Model Checking Groupware Protocols

Maurice H. ter Beek

Institute of Information Science and Technologies
National Research Council
(ISTI-CNR)

Pisa, Italy

Thursday 13 May 2004

COOP'04

French Riviera

⇒ joint work with Mieke Massink,
Diego Latella & Stefania Gnesi

Outline

- case study on Clock toolkit & its protocols
- focus on concurrency control & distributed notification aspects
- model checking with Spin, Promela & LTL
- verification of correctness criteria
- conclusions & future work

Clock: A Groupware Toolkit

groupware toolkits: programming abstractions simplifying groupware application development

e.g. Rendezvous, GroupKit, Clock, . . .

Clock: high-level groupware toolkit with design-level architecture based on the MVC paradigm

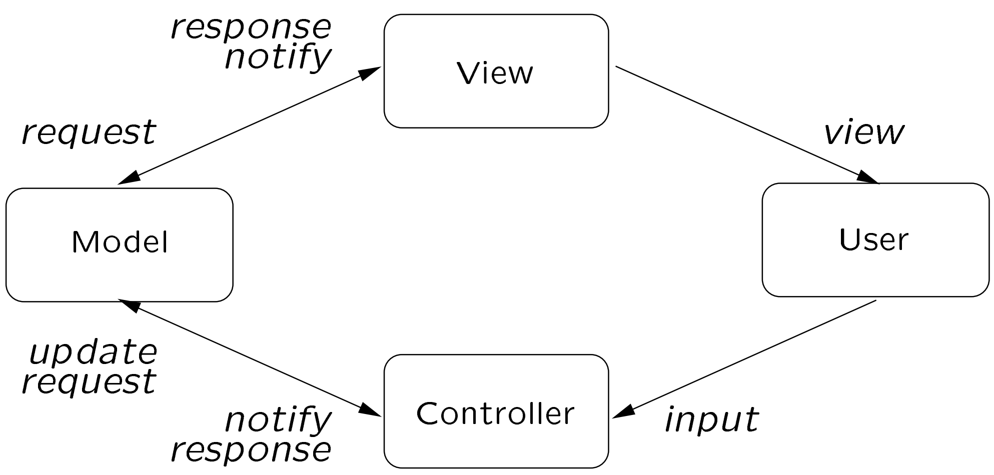
applications of Clock:

- multi-user video annotation tool
- multi-user GroupScape HTML browser
- multi-user design rationale editor
- ScenicVista user interface design tool
- . . .

MVC Paradigm

Partition of architecture organising interactive applications into three separate parts:

- *Model* — implementing data state
- *View* — computing graphical output
- *Controller* — interpreting user inputs



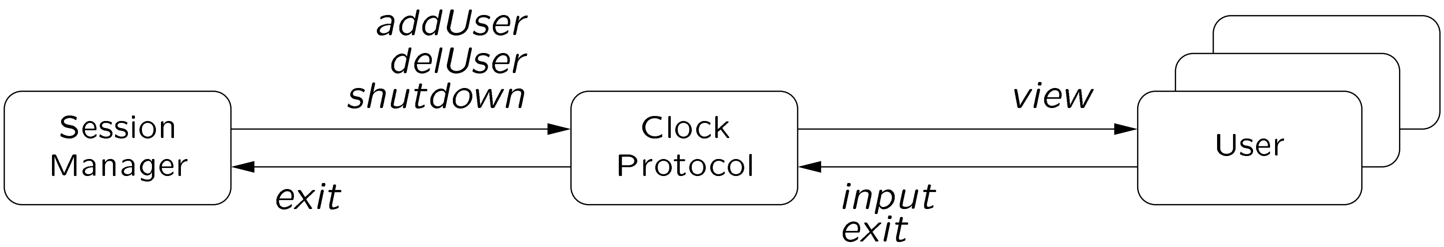
Clock Protocol

[Urn98]: Tore Urnes' Ph.D. thesis

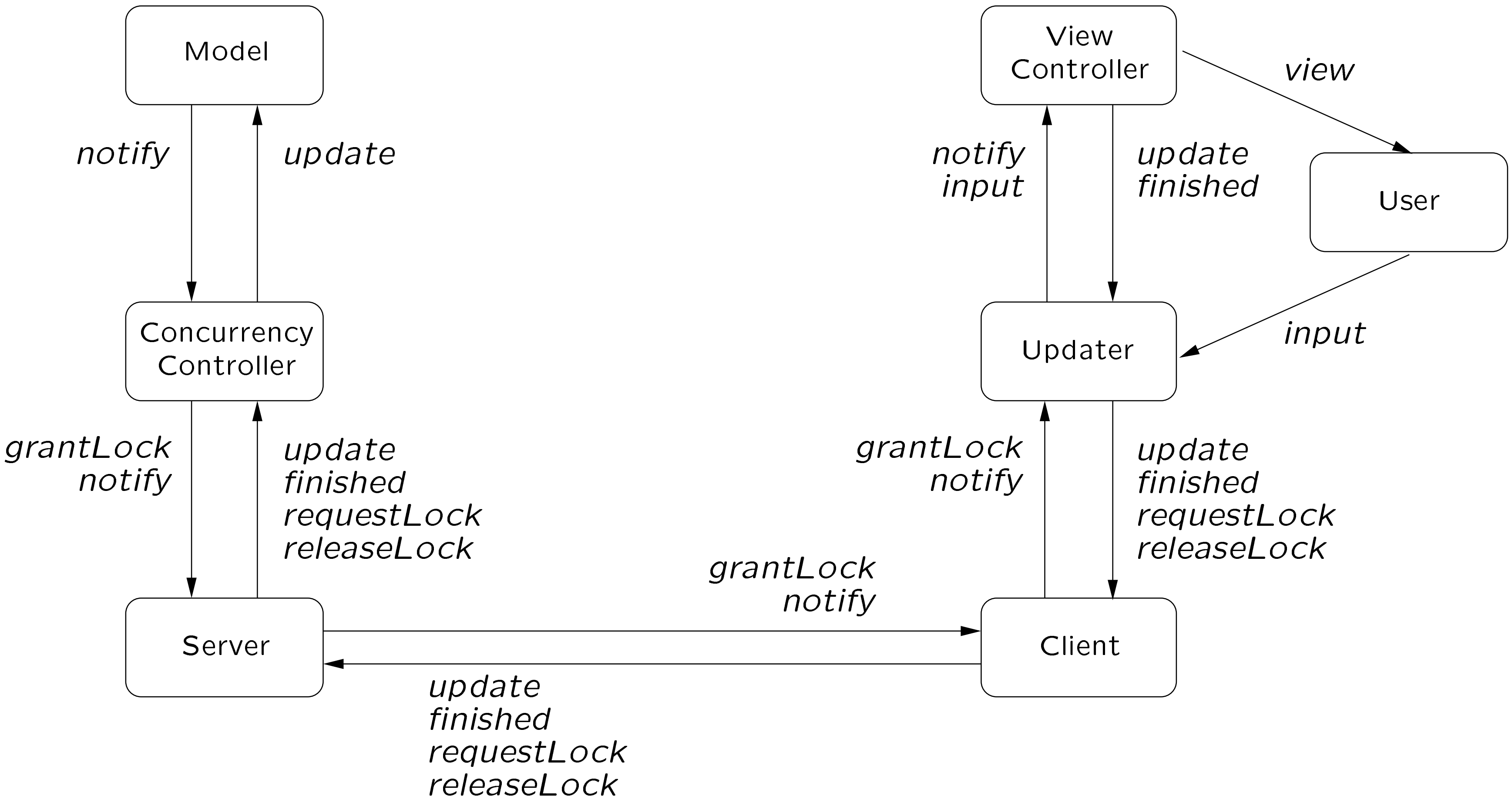
"Efficiently Implementing Synchronous Groupware"

Network communication protocols between the server (model) & the clients (view/controller):

- MVC protocol $\surd$
- Cache protocol
- Concurrency Control (CC) protocol $\surd$
- Replication protocol
- session management



MVCC Protocol



Case Study (1)

[Urn98]: full specification of Clock protocol, impossible to verify with model checker Spin

Model Checking

- automatic technique to verify if a concurrent system design satisfies its specifications

- very hard in standard ways (such as testing) due to non-determinism & interleaving

$\Rightarrow$ groupware systems are highly concurrent !

- exhaustive verification, i.e. takes into account all possible input combinations & states

- risk of running out of memory due to state-space explosion

$\Rightarrow$ a simplified model is used, still capturing the core of the system design while abstracting from unnecessary details

Spin — Simple Promela INterpreter

- state-of-the-art on-the-fly model checker
- developed at Bell Labs by G. Holzmann
- formal verification of distributed systems specified in Promela
- verify deadlocks/assertions/unreachable code/LTL formulae/liveness/. . .
- counterexample if a property is violated !
- very well documented: `www.spinroot.com`
- winner ACM System Software Award '01

Promela — PROcess MEta LAnguage

- non-deterministic C-like specification language
- loosely based on Dijkstra's guarded commands
- borrows notation for I/O operations from CSP
- finite-state systems communicating by channels

Example: The User Process

```
proctype User(byte id)
{
  do
  :: userToUpdater[id]!input;
  :: vcToUser[id]?view;
  od
}
```

Case Study (2)

Applied abstractions:

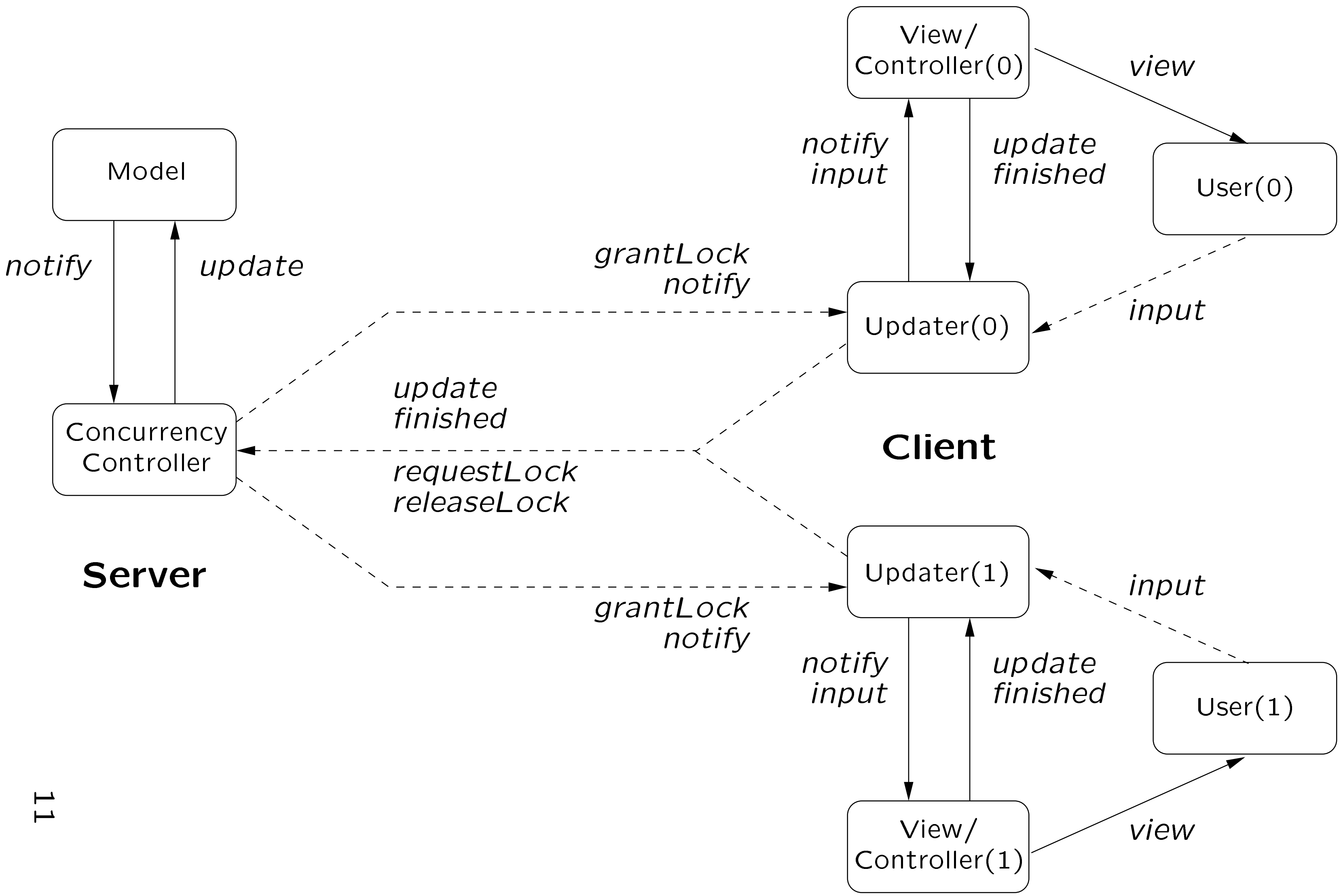
- only specification of MVC & CC protocols
- integrating the Server & Client processes
- handshake rather than buffered channels
- grouping assignments into atomic blocks

$\Rightarrow$ state vector reduced with a factor > 2 !

$\Rightarrow$ state space reduced with a factor $> 10^3$!

Towards template specification for verifying concurrency control & distributed notification issues in groupware protocols !

Modified MVCC Protocol



Correctness Criteria

MVCC protocol must satisfy [Urn98]:

data consistency: server's shared data state must be kept consistent

view consistency: no updates may be allowed during view recomputation

absence of starvation: every user input must eventually be responded to

MVCC based on locking must satisfy:

concurrency control: every lock request must eventually be granted, only one user at a time may possess a lock & every lock must eventually be released

Model Checking MVCC

⇒ formalise correctness criteria in LTL &
verify all correctness criteria with Spin !

LTL — Linear Time Logic

- propositional logic & temporal operators
- useful for specifying liveness properties
- property must hold always/eventually/. . .

e.g.: absence of starvation, i.e. each user can always eventually provide input

```
... userToUpdater[id]!input; doneInput: skip ...
```

LTL: `[] <> User[pid]@doneInput`

id User process   progress label

Concurrency Control

e.g.: every lock request is eventually granted

`updaterToCC!requestLock,id` `ccToUpdater[id]?grantLock`
`doneRequestLock: skip` `doneGrantLock: skip`

LTL: [] (Updater[3]@doneRequestLock −>

< > Updater[3]@doneGrantLock)

Spin: valid ! ($\pm$ 15 min.)

LTL: ! (< > Updater[3]@doneRequestLock)

Spin: *not* valid, i.e. counterexample ! ($<$ 1 sec.)

(idem for Updater[6])

Data Consistency

e.g.: each update of the server's shared data state is communicated to all clients

i.e. : distributed notification !

```
ccToModel?update,ID          ccToUpdater[id]?notify,ID
doneUpdateID: skip           doneNotifyID: skip
```

LTL: [] (Model[1]@doneUpdateID −>

(<> Updater[3]@doneNotifyID &&

<> Updater[6]@doneNotifyID))

Spin: valid ! (± 30 min.)

View Consistency

e.g.: any user's view recomputation based on an earlier input has finished, *before* this user can provide further input

```
proctype User(byte id) {
do
:: userToUpdater[id]!input; doneInput:skip; doneInput2:skip
:: vcToUser[id]?view; doneView:skip
od }
```

LTL: [] (User[5]@doneInput2 ->

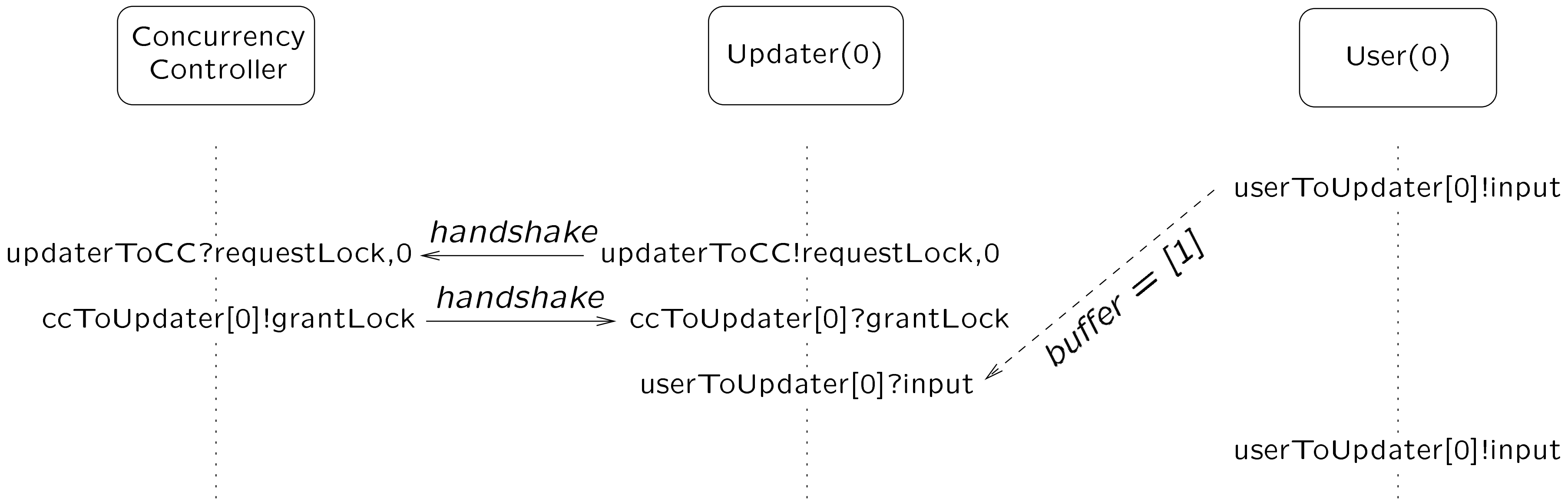
(!User[5]@doneInput && !User[8]@doneInput) U

(User[5]@doneView && User[8]@doneView)

Spin: *not* valid, i.e. counterexample! ($<$ 1 sec.)

(idem for User[8])

Counterexample View Consistency



Absence of Starvation

e.g.: User can always eventually provide input

```
proctype User(byte id) {
  do
  :: userToUpdater[id]!input;
     doneInput: skip
  :: vcToUser[id]?view;
  od }
```

LTL: [] <> User[5]@doneInput

Spin: *not* valid, i.e. counterexample ! (< 1 sec.)

(idem for User[8])

⇒ study (weak) fairness in relation to Spin/LTL !

Spin and Fairness

Spin's notion of weak fairness is implemented at the process level rather than statement level

⇒ express notion of fairness in LTL formulae!

```
proctype ViewController(byte id) {
  do
  ::  updaterToVC[id]?input -> checkInput: skip;
      vcToUpdater[id]!update
  ::  updaterToVC[id]?notify -> checkNotify: skip;
      vcToUser[id]!view; vcToUpdater[id]!finished
  od }
```

LTL: ([] <> VC[4]@checkNotify &&

[] <> VC[4]@checkInput) ->

[] <> User[5]@doneInput

Spin: valid! (± 20 min.)

(idem for VC[7] and User[8])

Conclusions

- case study on formalisation & verification of *those* protocols underlying Clock related to concurrency control & distributed notification

- feasibility & usefulness of model checking for verifying groupware protocols also in general

  e.g. important groupware issues such as data consistency, view consistency, absence of (user) starvation & concurrency control

- provide a template specification for verifying concurrency control & distributed notification issues in groupware protocols !

Future Work

- rest of Clock protocol: Cache & Replication protocols, session management, different CC

  e.g. use compositional modelling techniques like process algebras or *team automata*

- perform qualitative & quantitative verification

  e.g. stochastic process algebras & automata

- apply this acquired knowledge & experience to other (groupware) protocols !

Currently modelling & verifying the addition of a *publish/subscribe notification service* to the groupware protocol of thinkteam® (think3 Inc.)

Clock Groupware Toolkit

T.C.N. Graham, T. Urnes & R. Nejabi (1996)

"Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware"

UIST'96 - ACM Symp. on User Interface Software & Technology

&

T. Urnes & T.C.N. Graham (1999)

"Flexibly Mapping Synchronous Groupware Architectures to Distributed Implementations"

DSVIS'99 - Design, Specification & Verification of Interactive Systems

MVC Paradigm

G.E. Krasner & S.T. Pope (1988)

"A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80"

Journal of Object-Oriented Programming

Applications of Clock Toolkit (1)

Multi-user video annotation tool:

T.C.N. Graham & T. Urnes (1997)

"Integrating Support for Temporal Media into an Architecture for Graphical User Interfaces"

ICSE'97 - International Conference on Software Engineering

Multi-user GroupScape HTML browser:

T.C.N. Graham (1997)

"GroupScape: Integrating Synchronous Groupware and the World Wide Web"

INTERACT'97 - International Conference on Human-Computer Interaction

Applications of Clock Toolkit (2)

Multi-user design rationale editor:

M. Sage & Ch. Johnson (1998)

"Pragmatic Formal Design: A Case Study in Integrating Formal Methods into the HCI Development Cycle"

DSVIS'98 - Design, Specification & Verification of Interactive Systems

ScenicVista user interface design tool:

J. Brown & S. Marshall (1998)

"Sharing Human-Computer Interaction and Software Engineering Design Artifacts"

OzCHI'98 - Australian Computer-Human Interaction Conference