

Visuelle Transaktionen: Kontrolle der Nebenläufigkeit bei synchroner Gruppenarbeit

Lutz Wegner
FB Mathematik/Informatik
Universität Gh Kassel

Klassifikation der Gruppenarbeit am Beispiel computer-unterstützter Unterricht (CUU)

CUU als Spezialform des CSCW (Computer Supported Cooperative Work, Gruppenarbeit, Groupware, ...), Taxonomie nach Raum-/Zeitachse [Ellis, Gibbs und Rein in CACM 91]

| | | Ort | |
|------|------------|--|--|
| | | zentral | verteilt |
| Zeit | selbe | Lerntheater, reales Seminar mit MM-Einsatz | virtuelles Seminar synchrone Zusammenarb. |
| | verschied. | Computer Lernlabor Internet Café, ... | „anytime anyplace“ asynchrone Netzwerke |

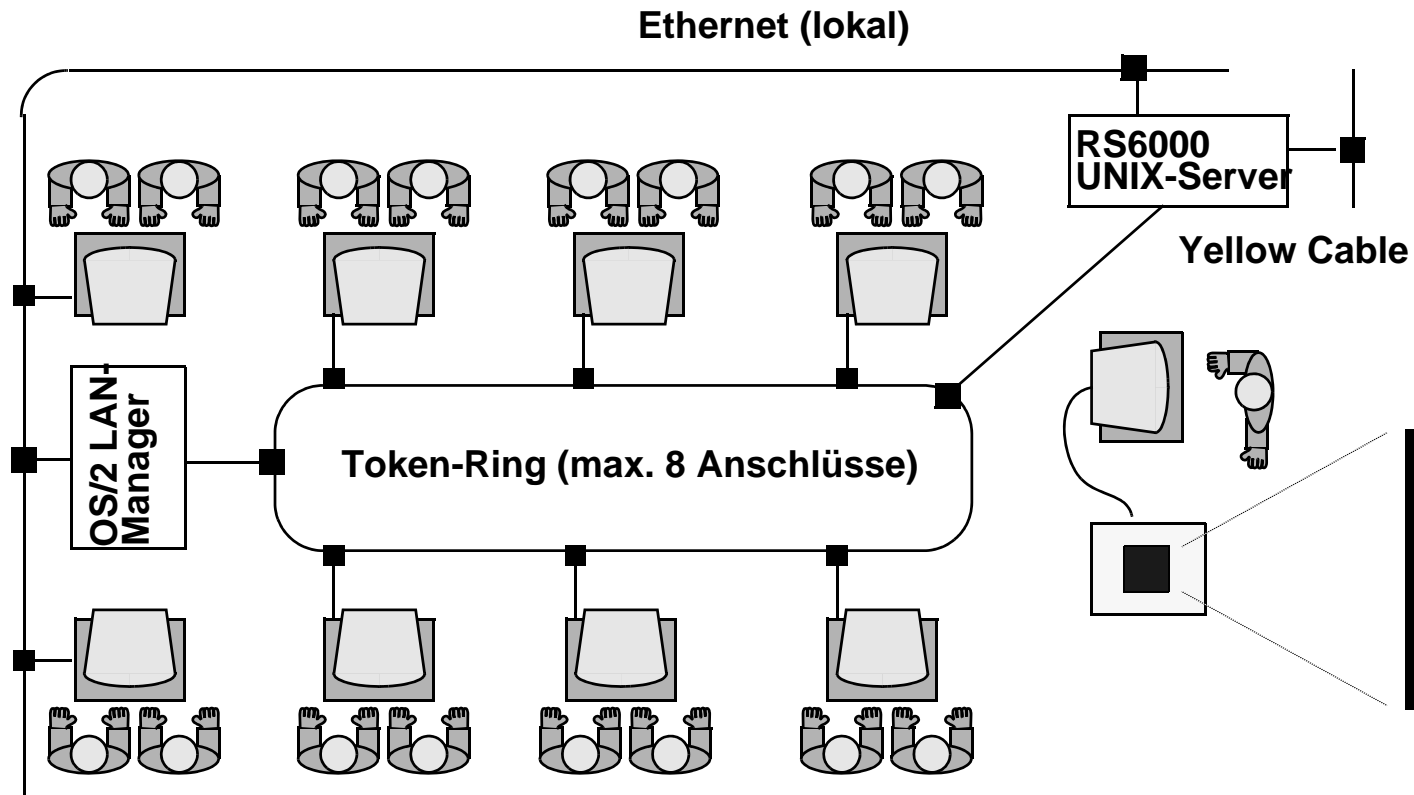
Bekannte Anwendungsszenarien für CSCW

- **Gemeinsames Editieren eines Dokuments**
gemeinsames Formulieren von Textpassagen, Einbringen eigener Textstücke, Schlußredaktion
- **Belegen von Veranstaltungen an der Uni**
An- und Abmelden in Veranstaltungen, Raum- und Zeitänderungen durch Dozenten
- **computergestütztes Lernen**
bei synchronen Formen Eintritt und Austritt aus virtuellen Seminaren (join and leave session), Moderation einer Diskussion, Kontrolle des Mikrophons (floor control)
- **Auktionen, Handel, Makeln**
- **Krisenmanagement**
- **Bauplanung, gemeinsames Konstruieren (CAD)**
- **Team Science und Tele-Medizin [PSL99]**
- **Unterstützung in einem Call Center (next generation call centers)**

Existierende Produkte

- Lotus Notes
- Microsoft NetMeeting
- CoolTalk im Netscape Communicator

Beispiel eines Lernlabors für synchronen CUU



Synchrone, räumlich verteilte Gruppenarbeit

- erscheint als natürlicher Schritt nach Batch- und Dialogverarbeitung
- Vorstufe zur Einbettung des Menschen als peripheres, notorisch unzuverlässiges Gerät ins globale Netz
- alter Traum, vgl. „Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings“ [Stefik, Foster et al. in CACM 87]

Unterschiedliche Vorstellungen über Ausgestaltung

- Virtuelle Seminarräume und Videokonferenzen, z.B. GMD DOLPHIN [Mark, Haake, Streitz auf der CSCW 96]
- Desktop Konferenzsysteme, z.B. Xerox Colab [siehe oben 1987] und Rendevouz [Patterson, Hill, Rohall, Meeks auf der CSCW 90]
- Bulletin boards und chat rooms
- Transparente Whiteboards, z.B. TeamWorkStation [Ishii, Kobayashi, Arita in CACM 94]
- Telepointer
- VRML Umgebungen

CSCW in Form von Videokonferenzen etc.

☞ nützlich, aber im wörtlichen Sinn oberflächlich

Artifakte der Diskussion liegen als rechnergestützte, strukturierte Objekte vor!

- Stundenplanung → Belegungspläne, Stundenpläne, Anmeldungen
- Bauplanung → Bauzeichnung, Statik, Raumgrößentabelle
- Projektmanagement → Gantt-Diagramm, Abhängigkeitsgraph, Ressourcenplan
- Unterricht → Formeln, Graphen, Algorithmen, Texte, Bilder, geometrische Objekte

☞ Eintauchen in Objekträume, Navigation, Objektmanipulation, Datenbankzugriff!

multiple shared data artifacts [PSL99]

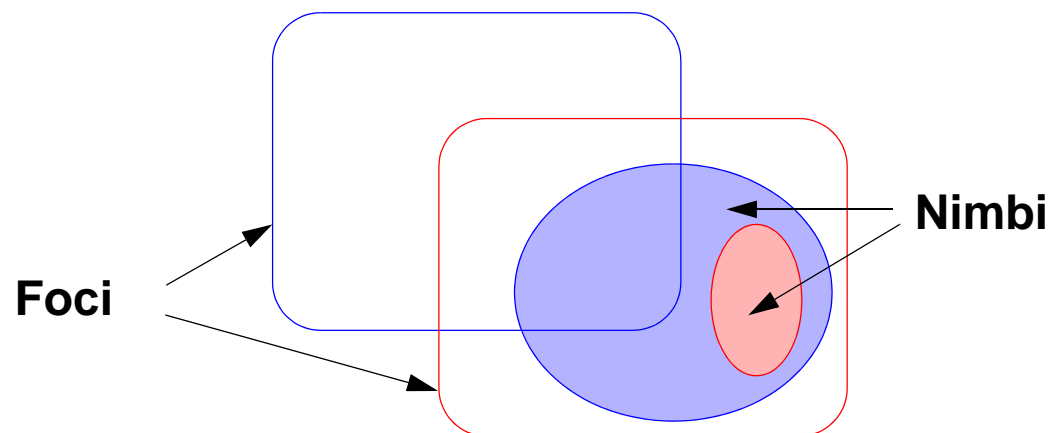
- GroupKit, TeamWave [Greenberg and Roseman, <http://www.teamwave.com/>]
- ESCHER/TclDB Entwicklung in Kassel,
Anwendung in CSCW Lernumgebungen [Krämer, Wegner auf der ED-MEDIA98]
- ...

Elemente der Gruppenarbeit

☞ Räumliche Kopräsenz (concurrency awareness, group awareness)

Überlegungen zu einem Präsenzmodell durch Tom Rodden et al., Lancaster University [CSCW96]

- *Fokus* (wohin die Aufmerksamkeit gerichtet ist, was im Licht steht)
- *Nimbus* (wo man agiert, worauf man Einfluß nimmt, was andere von einem wahrnehmen)



Deictische Strategien, deictisches Handeln

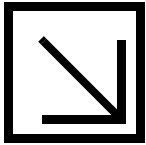
- Phänomene sind aus den Verhaltenswissenschaften bekannt,
vgl. z.B. *Dana H. Ballard, Mary M. Hayhoe, Polly K. Pook, and Rajesh P.N. Rao:*
Deictic Codes for the Embodiment of Cognition, *Behavioral and Brain Sciences* 20(4) 1997
- Deictisches Handeln unterstützt den Prozeß des Verstehens des Menschen und ist zwischen neuronalen Prozessen und höheren Lernprozessen angeordnet. Das Handeln besteht darin, daß der menschliche Körper im Abstand von ca. 1/3 s durch Bewegungen (Hand, Auge, Kopf, Mund) sensorische Eingaben mit Körperbewegungen verbindet und dadurch sein Kurzzeitgedächtnis steuert.
- Die Bewegungen unterstützen das Binden (Speichern, Setzen) von Variablen mit Wahrnehmungen im menschlichen Arbeitsspeicher (working memory). Diese Variablen werden befragt zur Abarbeitung eines Verstehensprozesses. Das Kurzfristgedächtnis hat Verfallszeiten von einigen Sekunden.
- Das (räumliche) Fixieren von Objekten mit dem Auge spielt dabei eine große Rolle. Die Neigung dorthin zu zeigen, wohin man schaut, ist ein Ausdruck des Phänomens (genauso, dorthin zu schauen, wo man handelt; kann man sich abtrainieren, vgl. blindes Schreiben auf der Tastatur).



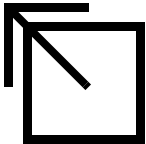
Das Fingerkonzept

- Seit ca. 1987 in Kassel Entwicklung des Datenbankeditors ESCHER für erweiterte relationale Datenbanken
- Navigierender Ansatz
- Primäre Darstellung durch geschachtelte Tabellen
- Navigation und Manipulation von Objekten durch einen Cursor der *Finger* heißt
- Finger eines Benutzers = Nimbus
sichtbarer Ausschnitt (viewport) eines Benutzers = Fokus
- Nebenläufigkeit von Fingern → CSCW, synchrone Gruppenarbeit
- Auch realisierbar im Web als Client-Server Lösung über sog. Tclets, dazu eine Skriptsprache TclDB als Erweiterung von Tcl/Tk [Ousterhout]

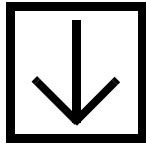
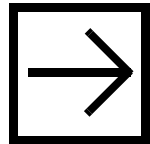
Universelle 4-Tasten Navigation



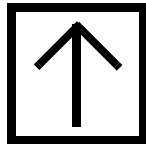
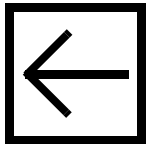
In, Enter, OK, Push, Mouse Click



Out, Escape, Cancel, Pop, Mouse Drag



**Next, Succ, Arrow Keys
Mouse Click**

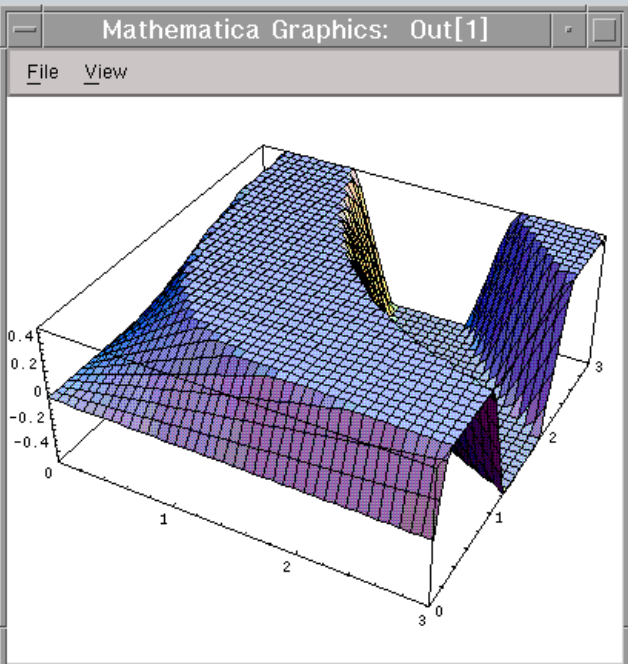


**Back, Pred, Arrow Keys
Mouse Click**

Beispiel

Navigation in Mathematica Daten

Mathematica Graphics: Out[1]



xescher

System Schema Table Application

Application: wegner

TABLE: DEMOPLOTS.tbl / SCHEMA: PLOT3D

File Edit Fingers Query Sorting Options

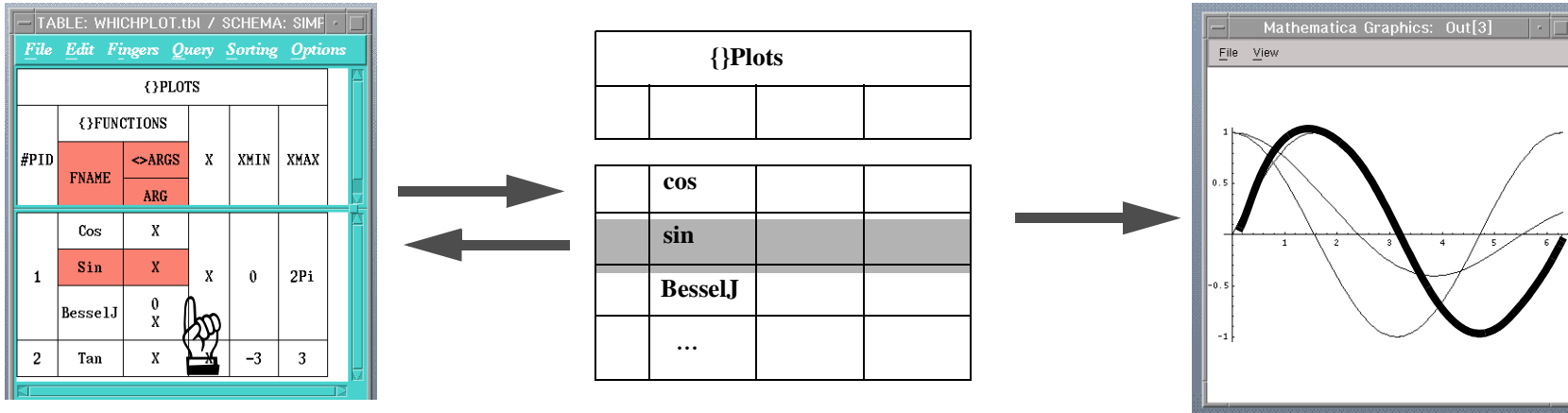
{ }PLOT3D

| #PID | FNAME | ◁ARGS | { }RANGES | | | { }STYLES |
|------|---------|--------------|-----------|----------|--------|----------------|
| | | | ARG | VAR | FROM | TO |
| 1 | Sin | x y | x y | 0 0 | 3 3 | PRhalf PP40 |
| 2 | BesselJ | 3x nu | nu x | 0 0 | 3 3 | PP40 |
| 3 | Exp | $-(x^2+y^2)$ | x y | -2 -2 | 2 2 | LT VPabove |
| 4 | Cos | x y | x y | 0 0 | 3 3 | PRhalf |

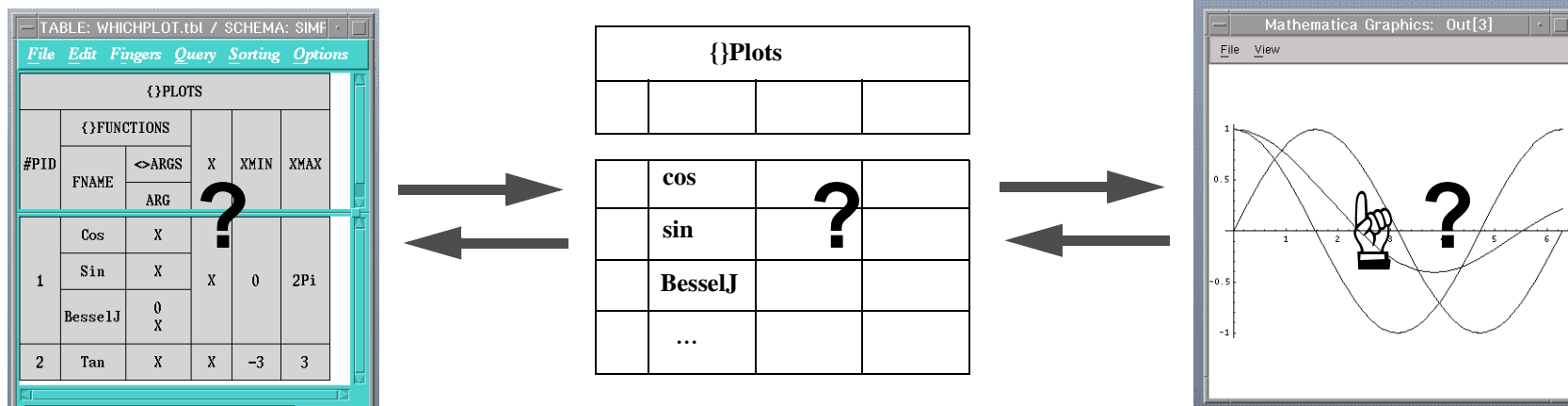
Null

Mode: Browse Active Finger: **F1**

Visualisierung: Offene versus geschlossene Objekträume



(a) Fingerposition kommt vom DB-Editor



(b) Inverse Abbildung: Finger aus Graphanzeige



Beispiel: Kooperatives Projektmanagement

netscape: ESCHERS Tcl/Tk interface

File Edit View Go Communicator Help

Back Forward Reload Home Search Guide Print Security Stop

Bookmarks Location: <http://www.db.informatik.uni-kassel.de/~escher/tcldbTclt/escherClient.html>

Internet Lookup New&Cool

exit path application table

| {} TASKS | | | | | | | | |
|----------------|------------------------|-----------|-----------|-----------|-----------|-----------|------------|-----------------------------------|
| TASKID | DESCRIPTION | DUR | ES | LS | EF | LF | ISOTIME | { } REQUIRES TASK |
| START | project kickoff | 0 | 0 | 0 | 0 | 0 | 01-03-1999 | { } |
| REQ | requirement analysis | 2 | 0 | 0 | 2 | 2 | \? | START |
| STAFF-PREP | staff preparation | 4 | 0 | 2 | 4 | 6 | \? | START |
| GSPEC | global specification | 4 | 2 | 2 | 6 | 6 | \? | REQ |
| GDESIGN | global design | 5 | 6 | 9 | 11 | 14 | \? | GSPEC STAFF-PREP |
| FSPEC | fine specification | 8 | 6 | 6 | 14 | 14 | \? | GSPEC |
| SPEC-DOC | specification document | 0 | 14 | 14 | 14 | 14 | 11-06-1999 | FSPEC |
| FDESIGN | fine design | 10 | 14 | 14 | 24 | 24 | \? | SPEC-DOC GDESIGN |
| CODE+UTEST | coding and unit test | 52 | 24 | 24 | 76 | 76 | \? | FDESIGN SPEC-DOC |
| INTEGRATE | integration test | 24 | 76 | 76 | 100 | 100 | \? | CODE+UTEST |
| ALPHA | alpha release | 0 | 100 | 100 | 100 | 100 | 02-01-2001 | INTEGRATE |
| ACCEPT | acceptance test | 7 | 100 | 100 | 107 | 107 | \? | ALPHA |
| END | project close down | 0 | 107 | 107 | 107 | 107 | \? | ACCEPT |

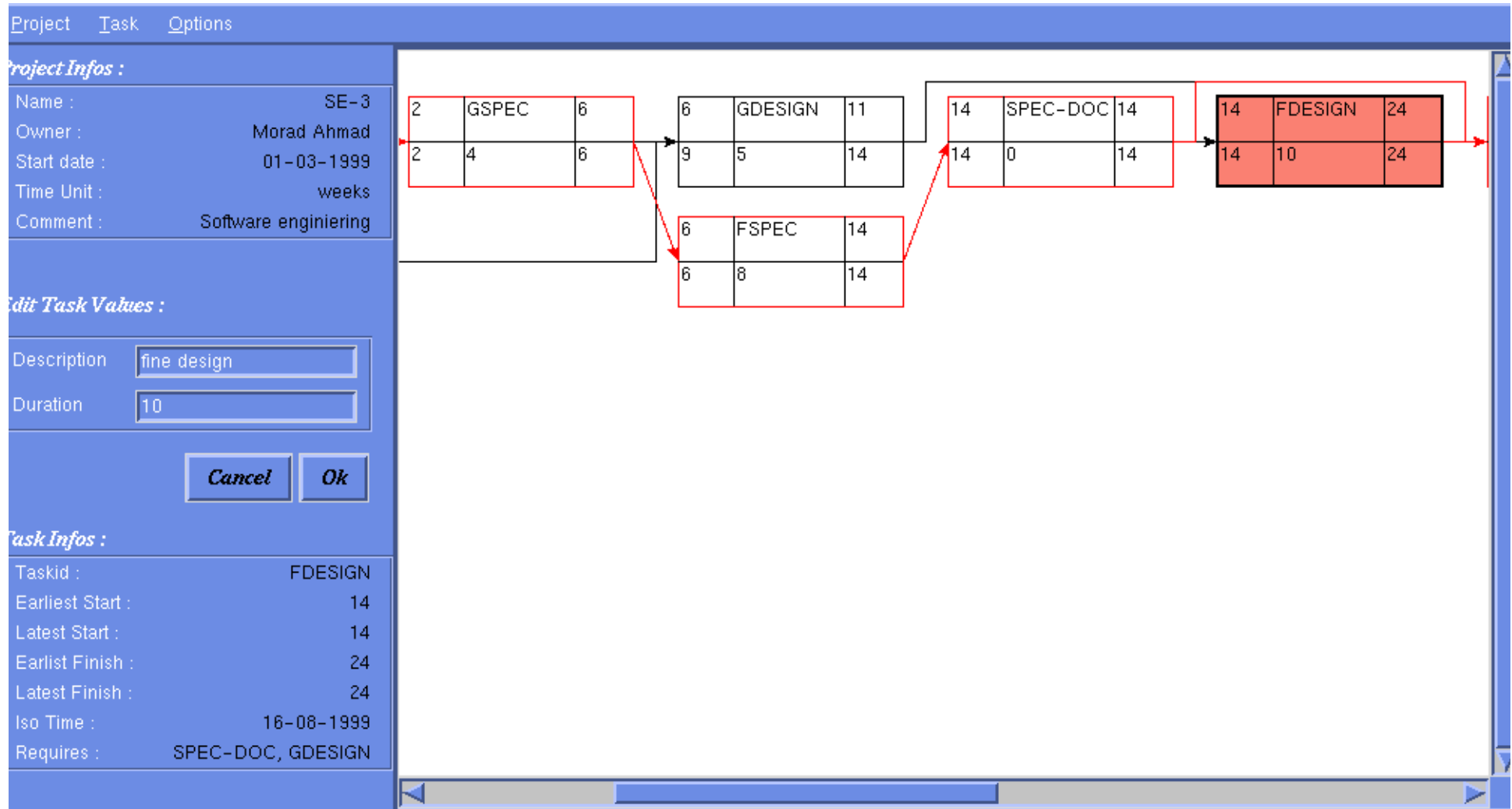
in out next back first last

Status: data received

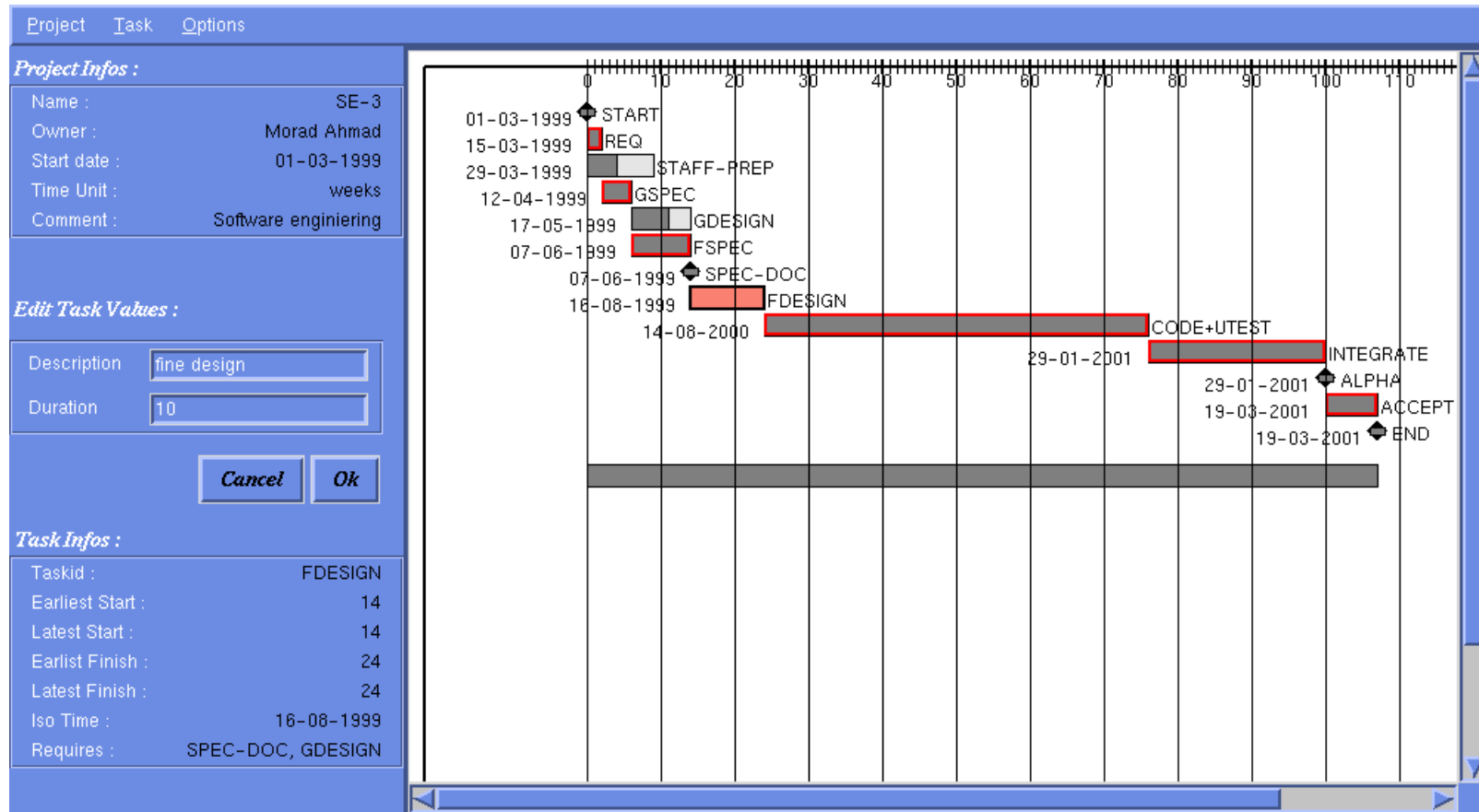
Document: Done.



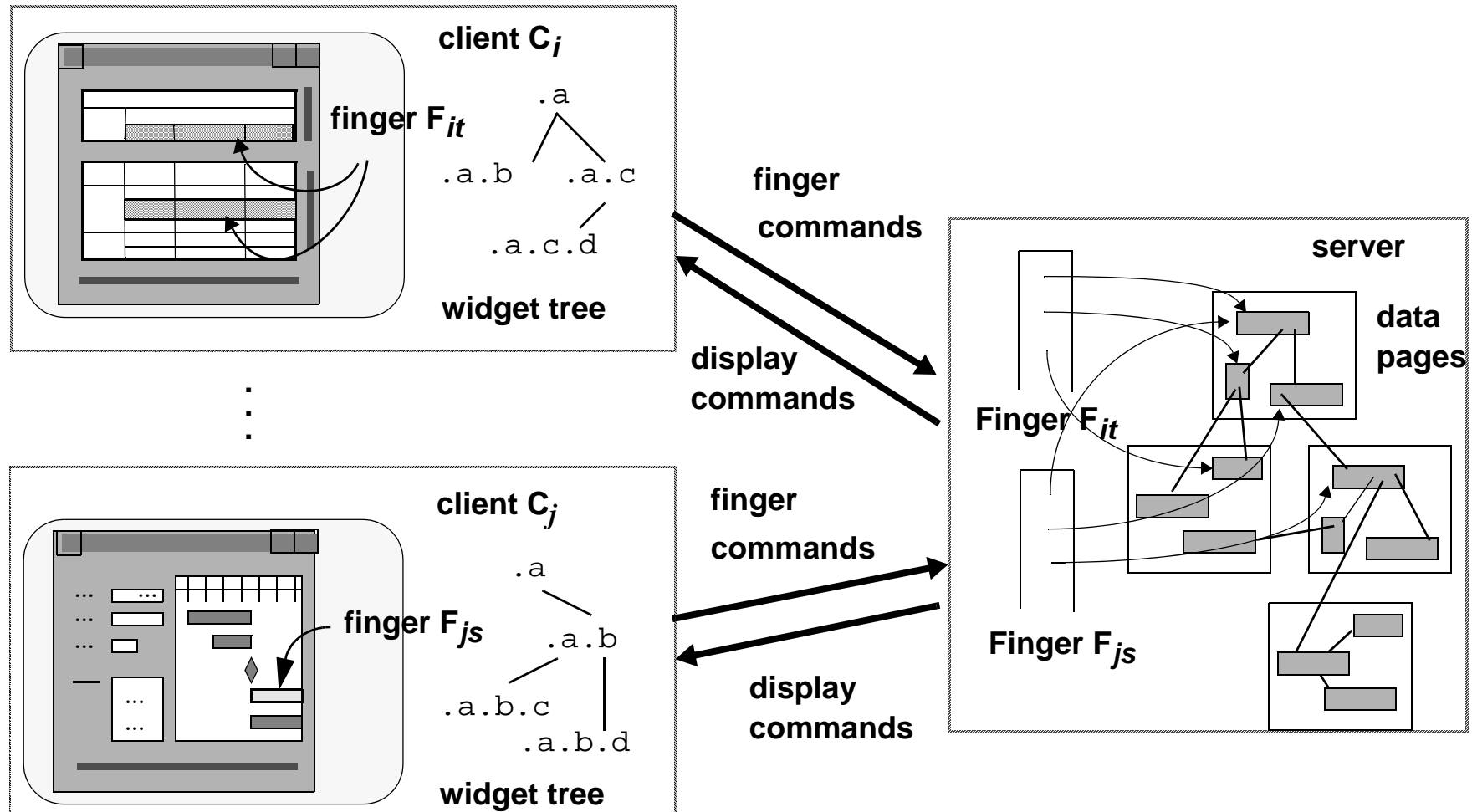
PERT-Diagramm



Gantt-Diagram



Schematische Sicht der Client(s)-Server Verbindung



Die TclDB Sprache

- Skriptsprache aufgesetzt auf Ousterhouts Tcl
- Navigiert, sammelt und editiert Daten aus ESCHER -Tabellen
- Skripte werden selbst ESCHER Attributes

| Command (Sample) | Remark |
|--|---|
| <i>fid</i> push -first -last -where <i>predicate</i> <i>path</i> | with an existing finger <i>fid</i> move on first (default) enclosed object, resp. move finger on last enclosed object move finger on object satisfying <i>predicate</i> , returns 0/1 move finger along a path including attributes and indexes |
| <i>fid</i> pop | move finger <i>fid</i> to enclosing object |
| <i>fid</i> get [<i>path</i>] | return value on which finger rests, optionally extended by path starting with finger position |
| <i>fid</i> set <i>value</i> [<i>path</i>] -tonull [<i>path</i>] -toempty [<i>path</i>] | set value for atomic object on which finger <i>fid</i> sits set complex object to database null turn a set-valued object into an empty set |

Beispiel

```

proc addStyle {fs styleid} {
# collect and return style
# from STYLES.tbl, assume finger
# on set of STYLES.
iterate $fs {
    $fs push -name "SID"
    if {[$fs get] == $styleid} {
        $fs go -name "OPTION"
        set style [$fs get]
        $fs pop
        break
    } else {$fs pop}
}
return $style
}

```

alternatively with descriptive path expression

```

return [$fs get
"?SID == $styleid?.OPTION"]

```

TABLE: STYLES3D.tbl / SCHEMA: STYLES.scm / APPL: w

File Edit Fingers Query Sorting Options Help

| {}STYLES | | |
|------------|--------------------------|-----------|
| SID | OPTION | {}APPLYTO |
| | | #PID |
| PRhalf | PlotRange -> {-0.5, 0.5} | 1 4 |
| LT | Lighting -> True | 3 |
| VPlhcorner | ViewPoint -> {-2, -2, 0} | {} |
| VPabove | ViewPoint -> {0, 0, 2} | {} |
| MF | Mesh -> False | {} |
| SF | Shading -> False | {} |
| PP40 | PlotPoints -> 40 | 2 |
| | | 1 |

Null

Mode: Browse Active Finger: F6

Weitere Elemente der Gruppenarbeit

- **Shared state:** Vorstellung bei allen Beteiligten, daß es einen konsistenten, globalen Zustand gibt
wichtig für **consistency** in Transaktions ACID-Prinzip!
- **WYSIWIS** - Synchrone Anlieferung eines (Teil-)Abbilds des globalen Zustands
dabei Unterscheidung
 - „was ich nicht weiß, macht mich nicht heiß“, wegschauen, passives Beobachten,
Telefon abstellen → information pull
 - „bitte um Nachricht, wenn sich Situation ändert“, Suchen nach Veränderungen, aktives
Beobachten, Schauen mit Telefon („haste gesehen was auf Kanal ... läuft“) !→ information push
- ähnlich Anzeige der **Gruppenzugehörigkeit**: benutzergesteuert (vgl. `who`, `finger`, `zlocate` in UNIX) versus automatisch (Farbmarkierungen im Rollbalken)
- Reduzierung der **kognitiven Last** (cognitive load) bei großem Zustandsraum und häufigen Änderungen

Weitere Elemente der Gruppenarbeit

- **Zugriffskontrolle** (wer darf was ändern, wer darf was sehen)
- **schnelle Reaktion** auf lokal ausgelöste Ereignisse (Mausklick), verzögertes Ansprechen auf entfernte Ereignisse ok wenn insgesamt konsistent (??)
- Metapher der Gruppenarbeit:
 - Tafel
 - Räume (Teilnehmer kann gleichzeitig in mehreren Räumen sein)
- **Ziele der Teilnehmer, Zweck der CSCW-Gruppe**
 - Kooperation, gemeinsame Ziele
 - Konkurrenz, konfliktäre Ziele

Zentrale versus replizierte Verwaltung gemeinsamer Daten

aus [PSL99] *Atul Prakash, Hyong Sop Shim, and Jang Ho Lee:*

Data Management Issues and Trade-Offs in CSCW Systems, IEEE TKDE, Jan. 1999

- **zentralisiert**

es existiert ein Satz von Daten, typisch auf dem Server

es läuft eine Applikation, die ihr GUI verteilt, Mausklick wird in der Applikation verarbeitet und Resultat der Ausgabe wird an alle Client-GUIs verschickt, geht gut mit X-Windows

Beispiele: NetMeeting, XTV, Jupiter, NSTP (notification servers for synchronous groupware)

- **repliziert (replicated)**

verteilte Prozesse halten lokale Kopien der Daten, darunter ggf. eine sog. primäre Kopie

verlangt nach Synchronisationsverfahren; geht in Richtung „verteilte Transaktionen“, kompliziert; vgl. Strom et al., ICDCS Conf. 97

Behauptung: CSCW-Texteditoren arbeiten nach diesem Prinzip

Beispiele: DistEdit mit Sperren verschiedener Granularität, Grove (Ellis, Gibbs, Rein) mit optimistischer Strategie, GroupKit und MMConf mit mehreren Applikationskopien, die Botschaften austauschen

Variationen zum zentralen Ansatz

- reines Fensterverteilen ist unflexibel (Microsoft NetMeeting, XTV, SharedX)
ggf. wird ganzer Bildschirm von der Applikation beherrscht, d.h. eigenes, getrenntes Arbeiten nicht möglich; Vorteil: existierende single user Anwendung muß nicht modifiziert werden.
- shared objects (verteilen der Ansicht einzelner Applikationsobjekte) ist flexibler (NSTP, COAST, DistView), erlaubt verschiedene Sichten der Objekte (vgl. ESCHER Beispiel für Projektmanagement), Applikation muß „collaboration aware“, d.h. auf Gruppenarbeit zugeschnitten sein.
[PSL99]: *Performance is also usually an order of magnitude higher because internal application state changes are propagated (usually small) as opposed to user interface updates (usually larger as they can involve bitmaps and complex graphics operations).*
- Systeme bauen auf Gruppenkommunikations-Untersystem auf, z.B. ISIS (1987), Transis (1992), Consul (1993); bieten z.B. message ordering und Verteilung konsistenter Sichten an
- DistView verlangt nach Sperren, hat aber keine Unterstützung für Abort wenn Sperren nicht erlangt werden können
- COAST und DECAF (Strom et al., ICDCS 97) arbeiten mit optimistischen Protokollen und brechen Transaktionen ab bei Verletzung, DECAF arbeitet mit Zeitstempeln

Anmerkungen zum verteilten Ansatz

Es bietet sich an, Zustandsänderungen inkrementell zu verschicken (gilt auch im zentralen Ansatz mit Objektsichtenverteilung);

dann aber Probleme beim temporären Ausfall eines Anwenders; Stichwort „Heranführen an momentanen Zustand“

Das MVC Prinzip

- MVC = Model-View-Controller
- geht zurück auf Smalltalk-Arbeiten am Interface (InterView), ggf. auf Prof. Trygve Reenskaug, der Xerox PARC 78/79 besucht hatte
- zitiert wird meist: G.E. Krasner and S.T. Pope, „A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk“, J. Object Oriented Programming, pp. 26-49, Aug./Sept. 1988
- Model = Applikationsobjekt (ESCHER: Knoten im Objektbaum)
View = graphische Darstellung eines Applikationsobjekts
Controller = besorgen die Interaktion (Ereignisbehandlung)

Die DECAF Entwicklung

- Distributed Extensible Collaborative Application Framework, „macht nicht wach“
- baut auf MVC auf,
- Quelle: Strom, Banavar, Miller, Prakash und Ward (IBM TJ Watson Research Center und Univ. of Michigan, Ann Arbor), „Concurrency Control and View Notification Algorithms for Collaborative Replicated Objects, Proc. ICDSC 1997, Baltimore
- replicated objects mit optimistische Synchronisation der Updates (optimistic guess propagation, Strom und Yemini 1987)
- optimistische oder pessimistische Sichtennachführung mittels Snapshots (Versand von Momentaufnahmen); pessimistische Sicht: sieht nur Updates von Transaktionen, die Commit gemacht haben
- arbeitet mit Zeitstempeln (virtual time, vgl. auch Lamport Time)
- Synchronisation mit Primärkopienalgorithmus (exclusive writer approach, Chu und Hellerstein 1985)
- abgebrochene Transaktionen werden automatisch von der Auftraggeberstation neu gestartet (Anmerkung: ist gerade bei visuellen Transaktionen fraglich)

Vielzitierte Arbeit ...

die wir nicht haben :-)

- C. Ellis, S.J. Gibbs, and G. Rein. Concurrency control in groupware systems, In Proc. of the ACM SIGMOD '89, pp. 399-407

Behauptung: zentralisierter Ansatz = nicht-optimistisches Protokoll = langsam

- Wenn es nur einen Satz der Daten (beim Server) gibt, lassen sich die Ereignisse gut sequenzialisieren und Sperren können zentral verwaltet werden
- wegen der Latenzzeit und der Notwendigkeit alles zum Server schicken zu müssen, entsteht aber ein System mit „lahmen“ (sluggish) Reaktionen
- Quelle: Saul Greenberg and David Marwood. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface, Proc. 1994 ACM CSCW, Chapel Hill, NC, pp. 207-217

 **ist das wirklich so?**

weitere Aussagen:

- Konflikte in CSCW-Umgebungen sind rar, weil Anwender sich arrangieren
- Warten auf Zuteilung von Sperren ist Anwendern nicht zuzumuten (?); Beispiel: Anwender will Cursor setzen in einem CSCW-Editor und sofort loslegen mit dem Eintippen
- Zurücksetzen der Aktion bei Verweigerung optimistischer Sperren ist schwierig und kann, je später die Verweigerung erfolgt, zur Verwirrung führen:
 - was genau hat zur Änderung der Anzeige geführt?
 - Aktion eines anderen Teilnehmers?
 - unvorhergesehene Reaktion der Anwendung auf eigene Aktion?
 - Abbruch und Zurücksetzen der eigenen Aktion?
- Sperrwegnahme bei Deadlock, wenn Anwender (Besitzer) schläft
- großflächige Sperren (large grain granularity) können ggf. in einem Token-passing Verfahren vergeben werden
- vgl. auch Kapitel 9 in der Doktorarbeit (?) von Alex Mitchell, Verfasser von Callope
www.dgp.toronto.edu/people/alex/thesis/chapter9.html:
Concurrency Control



Klassische Transaktionsanomalien aus CSCW-Sicht

Lost Update

| Prozeß 1 | Zeit | Prozeß 2 |
|-----------|------|-----------|
| read(x) | 1 | |
| | 2 | read(x) |
| update(x) | 3 | |
| | 4 | update(x) |
| write(x) | 5 | |
| | 6 | write(x) |

update „verloren“



In visuellen Umgebungen ist lost-update wie schnell aufeinanderfolgendes Ändern
kann aber zu Verunsicherung führen

Beispiel: Ändern Vorlesungszeit

Nimbus Arbeitsfinger

Fokus Steuerung (viewport)

Finger eines anderen Teilnehmers

Courses.tbl

File Edit Fingers Options Help

CourseID
CS409

Title
Graphical User Interfaces

Time
Mo. 3 pm

Lecturer
Thamm

<>Students
mary@cs
lukas@cs
anne@cs
john@ee
paul@ee

CourseID
CS411

Title
Network Programming

Time
We. 9 am

Lecturer
Wilke

<>Students
susan@cs
gabi@ee
steve@cs
marc@ee
john@ee

Mode: Browse Active Finger: F1

In Out Prev Next Save BeginQ

Dirty Read

| Prozeß 1 | Zeit | Prozeß 2 |
|-----------|------|-----------|
| read(x) | 1 | |
| update(x) | 2 | |
| write(x) | 3 | |
| | 4 | read(x) |
| | 5 | update(x) |
| abort | 6 | |
| | 7 | write(x) |

x - Wert wird zurückgezogen

Beispiel in CSCW-Szenario:

Prof W. will Zeit für Vorlesung ändern. Student A sieht noch nicht gesicherte Änderung und schreibt sich daraufhin für Vorlesung ein, Prof. W. macht Rückzieher wegen Raumkonflikt und läßt altes Datum stehen, Student A ist schon abgemeldet und bekommt von Rücksetzung nichts mit.

Inconsistent Read

| Prozeß 1 | Zeit | Prozeß 2 |
|----------------|------|-------------|
| sum := 0 | 1 | |
| read(x) | 2 | |
| read(y) | 3 | |
| sum := sum + x | 4 | |
| sum := sum + y | 5 | |
| | 6 | read(z) |
| | 7 | z := z - 10 |
| | 8 | write(z) |
| | 9 | read(x) |
| | 10 | x := x + 10 |
| | 11 | write(x) |
| read(z) | 12 | |
| sum := sum + z | 13 | |

Mögliche Vorgehensweise in ESCHER: fingerloses Lesen

- Focus = Fenster, alle Tupel werden gelesen, die man im Fenster sieht
- Nimbus = Finger, man kann nur Werte verändern, auf die der Finger zeigt
- Setzen eines Fingers auf einen Wert (ein Objekt, z.B. eine Zeichnung) = Schreibsperre setzen
- Implizites Beginn-of-Transaktion (BOT) durch erstes Fingersetzen
- Explizites EOT verlangt (OK-Taste drücken)

Ablauf:

- Teilnehmer clickt mit Maus Objekt an
- Objekt bekommt lokal sofort blinkenden Finger (lock requested)
- RID des Objekts geht an Server, dieser entscheidet durch hash-Tabellenzugriff ob Sperre gewährt werden kann
- Wenn ja kommt ok zurück, Blinken geht in Dauerfärbung über, andere Clients, die das Objekt in einem Fenster haben, erhalten Nachricht (RID + Teilnehmerkennung) und färben ihr Objekt
- Wenn nein, Konfirmation-Box (Warning: Lock not granted! Wait or Abort?) und Färbung des Objekts durch den Finger der anderen Transaktion, die das Objekt sperren konnte, weil sie früher dran war.



- Wenn bei Nichterlangung der Sperre „Warten“ gewählt wird, muß Verklemmungserkennung nach einer Zeit anlaufen. Auflösung ggf. durch Verhandlung der Teilnehmer.
- Verlangt Transaktion mehrere Schreibzugriffe (z.B. Cut & Paste) sollte Teilnehmer zuerst alle Finger setzen, bevor er die tatsächlichen Operationen ausführt.
- Gelingt es ihm, alle Sperren zu erlangen, kann er alle Aktionen ausführen und danach - auf einmal oder Schritt um Schritt - alle Sperren (Finger) wieder freigeben → Zwei-Ohasen-Sperren oder striktes 2PL
- In der Regel wird Teilnehmer lieber Sachen Schritt um Schritt ändern (also nicht erst alle Sperren erlangen); dann aber Gefahr abbrechen zu müssen (Abort mit Undo), durchaus auch mehrstufig in graphischen Editoren üblich
- Änderungen an den Objekten erfolgen im Server und werden an die Teilnehmer mit Fenstern, in die das Objekt hineinragt, als View-Updates weitergegeben. Erfolgt dies sofort, kann es zu **Dirty Read** kommen, wenn Transaktion abgebrochen wird. Anderer Teilnehmer hat aber gesehen, daß ein Finger auf dem Objekt steht und daß der Wert „in Bearbeitung“ ist.
- Alternativ pessimistisches Verfahren, d.h. Werteänderungen erst weitergeben wenn Commit erfolgt (andere Teilnehmer sehen nur Werte von garantiert erfolgreich abgeschlossenen Transaktionen)
- Alternativ löschen aller Werte auf die ein Schreibfinger zeigt und schrittweises neusetzen;



dieses Verfahren beseitigt Unklarheit, ob ein Wert schon geändert ist oder noch geändert wird (blank = noch nicht geändert). Kann mit pessimistisch oder optimistischen View-Updates kombiniert werden.

- Intention Locks für komplexe Objekte (hart für den Anwender, ggf. automatisiert in Abhängigkeit wohin ein Finger gesetzt wird)

Alternativ Lesen mit Finger (vgl TclDB Skripte)

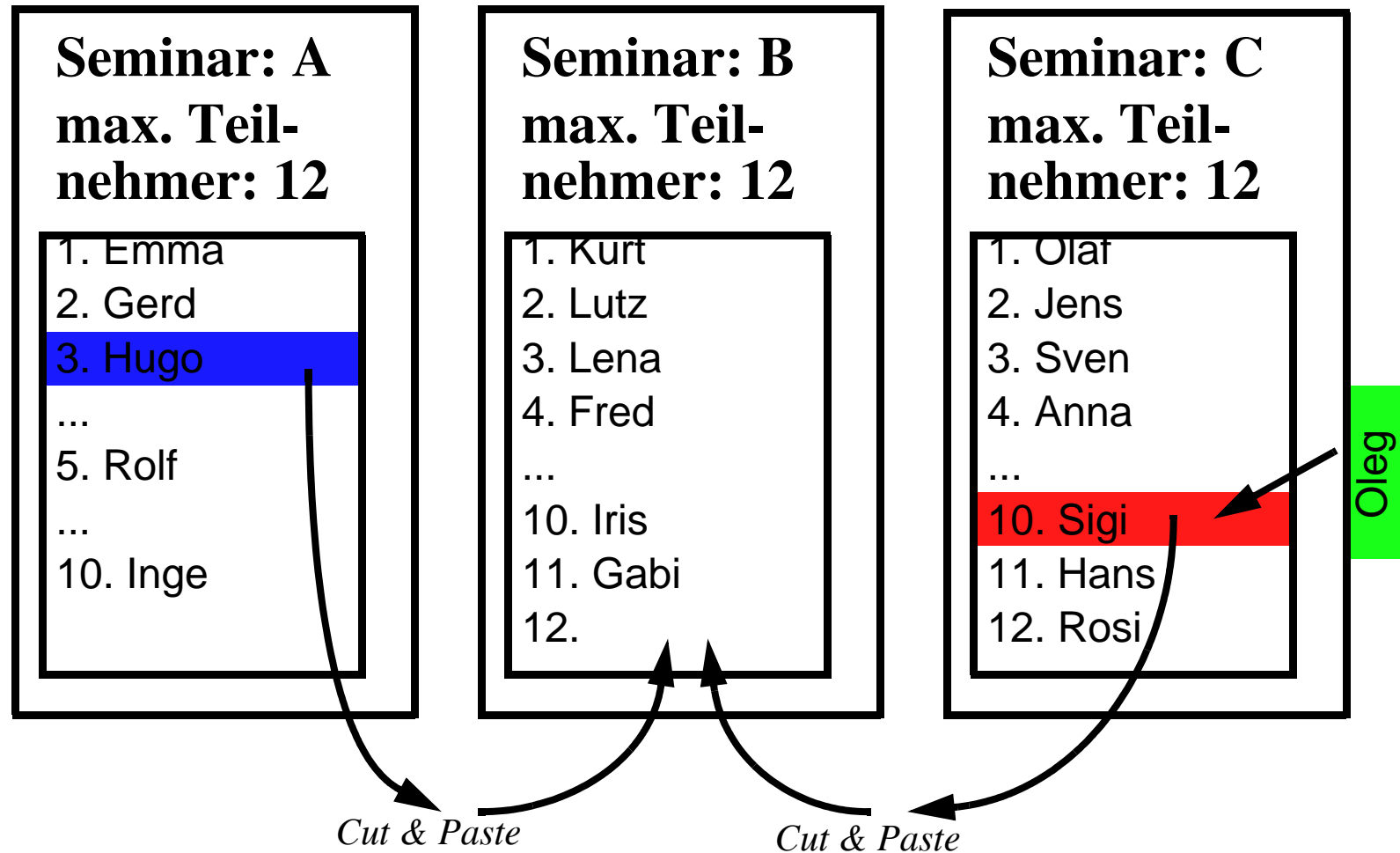
Unklar wie Kombination von visuellen und Skripttransaktionen ablaufen

Beispiel unten:

Hugo aus Seminar A und Sigi aus Seminar C wollen auf den einen freien Platz in Seminar B. Hugo löscht aus A, Sigi löscht aus C. Hugo kann zuerst Paste auf Seminar B machen und löscht Sperren. Sigi kann sich nicht eintragen, da Seminar B voll, hat ggf. Sperre auf C freigegeben. Oleg kommt und trägt sich in C ein, das jetzt auch wieder voll ist! Sigi kann nicht zurück, verzweifelt und beschließt Datenbankprofessor zu werden ...

Alternative Szenarien: Hugo möchte lieber nach C als nach B, kann aber erst, wenn Sigi nach B umgezogen ist und niemand sofort seinen Platz einnimmt ... Koordination!

Beispiel: Anmeldung in Seminar ändern



Zusammenfassung

- ESCHER als CSCW-Tool tendiert Richtung zentrale Lösung mit View-Updates
- erfüllt MVC-Konzept
- Finger als Schreibsperrern
- Fingeroperationen elementar, universell, natürlich (deictisches Handeln)
- fingerloses Lesen ok durch visuelle Präsenz des Schreibvorgangs, Fokusänderung über Rollbalken, usw.
- klassisches 2PL mit simultanem Freigeben der Finger einer Transaktion
- Group Awareness über Finger und ggf. Rollbalkenmarkierungen
- Äquivalenz visueller Operationen zu Read/Write wäre noch zu zeigen
- Undo/Redo Operationen und Fragen der Persistenz hier nicht behandelt

☞ **klassisches Promotionsthema, hochaktuell, aber gefahrlos zu behandeln, weil Fingeridee und ESCHER-Ansatz mit TclDB proprietäre Kasseler Ideen sind**