support different levels of *coupling* Strong coupling: pure WYSIWIS Loose coupling: WYSIAWIS Very loose coupling: asynchronous

Master Informatique Paris-Saclay - (c) Michel Beaudouin-Lafon 2012-2021

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 (peer-to-peer): good response/notification time, but complex
- Hybrid:

mostly replicated with some centralized functions

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

С

Α

Ρ

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

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

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:

Site A	Site B	
Hello world	Hello world	
Hello m world (A inserts m)	, Hello <mark> </mark> orld	(B deletes character)
Hello world (A receives delete B)	Hello morld	(B receives insert m at A)

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 world	Hello world	
Hello m w orld (A inserts m)	, Hello orld	(B deletes character)
Hello morld (A receives delete B)	Hello morld	(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 world	Hello world
Hello wo rld (A moves right)	Hello w rld (B deletes character)
Hello w rld (A deletes character)	Hello wridd (B receives move A right)
Hello I rld (A receives delete B)	Hello wild (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

 A receives insert b at pos 6 in order
 B receives insert a at pos 6 out of order

 Hello b |world
 B receives insert a at pos 6 out of order

 Hello b |world
 B receives insert a at pos 6 out of order

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 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
p1: v1	ts1
p2: v2	ts2
pn: vn	ten

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



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

Master Informatique Paris-Saclay - (c) Michel Beaudouin-Lafon 2012-2021

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

	Calliope			
<u>F</u> ile	<u>T</u> ools		<u>C</u> ollaboration	<u>H</u> elp
togeth which commun writin writin collab out to look a negoti	Writi er is a comes t ication g need g proce This g, and orative explor t this ation f	ng is essentially a process of <u>negotia</u> o reflect the conse around and through to take into consid ss. thesis explores the attempts to use thi writing systems. e <u>collaborative wri</u> case study of colla or attention and co	process of communic overview Writing is essentially a proce together is a process of r which comes to reflect the co communication around and through th writing need to take into consideration writing med to take into consideration writing need to take into the process of writing together the the negotidation to the process of writing together the two of spaces. I will be looking significally at the choice of control mechanisms influence	ess of comr negotiatic nsensus of the communi the communi of collebondive ign of enchronou and the matetion aris in a dward or the social a:
			I will then use these observa and implementation of a writing tool which attempts to	tions to inf a next ge) take into (

Multi-scrollbars

	Calliope
<u>F</u> ile <u>T</u> ools	<u>C</u> ollaboration <u>H</u> elp
Writing is esser is a process of negotiat which comes to reflect to communication around and writing need to take int writing process. This thesis expl writing, and attempts to collaborative writing sy out to explore collabora- look at this case study negotiation for attention of this negotiation to to text workspace. I will choice of control mechan together.	tially communication Writing toge ion for the content and meaning of he consensus of the group. This is through the text. Tools to support o consideration the communicative is the communicative nature of use this to inform the design of stems. An extended field study we tive writing in a situated contex of collaborative writing from the n and control of discourse, and the he process of writing together in be looking specifically at how the isms influences the social aspects

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

Accompanying video for the paper Webstrates: Shareable Dynamic Media

Clemens N. Klokmose¹, James R. Eagan², Siemen Baader¹, Wendy Mackay^{3,4}, & Michel Beaudouin-Lafon^{4,3}

¹Aarhus University, ²Télécom ParisTech, ³INRIA, ⁴Université Paris-Sud & CNRS

UIST 2015

Conclusion

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

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