CISC 839
Software Engineering of Usable Computing Systems

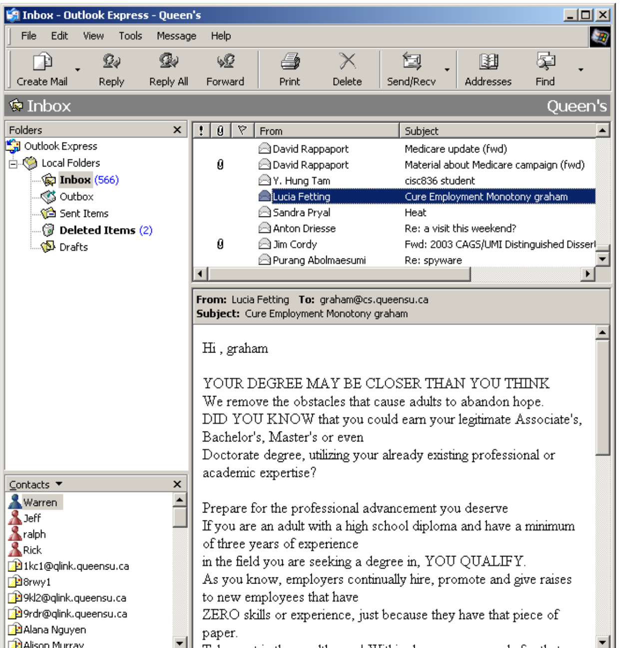
T.C. Nicholas Graham

http://www.cs.queensu.ca/~graham/cisc836

Implementing Groupware

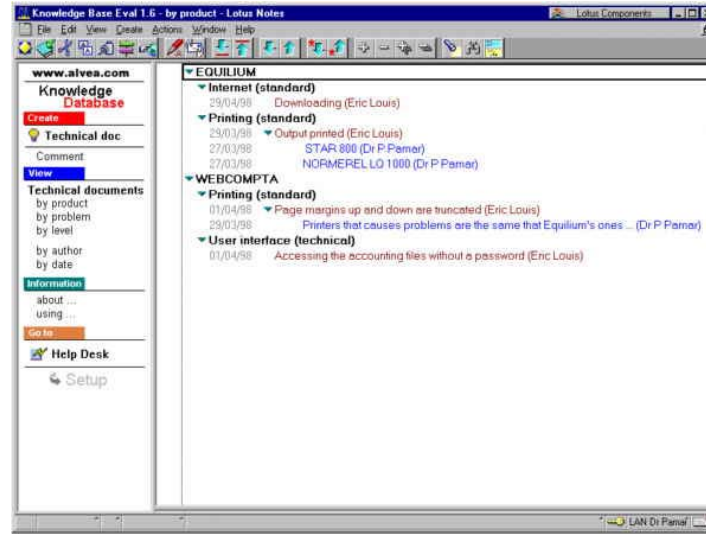
Examples of Asynchronous Groupware Products

- ⌘Email
- ⌘Discussion Forum
- ⌘Shared document space
- ⌘Shared repository



Outlook Express

WebCT Forum



Lotus Notes

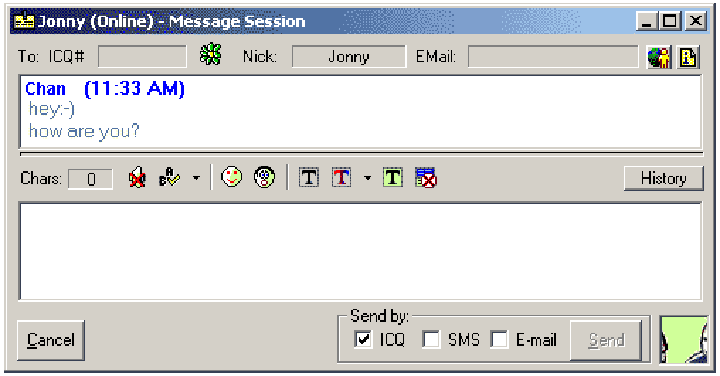
http://www.alvea.com/ehtml/know_screen_shot.htm

Bugzilla

Examples of Synchronous Groupware

- Instant messaging
- Multiuser video games
- Electronic meeting systems

ICQ Instant Messaging



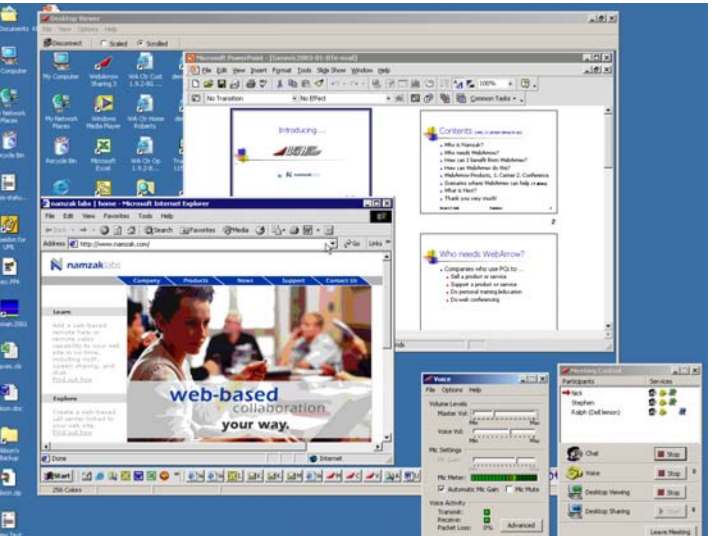
http://www.icq.com/icqtour/simple/user-menu.html

Dark Age of Camelot

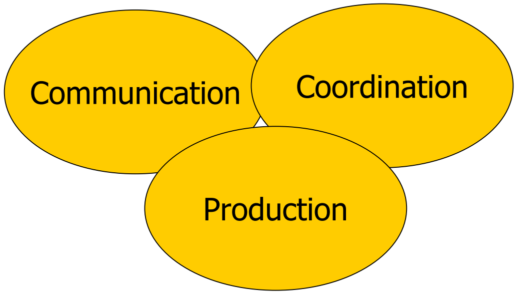
Multiuser Video Game

WebArrow/Conference

Electronic Meeting System

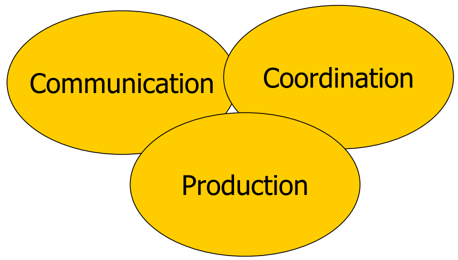


Clover Model of Communication [Calvary et al., 1997]



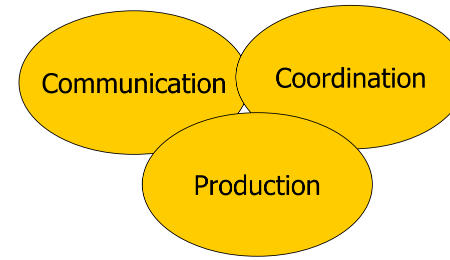
Clover Model

- People convey information to each other in three ways (termed *spaces*)
  - *Communication:* direct exchange of information, e.g., speech, gestures, facial expression, gaze information
  - Supported by (e.g.) telephone, videophone



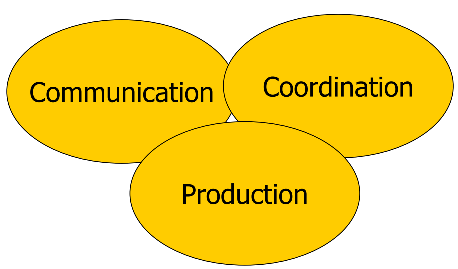
Clover Model

- *Production:* exchange of information through some artifact being collaboratively produced
- E.g., shared whiteboard, shared text editor
- Supported by (e.g.) MS NetMeeting, multiuser Video Annotator, Ultima Online



Clover Model

- *Coordination:* organization of how collaboration is to be enacted. E.g.,
  - In a chess game, use turn-taking
  - For patient's medical records, use workflow
  - For software development, use locking at class level
  - For collaborative drawing, use social protocols
- Supported by workflow systems, concurrency control algorithms

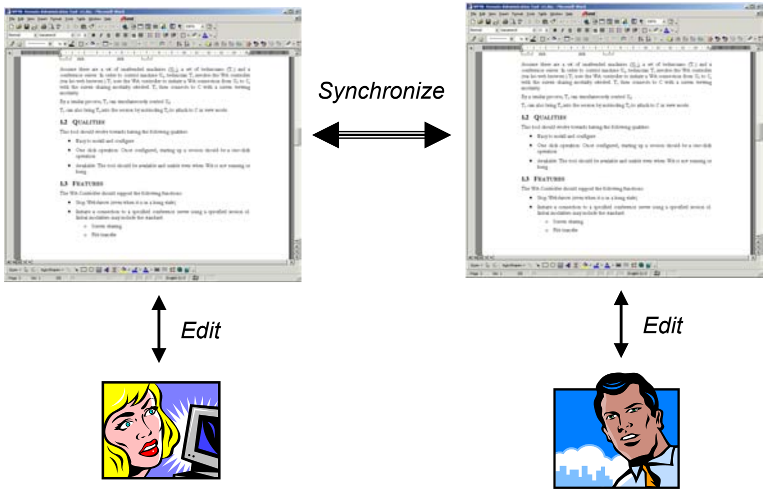


Concurrency Control

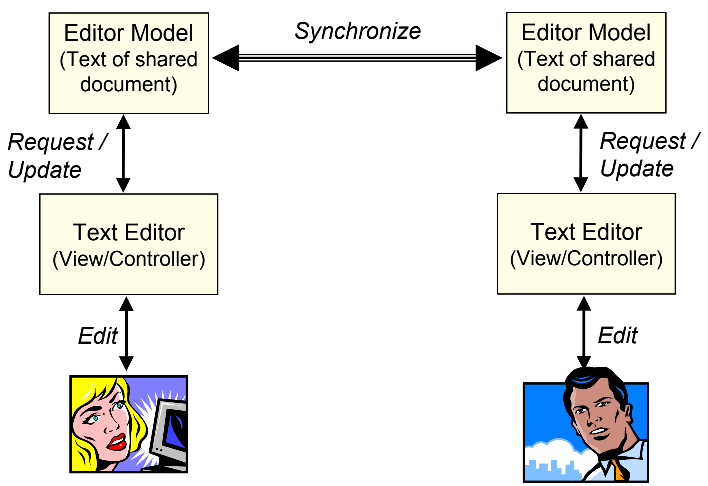
Problem

- Two people simultaneously update the same shared context
- Updates may conflict
- Concurrency control: algorithms that ensure:
  - All users' views of the shared context agree
  - In case of conflicts, result is intuitive

Example: Shared Text Editor



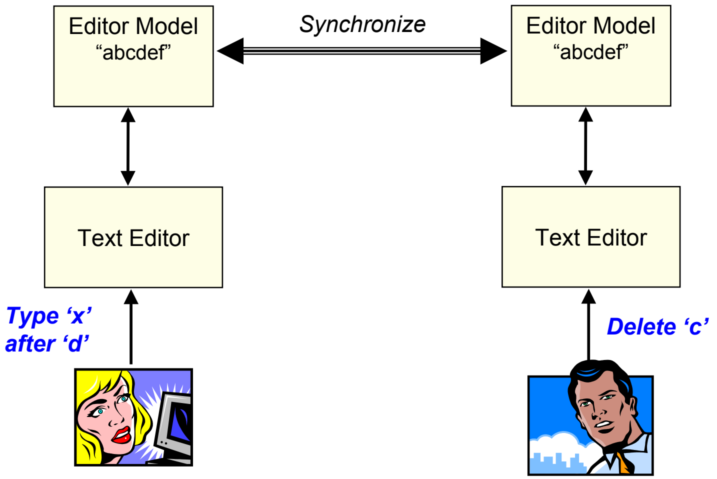
Example: Shared Text Editor



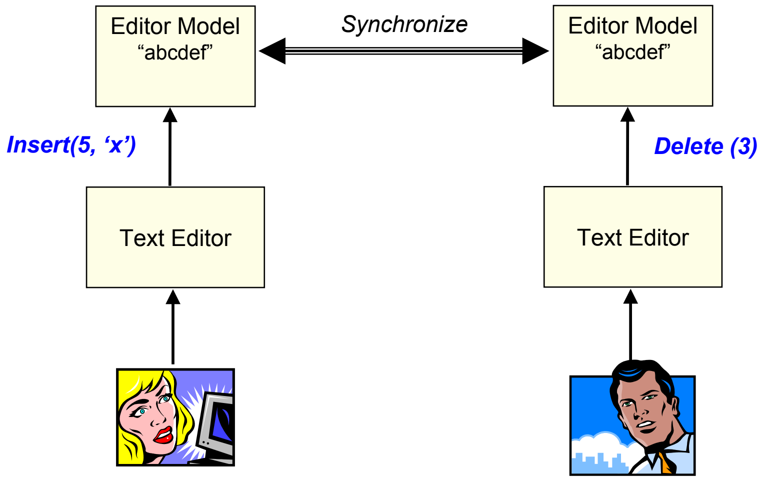
Editor Model

- Insert (int pos, char c)
  - Inserts *c* before position *pos,* where pos ≥ 1
- Delete (int pos)
  - Deletes character at position *pos*, where pos ≥ 1

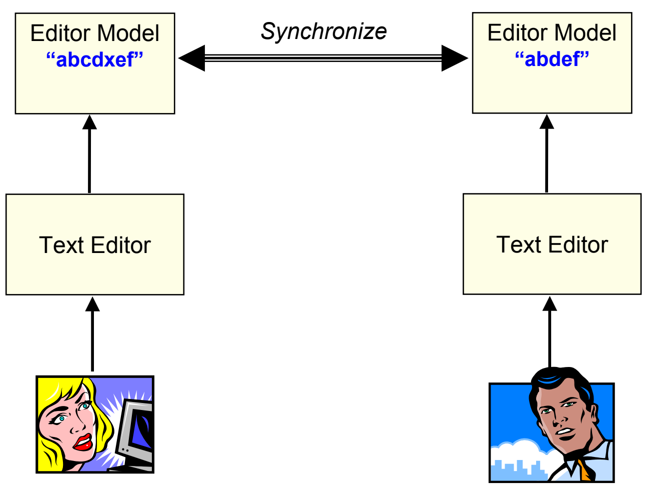
Example: Shared Text Editor



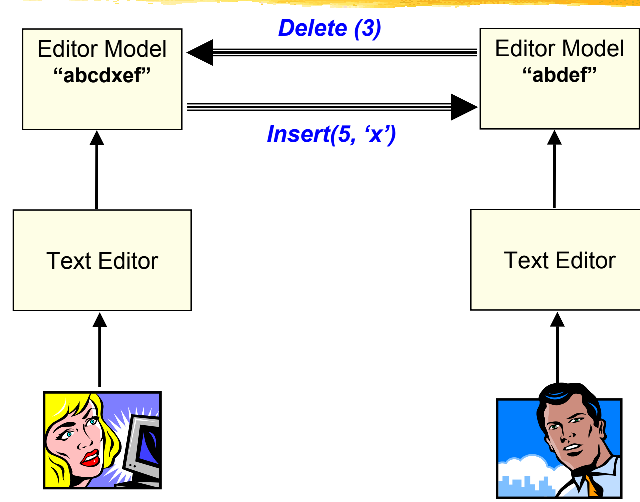
Example: Shared Text Editor



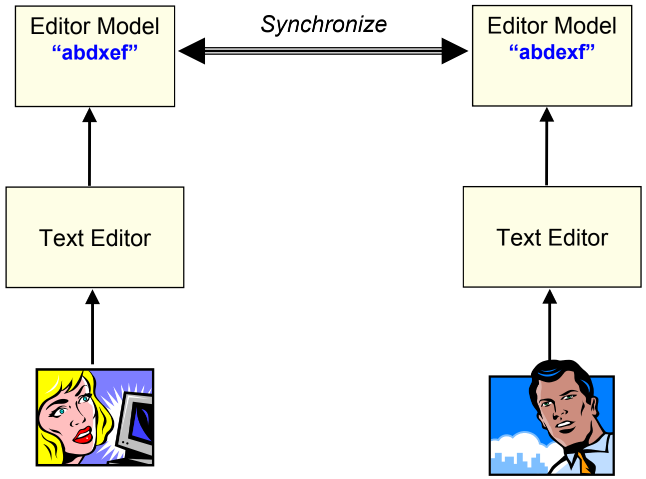
Example: Shared Text Editor



Example: Shared Text Editor



Example: Shared Text Editor



Problem

- Replicated data
- Performing operations in different order results in divergent values in text editor model
  - Users will see different documents on their screen

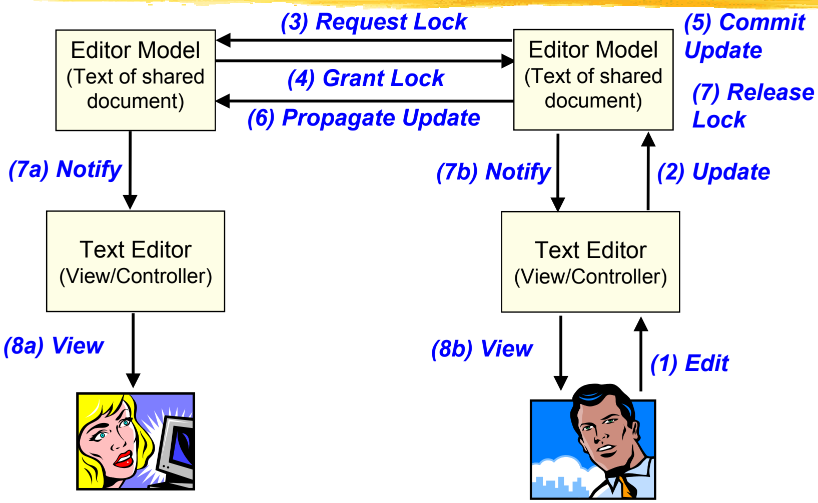
Goals of Concurrency Control

- Maintain consistent versions of models for all users
- Provide intuitive result when conflicts occur
  - E.g., could solve conflicts in text editor by setting document to "". Not an intuitive result!
- Maintain adequate feedthrough, feedback times
  - Many algorithms make feedback times unacceptable

Locking

- Before performing update to shared context, must obtain a lock
- Negotiated amongst all instances of shared context
  - If another user holds the lock, must wait until it is released
- Release lock when update has been propagated through system

Example: Locking



Locking

- Simple to implement
- Intuitive results
  - No conflicts
- Poor feedback time
  - Must go over network to obtain lock before user sees result of change
    - Bad for interactive tasks like text editing

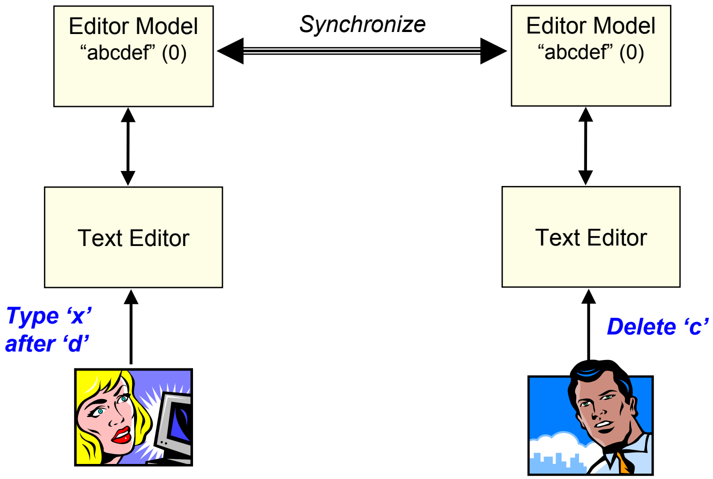
Rollback

- Detect conflicts via timestamp
- When conflicts occur
  - Roll back to earlier state
  - Reapply updates in canonical order
    - I.e., same order on each machine

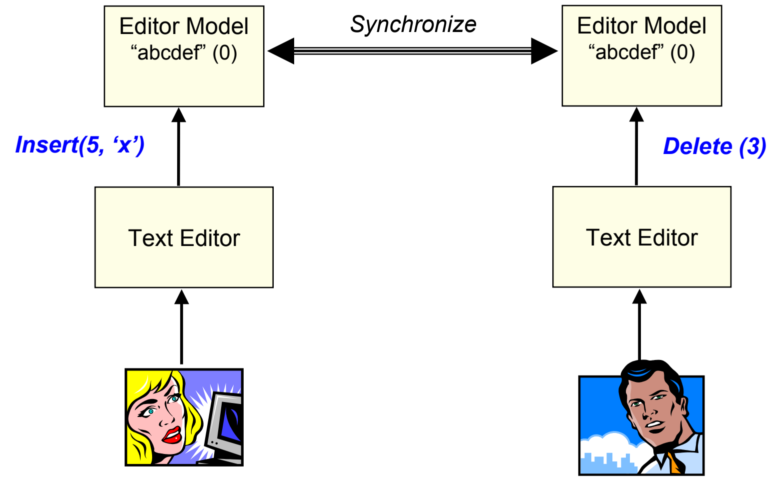
Timestamps

- Simply increment a counter every time an update is performed
  - Counter is timestamp
- Include timestamp in messages carrying remote updates
  - Specifies version of model against which update applied
- If remote update has older timestamp than model, a conflict has occurred

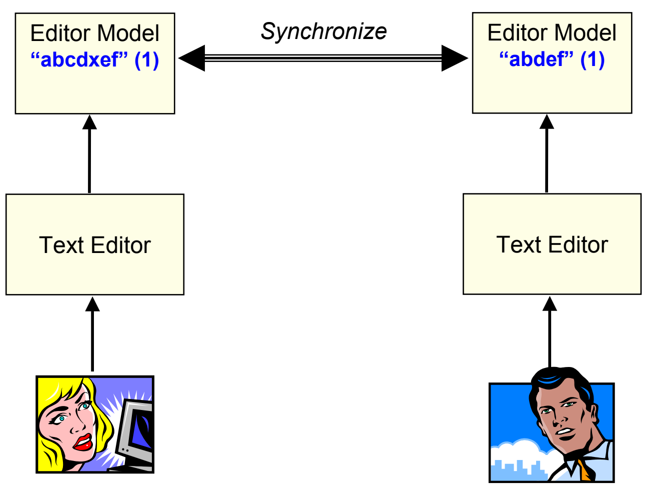
Example: Rollback



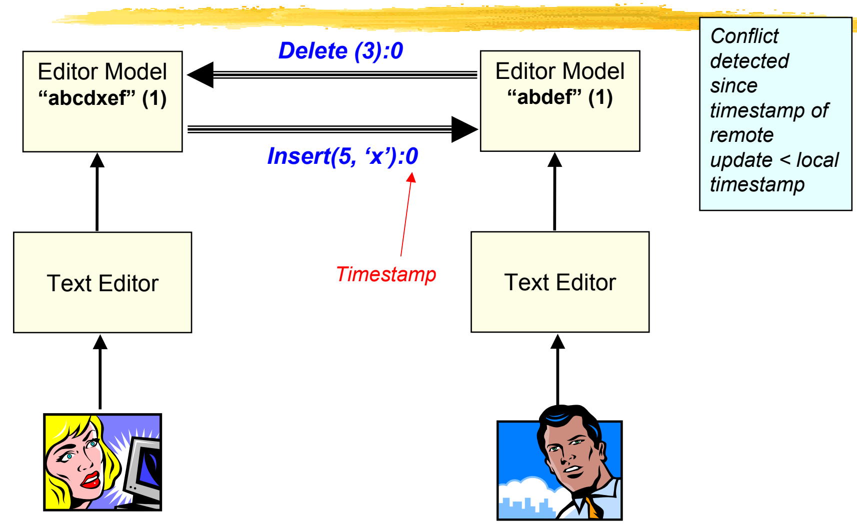
Example: Rollback



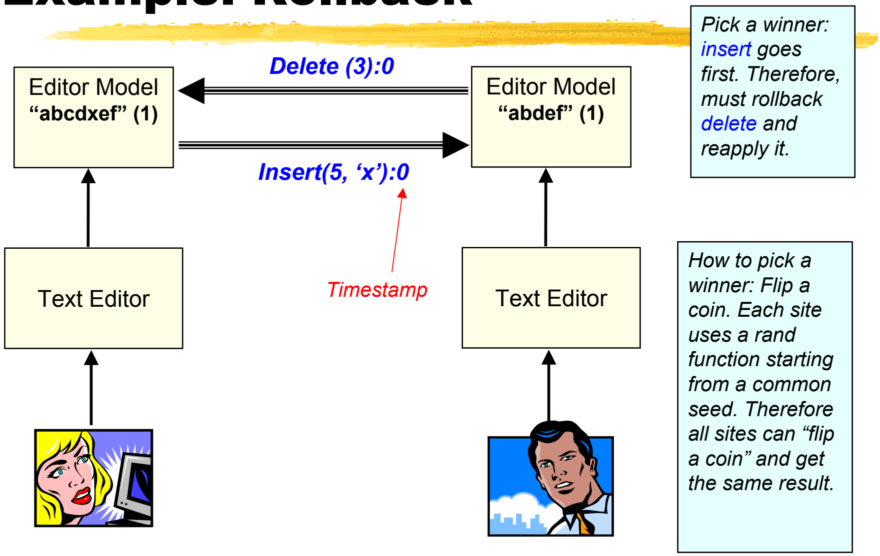
Example: Rollback



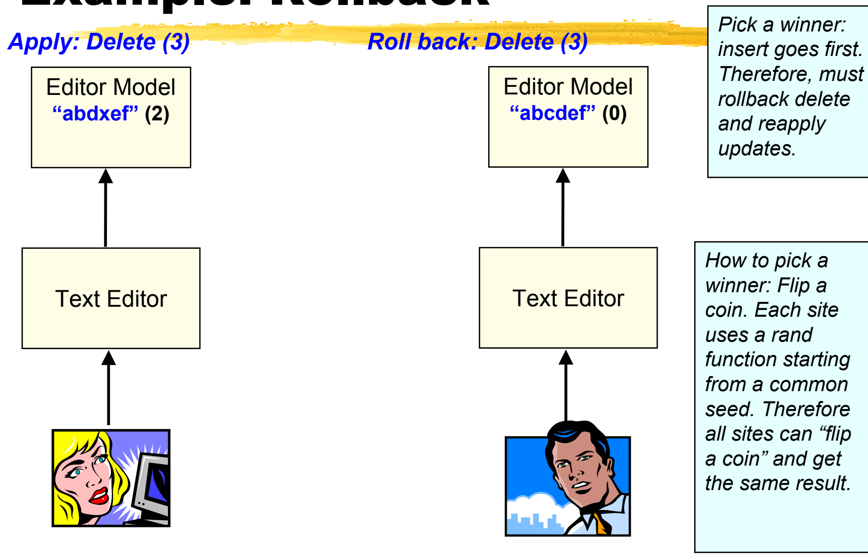
Example: Rollback



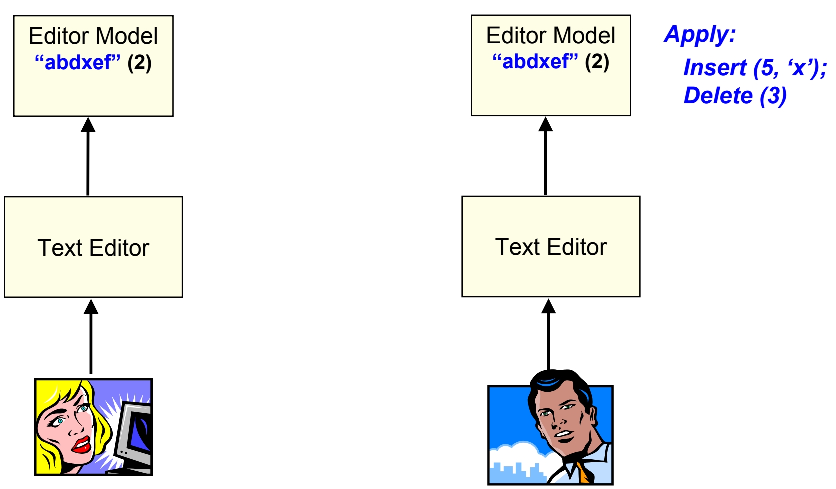
Example: Rollback



Example: Rollback



Example: Rollback



Rollback

- Immediate feedback
- Requires ability to roll back all operations
  - May be hard to program, costly in resources
  - Some operations can't be rolled back – e.g., file save, send email, "launch" button on nuclear missile
- Rollbacks may be confusing to user
  - See result of input, then it's changed later

Operation Transform

- Similar to rollback, except:
  - When conflict detected, instead of rolling back, *transform* operations
  - Requires *fixup* operations that undo and reapply all in one operation

Operation Transforms

- Example: Should have executed:
  - Insert (5, 'x') ; Delete (3) ***"abcdef" → "abdxef"***
- Actually executed:
  - Delete (3) ***"abdef"***
- Now want to execute:
  - Delete $(3)^{-1}$; Insert (5, 'x') ; Delete (3)
  - ≡ Insert (4, 'x') ; Delete $(3)^{-1}$; Delete (3)
  - ≡ Insert (4, 'x') ***"abdxef"***

Operation Transforms

- No need to roll back
  - ⇒No need to save earlier states
- Need to provide "fix up" operations for pairs of operations in model
  - n^2 operations
  - Some may be hard to implement
  - Some may not have intuitive fix ups

Concurrency Control

- Three methods:
  - Locking
  - Rollbacks
  - Operation Transform
- See next week's readings for more types

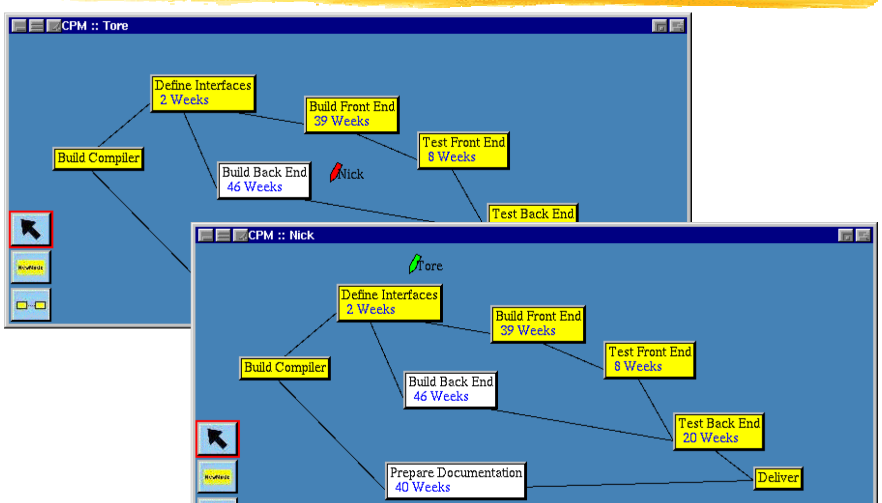
Groupware Architectures

- ⌘Clock
- ⌘Dragonfly
- ⌘…plus see next week's readings

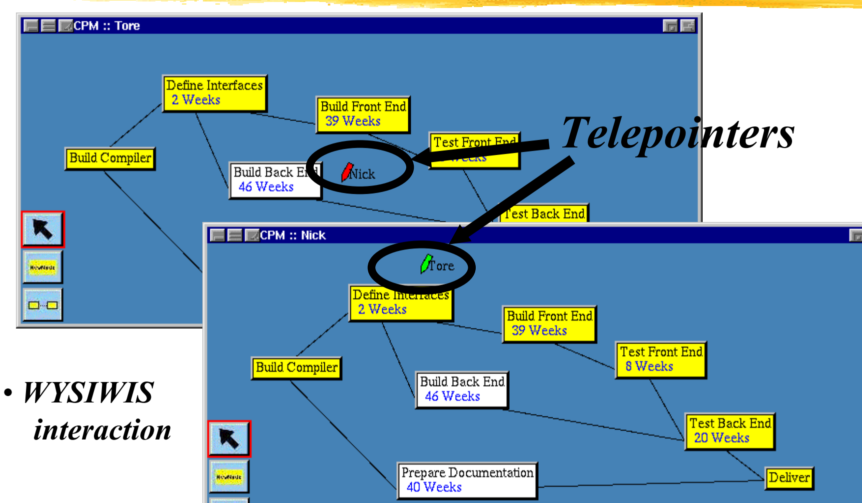
Outline

- ⌘Why is developing groupware hard?
- ⌘Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- ⌘Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- ⌘Timings

Synchronous Critical Path Scheduling Application

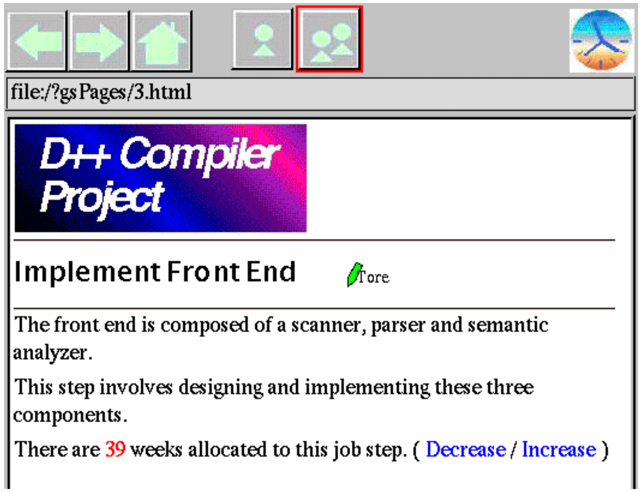


Synchronous Critical Path Scheduling Application

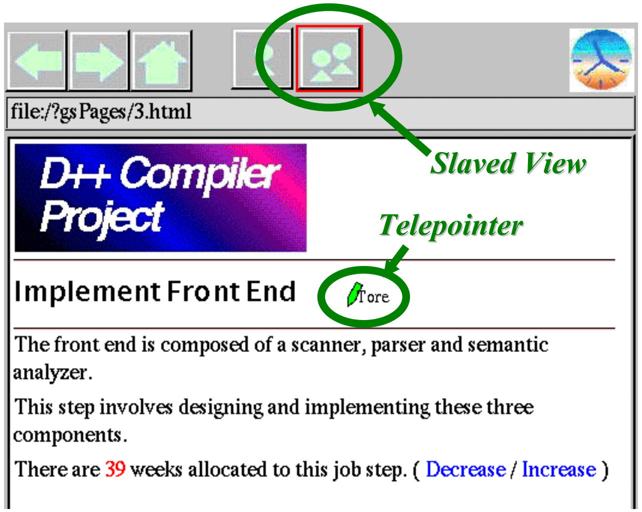


- *WYSIWIS interaction*

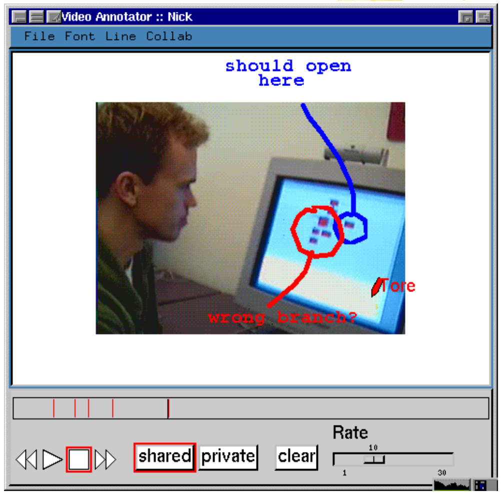
GroupScape Web Browser



GroupScape Web Browser



Synchronous Video Annotator



Two users

- Annotating tape of usability testing session
- Standard VCR controls
- Annotations saved with video frame

Why is Developing GW Hard?

- Synchronizing Relaxed WYSIWIS Views
- Concurrency
  - Race Conditions
- Distribution
  - Networking, Latency, Scalability, ...
- Multimedia
  - Synchronizing, Integrating MM Streams
- User-Centered Design

Outline

- Why is developing groupware hard?
- Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- Timings

Conceptual Architectures

High level architecture style hides:

- Distribution
- Networking
- Concurrency control

...allowing developer to concentrate on functionality of application

- E.g., Clock, PAC, ALV, C2, ...

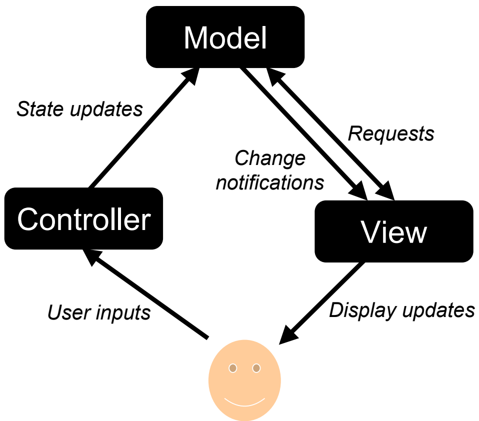
Clock Architecture Style

- Conceptual architecture that is also a program
- ClockWorks visual programming environment
- Connectors encode temporal properties
  - Concurrency control
  - Temporal media

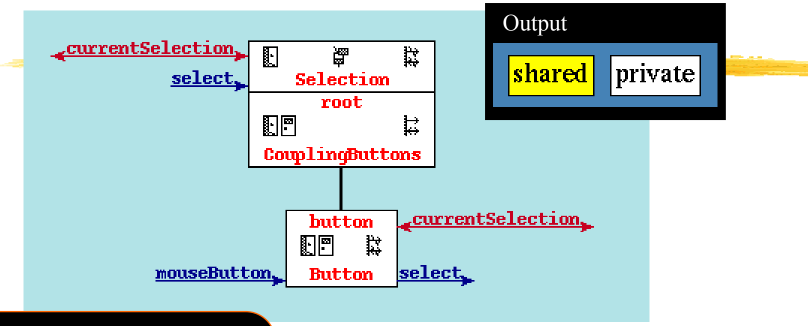
Clock Architecture Style

- Two views
  - Layered MVC
  - PAC where Control encoded by connectors

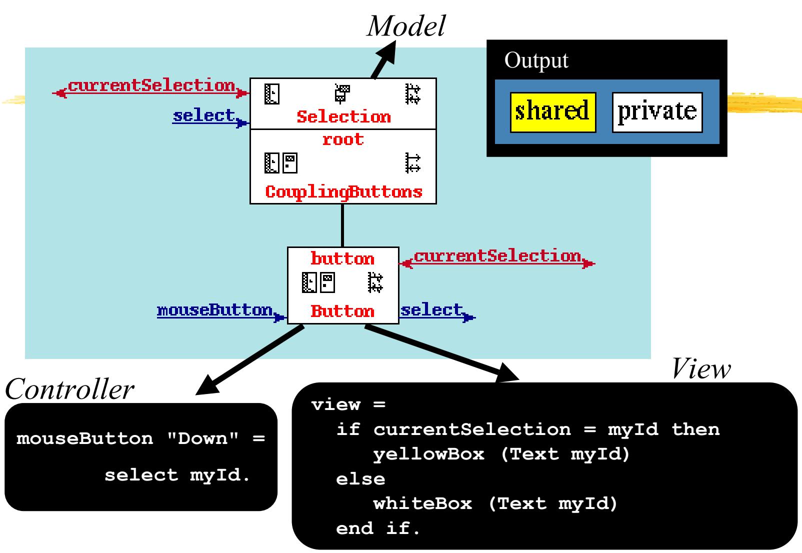
MVC Architecture

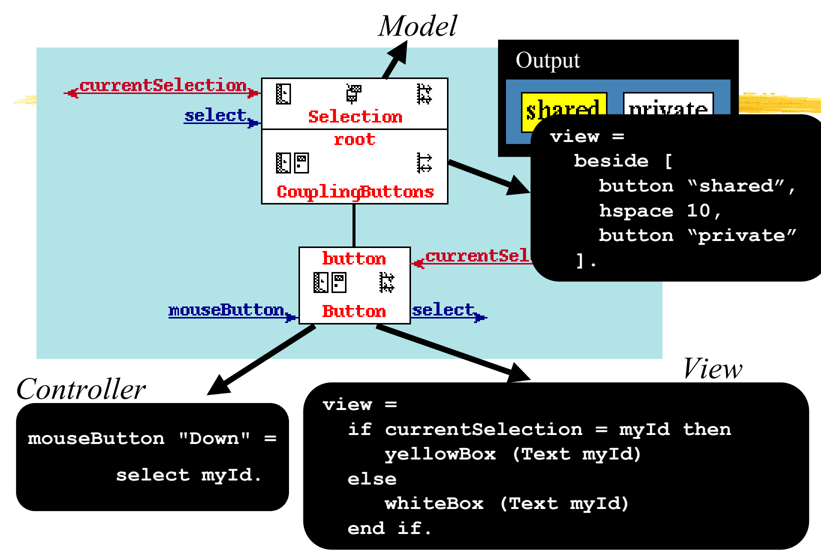


• Extended with layering, constraints, multimedia



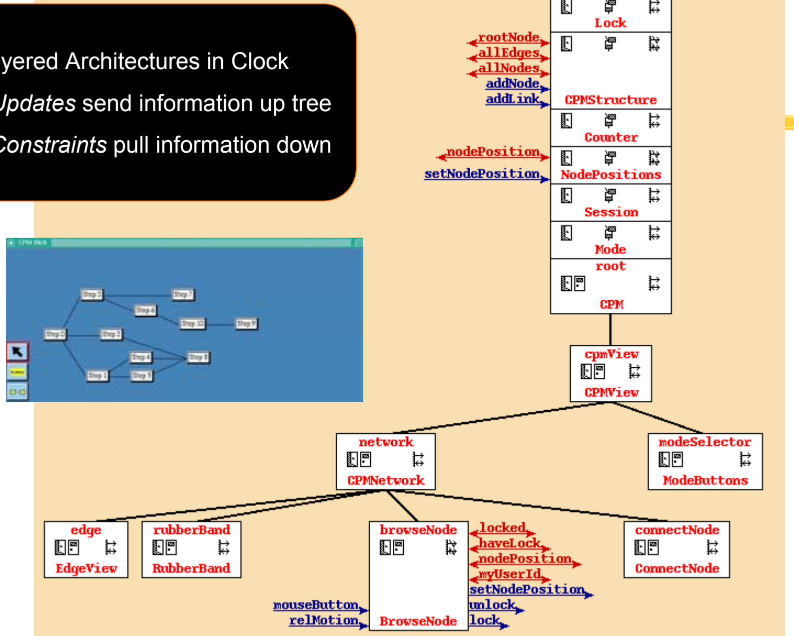
MVC in *Clock* architecture style





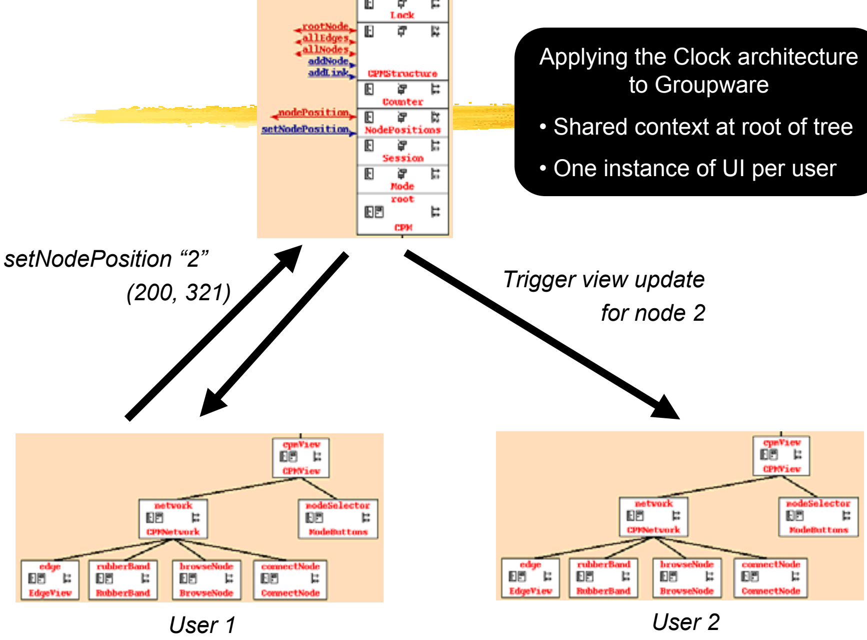
Layered Architectures in Clock

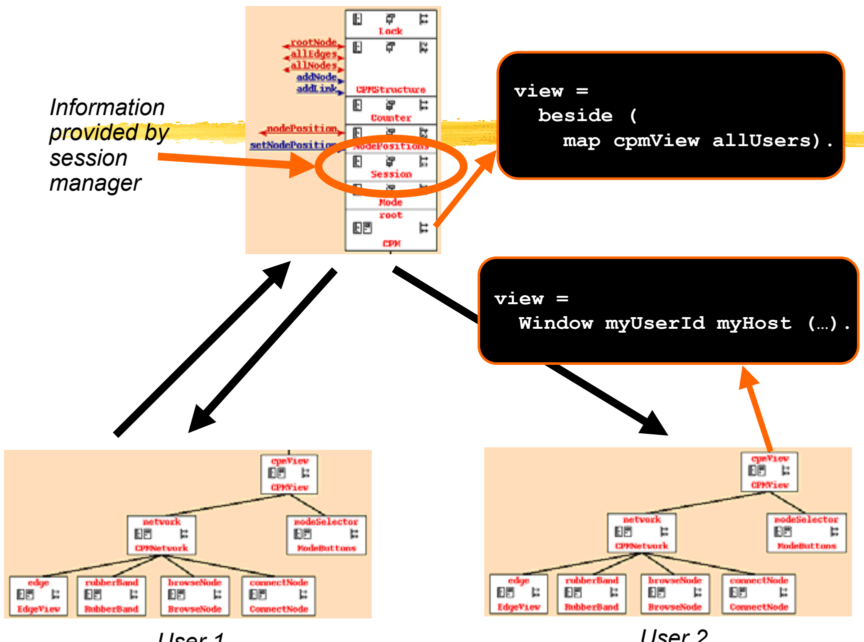
- *Updates* send information up tree
- *Constraints* pull information down



Applying the Clock architecture to Groupware

- Shared context at root of tree
- One instance of UI per user



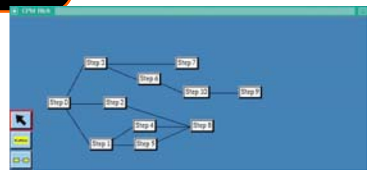


Tree Structure

- View composition
- Data visibility
- Concurrency control
  - Avoiding race conditions between user inputs
- Synchronization of multimedia streams
  - Sharing sound, video, animations

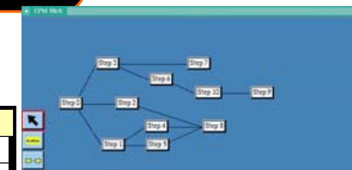
Concurrency Control

```
% Create a new node
mouseButton "Down" =
 all [addNode getCount mousePosition,
      incrementCount].
```



Concurrency Control

```
% Create a new node
mouseButton "Down" =
 all [addNode getCount mousePosition,
      incrementCount].
```

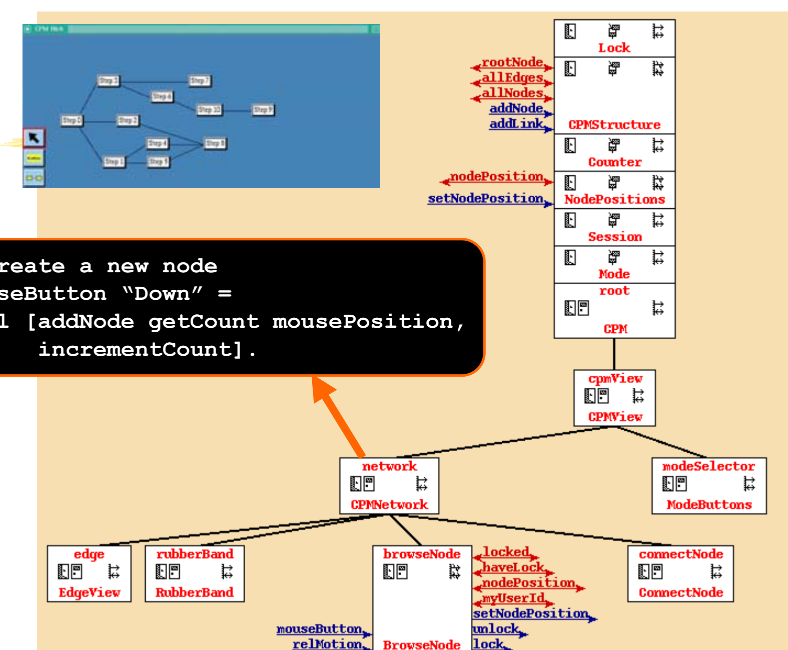


Possible execution order

User 1	User 2
getCount	
addNode (… ,…)	getCount
incrementCount	addNode (… ,…)
	incrementCount

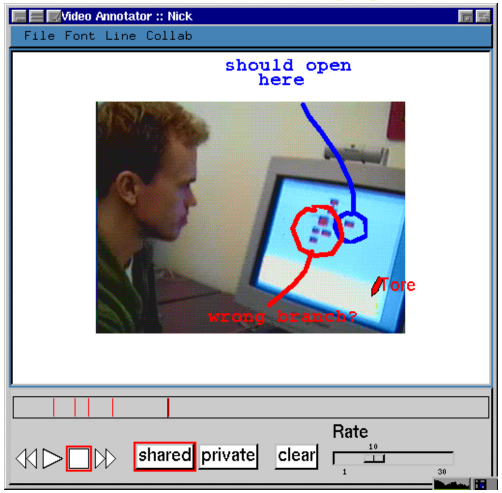
Concurrency control

- ⌘Each user's action leads to a *transaction*
  - request* update
- ⌘Transactions must be executed in a serializable fashion




```
% Create a new node
mouseButton "Down" =
 all [addNode getCount mousePosition,
      incrementCount].
```

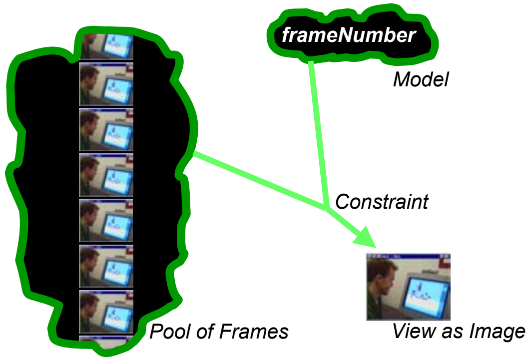
Multimedia Groupware

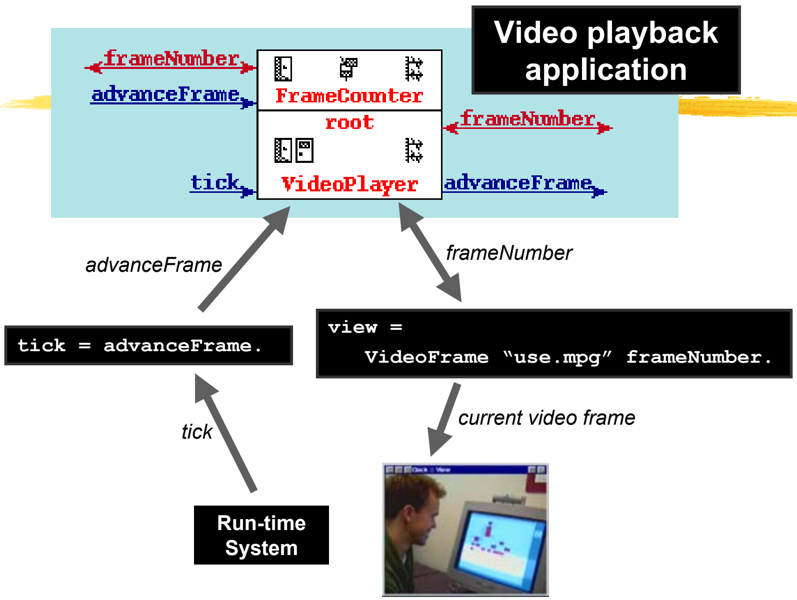


- Synchronization of multimedia streams
- Integration of temporal, static media

Extending MVC for Temporal Media

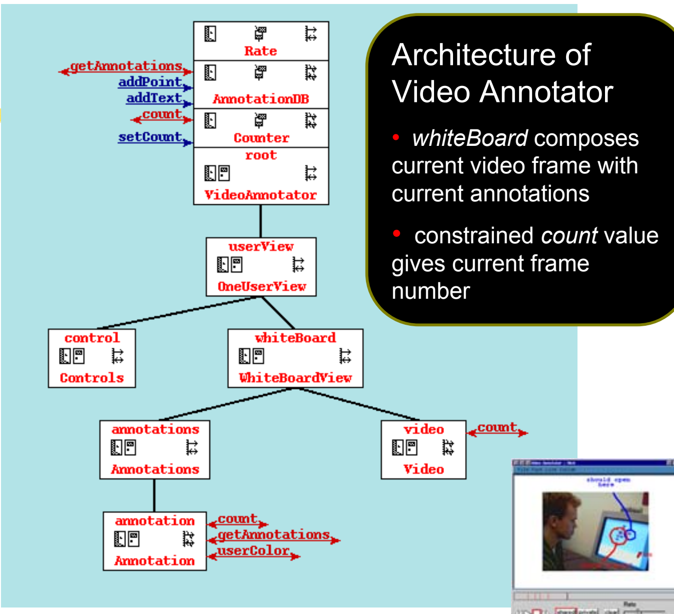
- *Constraints* remove time from temporal media
- *Composition* allows integration of temporal and static media





Architecture of Video Annotator

- *whiteBoard* composes current video frame with current annotations
- constrained *count* value gives current frame number



Clock Architecture Style: Summary

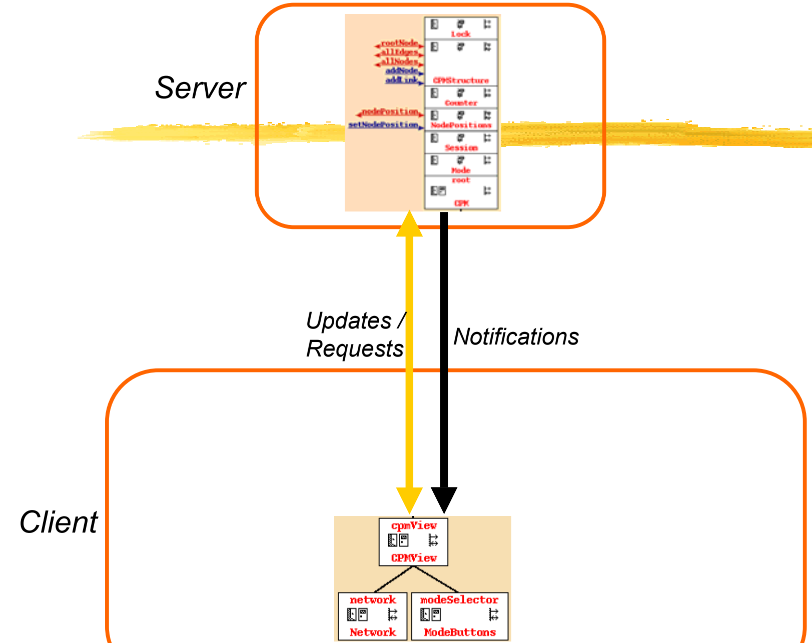
- Conceptual architecture is part of program
- Constraints help in:
  - Synchronizing relaxed WYSIWIS views
  - Multimedia
- Concurrency control guaranteed by connectors
- Visual editor for architectures

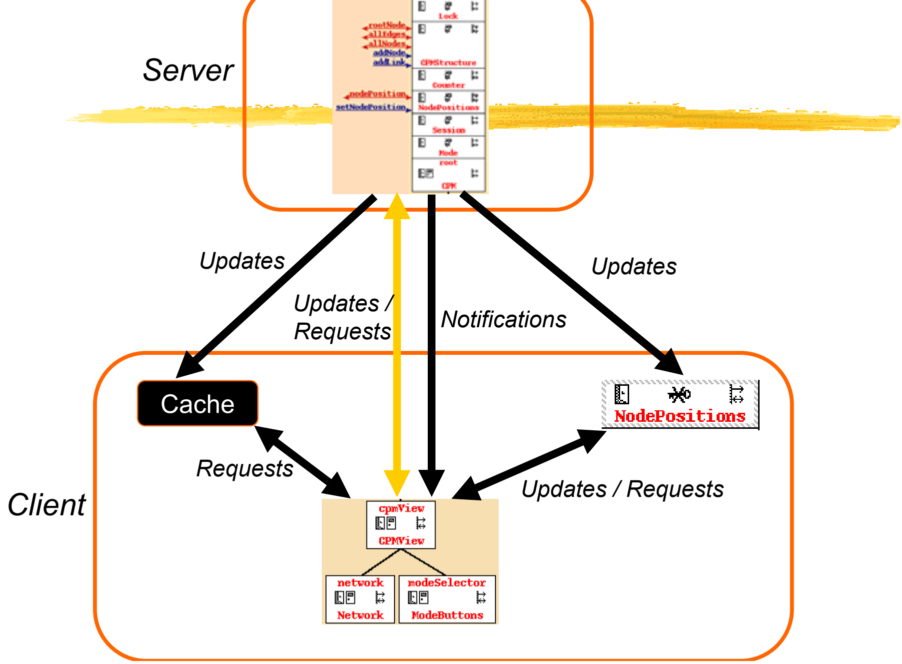
Outline

- Why is developing groupware hard?
- Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- Timings

Strategies for Cheating Latency

- Sacrifice group response time
- Reduce granularity of group actions
- Take turns
- Risk roll-backs
- Buffer
- Spend CPU, memory, bandwidth





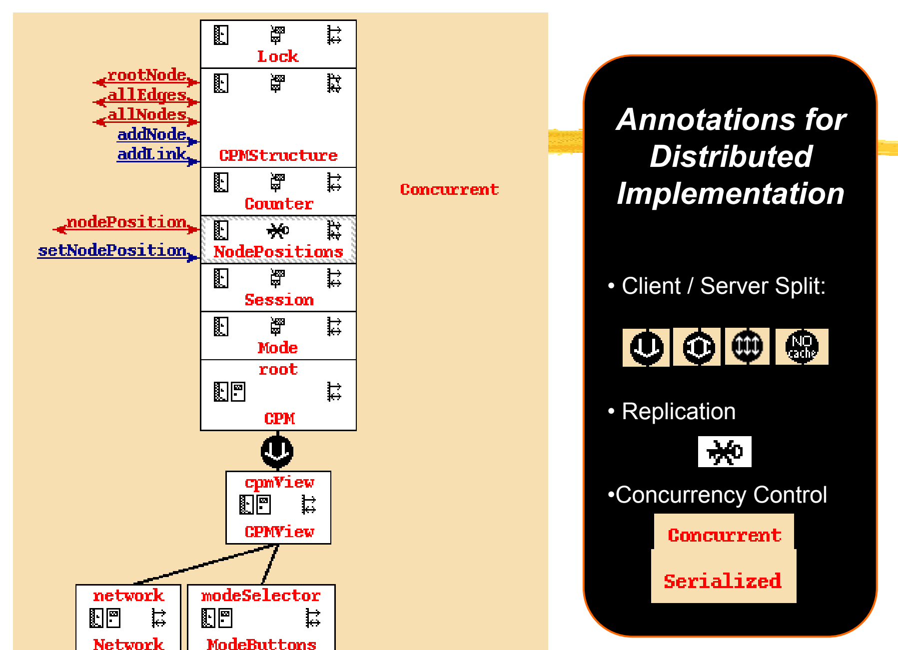
Space of Implementation Techniques

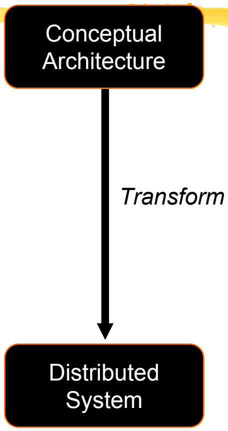
- Concurrency control: *none, optimistic, pessimistic*
- Topology: *tree, star, mesh, point*
- Replication: *centralized, partially replicated, replicated*
- Cache activity: *passive, active*
- Cache location: *standard, mirror*
- Network delivery: *lossless, lossy*
- Network order: *FIFO, any*

What Strategy is Best?

Depends on:

- Application characteristics
  - Symmetry, roles, synchronicity
- Hardware characteristics
  - How cheap is bandwidth, latency, CPU, memory?

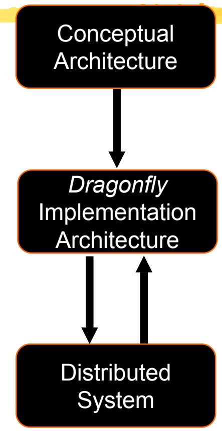




Hide issues of:
- **Concurrency control**
- **Distribution**
- **Networking**

E.g. Clock, PAC*, C2, ALV

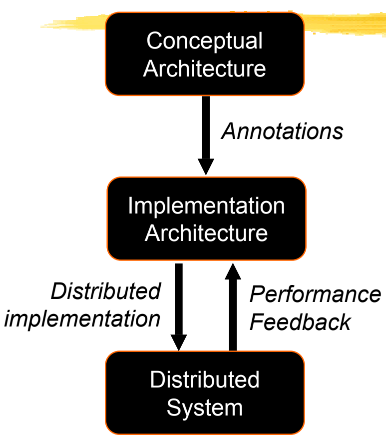
Choose:
- **Centralized or Replicated?**
- **Pessimistic or Optimistic?**
- **Caching?**
- **Hybrid?**



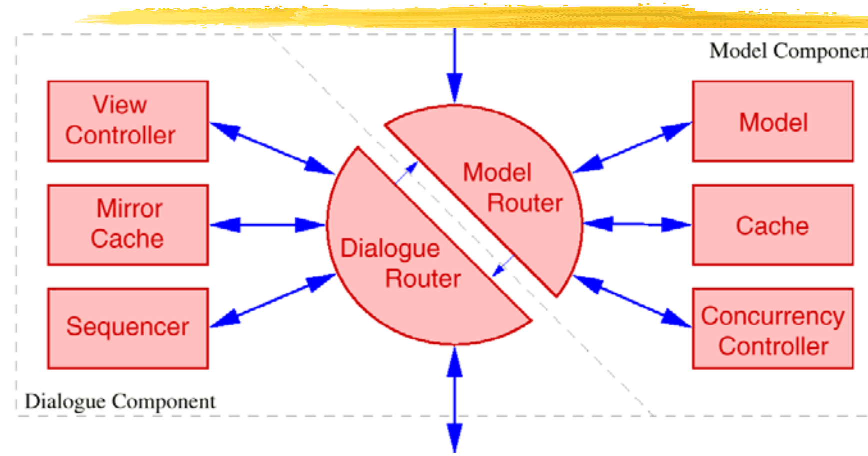
Developers concentrate of functionality of groupware application, not distributed implementation

Encapsulates design choices for:
- **Replication**
- **Caching**
- **Concurrency Control**
- **Client-Server Split**

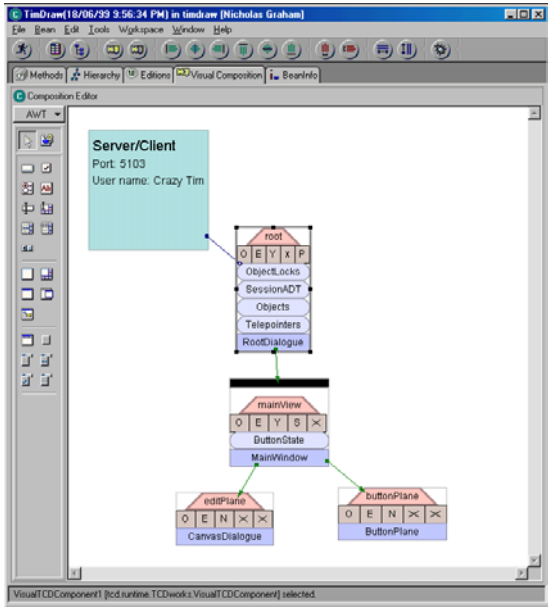
- **Hybrid implementation**
- **Dynamic reconfiguration**

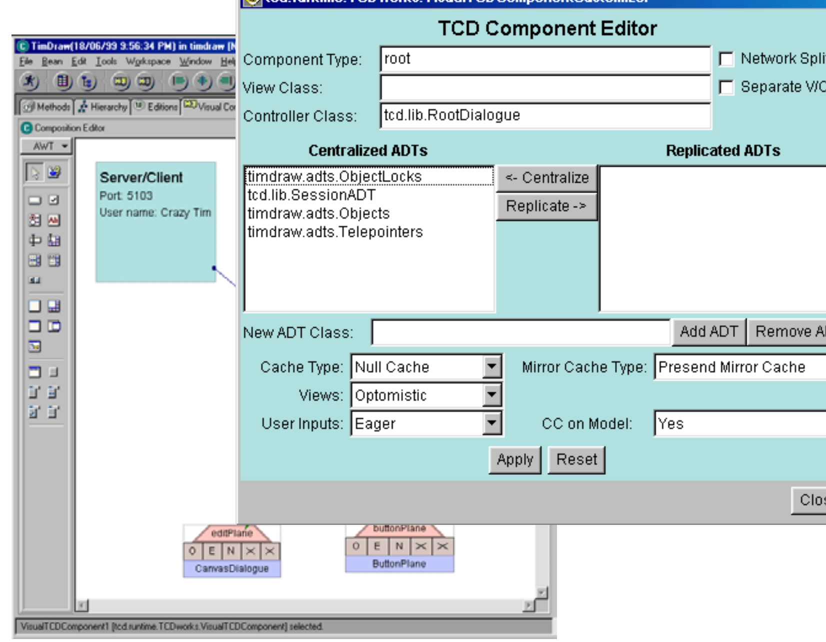


The Dragonfly Component



[Anderson et al 00]





Outline

- Why is developing groupware hard?
- Programming abstractions for groupware
  - Declarative development of groupware
  - Conceptual architectures as programs
- Flexible implementation of groupware
  - Annotations for implementation
  - The Dragonfly implementation architecture
- Timings

Timing Clock and Dragonfly

Response time where:

- *Server: LAN: UltraSparc 1, York U.; WAN: 100 MHz R4000 SGI Indy, Queen's University*
- *Client: UltraSparc 1, York U.*
- *Network Latency: LAN: 1 ms; WAN: 11 ms*

Conclusions

- Large gap between design-level architecture and distributed system
- Dragonfly acts as layer between, allowing experimentation with implementation strategies
- Programmers can give high-level guidance for implementation