Model Checking Publish/Subscribe Notification for thinkteam®

Maurice H. ter Beek

FM&&T, ISTI-CNR

Monday 12 July 2004

Workshop SP4

ISTI-CNR

⇒ this is joint work with Mieke Massink, Diego Latella, and Stefania Gnesi from FM&&T, and Alessandro Forghieri and Maurizio Sebastianis from think3

Outline

- case study on the groupware protocol that underlies ‘thinkteam’
- focus on the addition of a publish/subscribe notification service
- model checking with Spin, Promela & LTL
- verification of correctness criteria
- conclusions & future work

Case Study: ‘thinkteam’ (TT)

think3’s Product Data Management (PDM) application

A dispersed & asynchronous groupware system

Provides PDM needs of design processes in the manufacturing industry

Strengths: rapid deployment & startup cycle, flexible, smooth integration with ‘thinkdesign’ (think3’s CAD solution) & 3rd party products

Helps to capture, organise, automate & share engineering product information efficiently

Controlled storage and retrieval of document data in PDM applications is called *vaulting*

TT's Vaulting Subsystem

(1) provides a single, secure & controlled storage environment, where the documents controlled by the PDM application are managed

(2) prevents inconsistent updates or changes to documents, while still allowing the maximal access compatible with the business rules

Implementation:

(1) subject of vaulting subsystem's lower layers

(2) in TT's underlying groupware protocol by a standard set of operations on TT's *vault*, a file-system-like repository

Operations on TT's Vault

get: extract a read-only copy of a document

import: insert an external document

checkOut: extract an exclusive copy of a document (with the intent to modify it)

checkIn: replace an edited (& hence previously checked out) document

checkInOut: replace an edited document (while retaining it as checked out)

unCheckOut: cancel the effects of a *checkOut*

Adding Publish/Subscribe Notification

Raise user awareness by intelligent data sharing:

"whenever a user *publishes* a document by sending it to the vault, automatically all users that are *subscribed* to that document are notified via an asynchronous multicast communication"

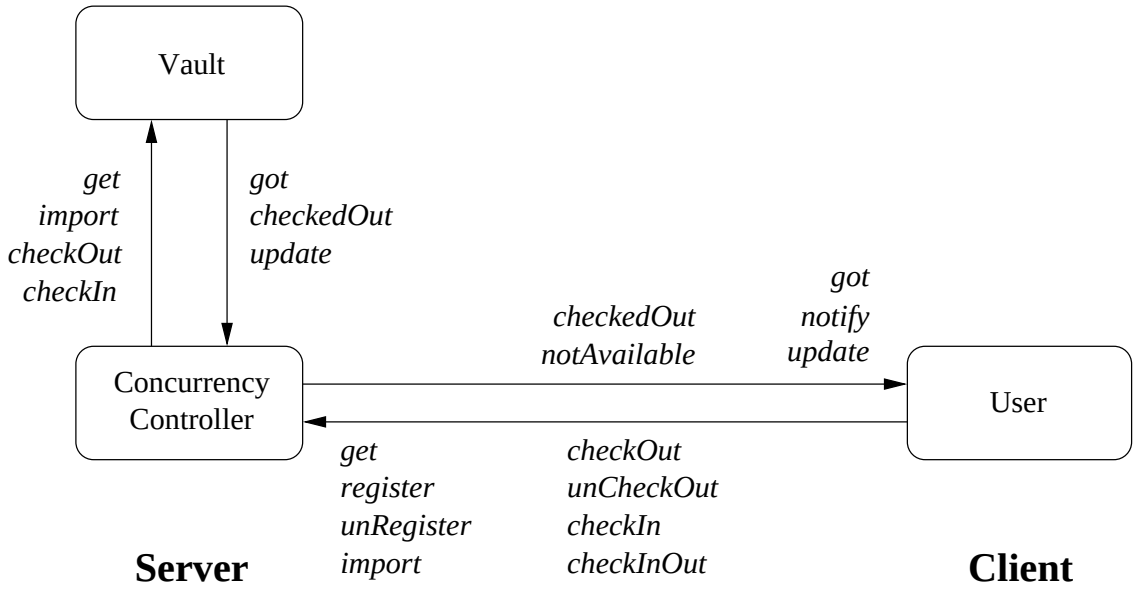
Notion recently much studied in the literature:

\+ "full decoupling of the communicating participants in time, space & flow" [EFGK03]

− generally difficult to verify [GKK03,ZGB03]

Aim: formally model & verify the addition of a publish/subscribe notification service to TT

The TT Protocol



Every user can *(un)subscribe* to a document by an explicit *(un)Register* or by an implicit *get*

Every user subscribed to a document receives:

- a *notify* the moment in which that document is checked out by another user

- an *update* the moment in which another user has returned that document to the vault via an *unCheckOut*, a *checkIn*, or a *checkInOut*

Model Checking

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

Very hard in standard ways (like, e.g., 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 a 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.J. Holzmann
- formal verification of distributed systems specified in Promela
- verifies 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

LTL — Linear Temporal Logic

- a propositional logic with temporal operators
- heavily used for specifying liveness properties
- property must hold always/eventually/until/...

Example: Excerpt from User Process

```
proctype User(byte id) {
  byte edit[numFiles], registered[numFiles];
  bool waitingForCheckedOut = false;
  do
  :: (!waitingForCheckedOut) ->
       userToCC!get,id;
       doneGet: skip;
       ccToUser[id]?got;
       registered[0] = true
  :: ...
  :: (!edit[0] && !waitingForCheckedOut) ->
       waitingForCheckedOut = true;
       userToCC!checkOut,id
  od }
```

e.g.: user can always eventually *get* a document

In LTL: [] <> User[pid]@doneGet

id User process label

Most Important Assumptions

Very low probability of competing user requests

$\Rightarrow$ most communication by handshake channels

There is only one document (file 0) in the vault

$\Rightarrow$ users currently cannot *import* any document

The *notify & update* action are always enabled

$\Rightarrow$ a UserAdmin process deals only with those

No message is ever lost

Validation with Spin

All verifications were performed by running Spin Version 4.1.3 on a SUN Netra X1 workstation with 1000 Mbytes of available physical memory

Full statespace search for deadlock states:

users	state vector	depth reached	errors
2	84 byte	4423	0
3	108 byte	434033	0
4	132 byte	10484899	0

users	memory used	runtime	flags
2	37.574 Mbytes	1.3	
3	114.783 Mbytes	3:06.5	
4	916.095 Mbytes	8:18:36.5	-DMA = 28

(the runtime is given as hours:minutes:seconds)

Correctness Criteria of TT Protocol

Concurrency Control

1) every lock request is eventually answered
2) per file only one user at a time may possess a lock
3) every file lock is eventually released
4) a file lock is not released after a *checkInOut*

Awareness

1) no user receives (a) a *notify* or (b) an *update* w.r.t. a file if not registered for it
2) every *checkOut* eventually leads to a *notify* to all (and only those) users registered for checked out file
3) every *unCheckOut*, *checkIn* & *checkInOut* eventually leads to an *update* to all (and only those) users registered for the respective file

Denial of Service

no user is forever denied a service

$\Rightarrow$ formalise in LTL & verify with Spin ! (3 users)

CC-1: Respond to Lock

"every lock request is eventually answered"

[] (CC[2]@doneCheckOut ->

<> (CC[2]@doneCheckedOut ||

CC[2]@doneNotAvailable))

Spin: valid ! (± 15 min.)

! (<> CC[2]@doneCheckOut)

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

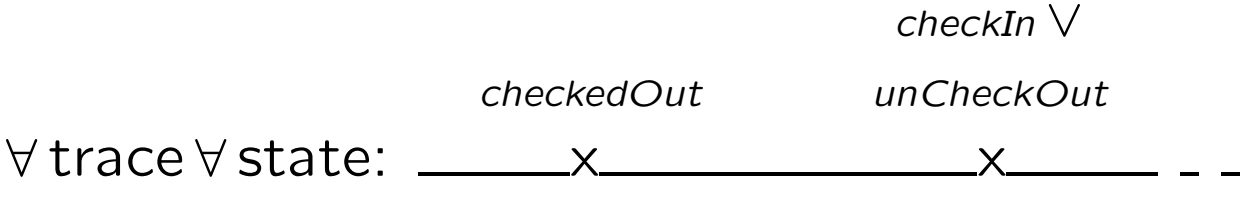
CC-3: Release File+Lock

"every file lock is eventually released"

[] (CC[2]@doneCheckedOut −>

<> (CC[2]@doneCheckIn ||

CC[2]@doneUnCheckOut))



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

"a user can endlessly perform *checkInOut*"

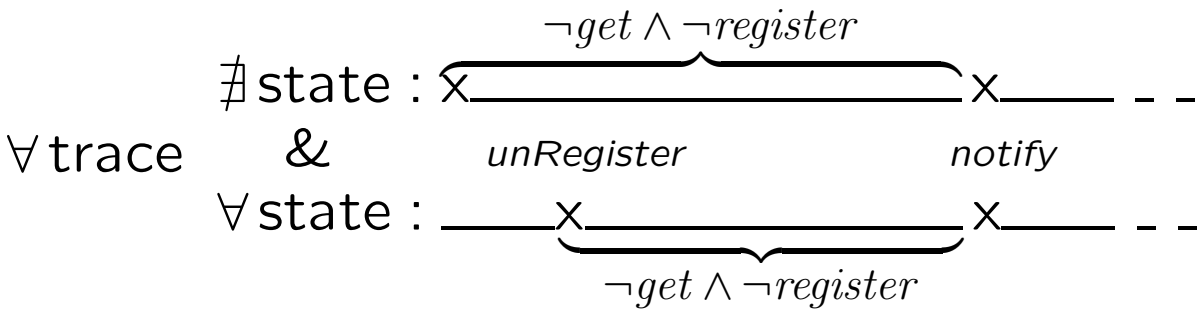
⇒ unavoidable property of TT protocol, which in TT practice is resolved by a 'superuser' !

AW-1a: No Illegal Notify

"no user receives a *notify* w.r.t. a file
if not registered for it"

$$\left.\begin{array}{l} !\,(\,!\,(\text{User[3]@doneGet} \,||\, \\ \qquad \text{User[3]@doneRegister})\,\mathtt{U} \\ \qquad\qquad \text{UserAdmin[4]@doneNotify}) \end{array}\right\} \varphi$$

$$\&\&\ \ [\,]\ (\text{User[3]@doneUnRegister} \mathrel{-\!\!>} \varphi)$$



Spin: valid ! ($\pm$ 20 min.)

(User[3] & UserAdmin[4] refer to user 0, but analogous formulae hold for users 1 & 2)

AW-2: Notify if Registered

"every *checkOut* eventually leads to
a *notify* to all (and only those) users
registered for checked out file"

[] ! ((CC[2]@doneGet0 || CC[2]@doneRegister0)

&& (<> φ) && (! CC[2]@doneUnRegister0 U

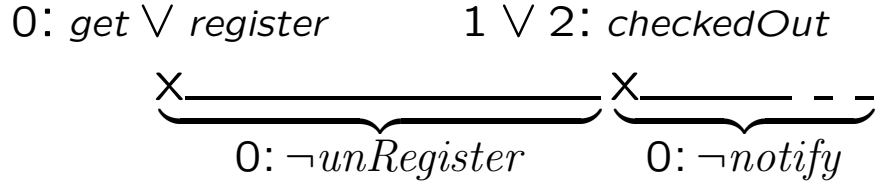
(φ && [] ! CC[2]@doneNotify0))),

where

φ = CC[2]@doneCheckedOut1 ||

CC[2]@doneCheckedOut2

$\nexists$ trace $\nexists$ state :



Spin: valid ! ($\pm$ 40 min.)

(analogous formulae — in which users 0, 1 & 2
change roles — also hold)

DoS: Denial of Service

"no user is forever denied a service"

[] <> User[pid]@doneGet

where pid is 3 (user 0), 5 (user 1), or 7 (user 2)

$\forall$ trace $\forall$ state: ——*get*——x—————————— - -

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

"one user can endlessly keep the CC busy"

$\Rightarrow$ unavoidable property of TT protocol, due to document access based on 'retrial' principle, i.e. no queue or file reservation system in TT

$\Rightarrow$ think3 interested in this for a future release !

Results of Validation with Spin

property	depth	errors	memory used	runtime
CC-1	2703909	0	597.471 Mb	14:57.0
CC-2	434033	0	114.783 Mb	3:06.0
CC-3	310	1	353.759 Mb	0.7
CC-4	434033	0	114.783 Mb	3:06.0
AW-1a	3071518	0	539.769 Mb	21:22.1
AW-1b	3057025	0	558.508 Mb	22:45.4
AW-2	3338868	0	967.955 Mb	39:22.2
AW-3	4183223	0	925.049 Mb	38:57.6
DoS	123	1	33.759 Mb	0.1

(the runtime is given as hours:minutes:seconds)

+ concurrency control & awareness aspects of the TT protocol augmented with a publish/subscribe notification service well designed !

− 'superuser' required to force a user to ever return a checked out file to the vault !

Conclusions

- case study on formalisation & verification of concurrency control & distributed notification aspects of groupware protocol underlying TT

- show feasibility & usefulness of model checking when verifying groupware protocols in general

- among first successful applications of exhaustive model checking to verification of publish/subscribe notification services in groupware

- think3 intends to use specification as basis for planned implementation of such services in TT

Future Work

- `numFiles` > 1 in specification of TT protocol

- abandon file access based on 'retrial' principle (i.e. handshake instead of buffered channels)

$\Rightarrow$ initial verifications show feasibility !

- extend publish/subscribe notification service so that user who checks out a file is informed automatically of existing outstanding copies

- abandon assumption 'no message is ever lost' (e.g. tag messages or send redundant copies)

- perform qualitative & quantitative verification (e.g. stochastic process algebras & automata)

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