Model Checking Groupware Protocols

bringing together theory & practice

Maurice H. ter Beek

ERCIM fellow

Thursday 19 June 2003

Workshop SP4

ISTI-CNR, Pisa

$\Rightarrow$ joint work with Mieke Massink!

Outline

- Background: groupware & Team Automata
- Case study: Clock toolkit & its protocols
- Focus: Concurrency Control protocol
- Verification: correctness criteria CC protocol
- Model checking: Spin, Promela & LTL
- Current research: questions & directions
- Future research: bridging theory & practice

Groupware

P. Johnson-Lenz & T. Johnson-Lenz (1980)

"Groupware: the emerging art of orchestrating collective intelligence"

World Future Society's First Global Conference on the Future

groupware: computer technology intended to support groups of people working together

synchronous groupware: groupware that allows real-time collaboration

e.g. collaborative writing/designing (Think3), video conferencing, multi-user games, . . .

groupware toolkits: programming abstractions simplifying groupware application development

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

$\Rightarrow$ distributive nature: (network) communication and concurrency control are key issues, i.e. possible migration to web applications!

History of Team Automata

C.A. (Skip) Ellis (1997)

"Team Automata for Groupware Systems"

GROUP'97 International ACM Conference on Supporting Group Work

•

M.H. ter Beek, C.A. Ellis, J. Kleijn & G. Rozenberg ($>$1997)

"Synchronizations in Team Automata for Groupware Systems"

Computer Supported Cooperative Work: J. of Collaborative Computing

•

"Team Automata for Spatial Access Control"

ECSCW'01 Seventh European CSCW Conference

•

M.H. ter Beek & J. Kleijn

"Team Automata Satisfying Compositionality"

FM'03 Twelfth International FME Symposium

•

M.H. ter Beek's PhD thesis "Team Automata"

Team Automata

- model logical architecture of a design
- abstract from concrete data and actions
- describe behaviour in terms of

  - state-action diagram (automaton)
  - role of actions (input, output, internal)
  - synchronizations (simultaneous execution of common actions)

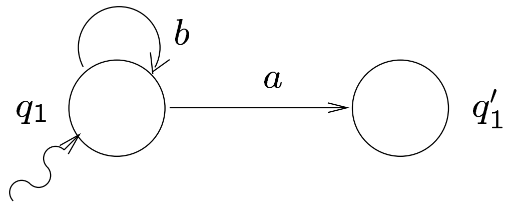
$\Rightarrow$ precise & rigorous:

+ flexible (role of actions, choice of transitions)
+ scalable (modular construction, iteration)
+ extendible (time, probablities, priorities)
+ verifiable (automata-theoretic tools)

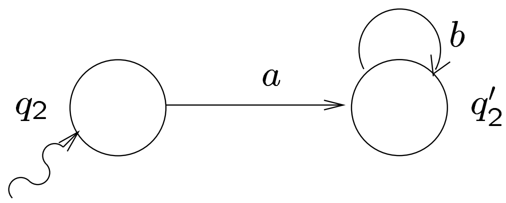
– no tool (yet)

Example Team Automata

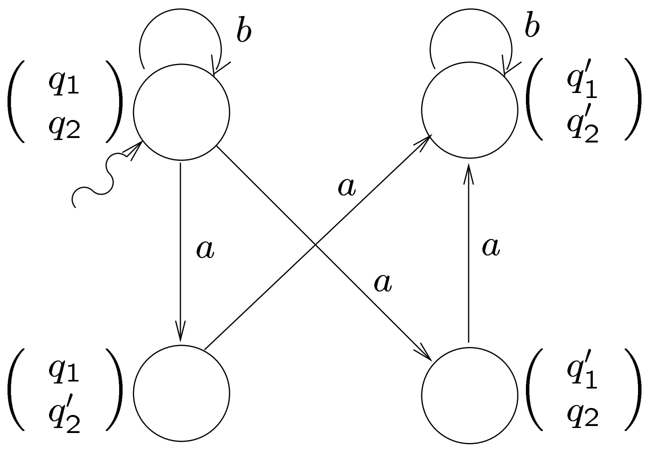
$\mathcal{C}_1$:



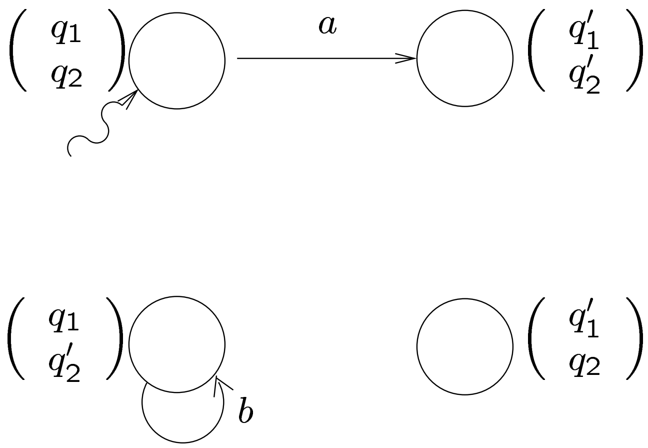
$\mathcal{C}_2$:



$\mathcal{T}^{free}$:



$\mathcal{T}^{ai}$:



$\Rightarrow$ team automata $\mathcal{T}^{free}$ and $\mathcal{T}^{ai}$ over $\{\mathcal{C}_1, \mathcal{C}_2\}$ are defined by choosing their transitions!

Examples Groupware & TA: GROVE Document Editor Architecture & Spatial Access Control

C.A. Ellis, S.J. Gibbs & G. Rein (1990)

"Design and Use of a Group Editor"

Engineering for Human Computer Interaction

A. Bullock & S. Benford (1999)

"An Access Control Framework for Multi-User Collaborative Environments"

GROUP'99 International ACM Conference on Supporting Group Work

$\Rightarrow$ clear need for combining theory & practice: automated verification / model checking

[BB03]: initial attempt to verification of TA design for access control with Spin

Clock Toolkit

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

"Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware"

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

•

high-level groupware toolkit

with

design-level architecture based on Model-View-Controller paradigm

•

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

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

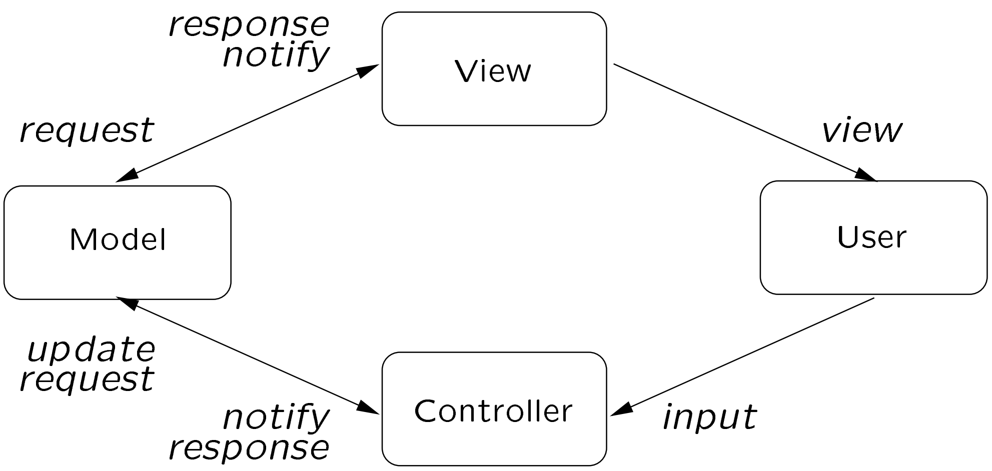
Journal of Object-Oriented Programming

MVC Paradigm

$\Downarrow$

partition architecture organizing interactive applications into three separate parts:

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



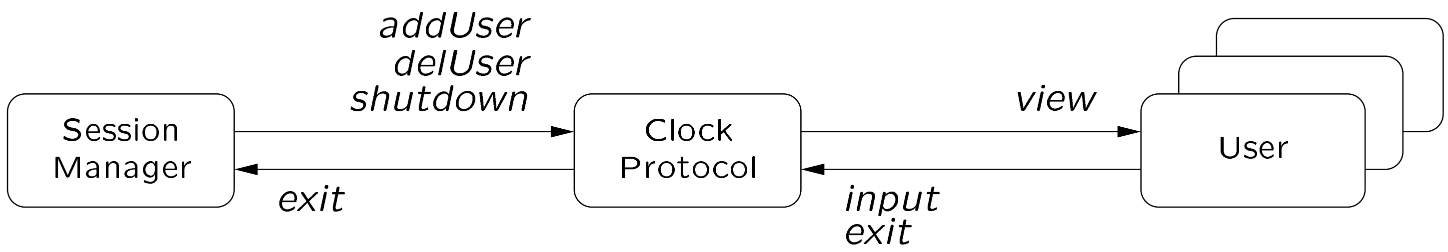
Clock Protocol

[Urn98]: T. Urnes' PhD thesis

"Efficiently Implementing Synchronous Groupware"

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

- MVC protocol $\surd$
- Cache protocol
- Concurrency Control protocol $\surd$
- Replication protocol
- Session control



Case Study

1. verify MVC & CC protocols with Spin

   + Promela specification available

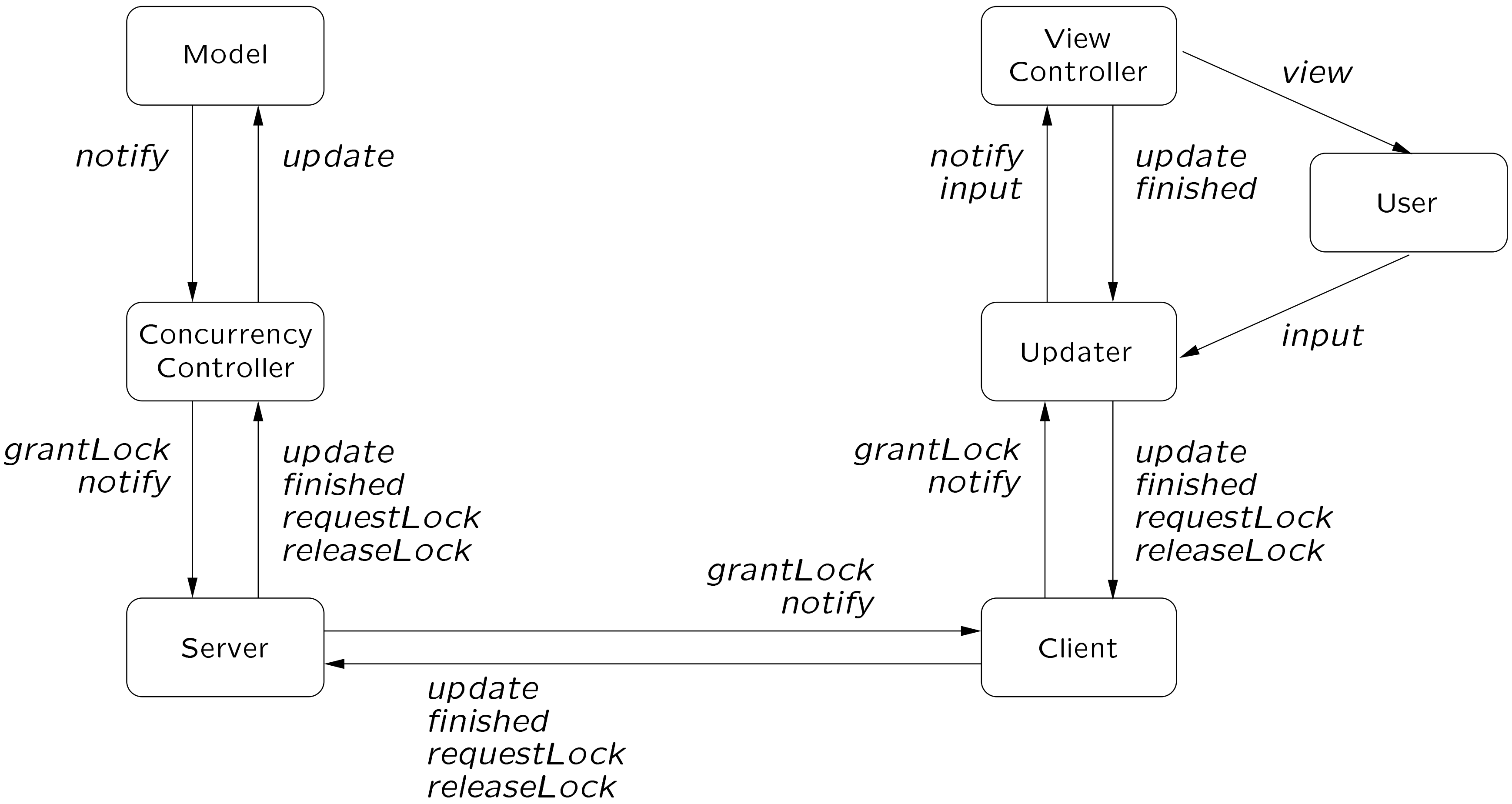
   + verification with Spin attempted

2. verify more protocols with Spin/Jack/Hal

⇒ Think3: communication, awareness, session control, data/view consistency, . . .

⇒ Web applications: network communication, concurrency control, . . .

CC Protocol with locking



Correctness Criteria [Urn98]

CC protocol must satisfy:

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

view consistency: no updates allowed during view recomputation

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

i.e. not only mutual exclusion, but also data/view consistency & atomic updates

[Urn98]: initial attempt to verification with Spin

Spin — Simple Promela INterpreter

- state-of-the-art on-the-fly model checker
- formal verification of distributed systems
- developed at Bell Labs (G. Holzmann, 1980)
- distributed systems specified in Promela
- verify deadlocks/assertions/LTL formulae/ unreachable code/liveness/...
- property violation shown by counterexample

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

LTL — Linear Time Logic

- propositional logic & temporal operators
- used for specifying liveness properties
- property must hold <u>always</u>/<u>eventually</u>/...

Correctness Criteria [BM$^+$03]

CC protocol must satisfy:

1. User can always provide *input*
   (a) If it does, each user screen must eventually be *updated*
   (b) Moreover, this *update* must happen before any user can provide *input*
2. CC may always have *granted* only one *lock*
3. Each *lock request* must eventually be *granted*
4. Each *notify* must eventually be followed by *finished*

⋮

Current Research Questions

? verify all correctness criteria with Spin

$\Rightarrow$ formalize correctness criteria in LTL &
suitable model with acceptable state space

e.g.: 1. User can <u>always</u> provide *input*

LTL: ([] <> (User[pid]@doneInput)

$\downarrow$ id User process $\downarrow$ progress label

! not valid

$\Rightarrow$ study (weak) fairness in relation to Spin/LTL

Current Research Directions

⇒ search and study related work

⇒ study relation synchronous groupware & web (e.g. required underlying protocols, available primitives, models & requirements)

⇒ use abstract models like TA to guide more detailed modelling in Promela/Spin (or other specification language & model checker)

⋮

Future Research Directions

- CC protocol with eager mechanism rather than locking mechanism (presend caching!)

  add environment (Session Manager & Session Updater processes with *addUser*, *exit*, *delUser*, *disconnect*, and *shutdown* actions)

  Cache protocols (simple, prefetch, presend)

  Replication protocol

- qualitative verification with Jack/Hal/ . . .

- quantitative verification with Petri nets/TA

⇒ apply acquired knowledge & experience to other (groupware) protocols!
e.g. Think3 & web applications

Audience participation time!

WANTED

WELL-DOCUMENTED
(GROUPWARE) PROTOCOLS

$\Downarrow$

RETURNED

FORMAL VERIFICATION

—

QUALITATIVE

&

QUANTITATIVE