Egon Börger (Pisa)

Lift Control Problem

A Ground Model and Refinement Exercise

Università di Pisa, Dipartimento di Informatica, boerger@di.unipi.it

Requirements (N.Davis, 1984)

Design the logic to move n lifts bw m floors, and prove it to be well functioning, where

- Each lift has for each floor one button which, if pressed, causes the lift to visit (i.e. move to and stop at) that floor.
- Each floor (except ground and top) has two buttons to request an up-lift and a down-lift. They are cancelled when a lift visits the floor and is either travelling in the desired direction, or visits the floor with no requests outstanding. In the latter case, if both floor request buttons are illuminated, only one should be cancelled.
- A lift without requests should remain in its final destination and await further requests.
- Each lift has an emergency button which, if pressed, causes a warning to be sent to the site manager. The lift is then deemed out of service. Each lift has a mechanism to cancel its out of service status.

The involved objects (model signature)

- $Floor$ with $button(floor, dir)$ where $dir \in Direction = \{up, down\}$, $bottom, top \in Floor$, previous/next floor functions +1,-1
- $Lift$ with
  - $button(floor)$ for each $floor \in Floor$
  - $emergency$ button
  - $mode \in \{halting, moving\}$
  - $curFloor \in Floor, curDir \in Direction$

Pressing of buttons enters into the model by *shared locations*:

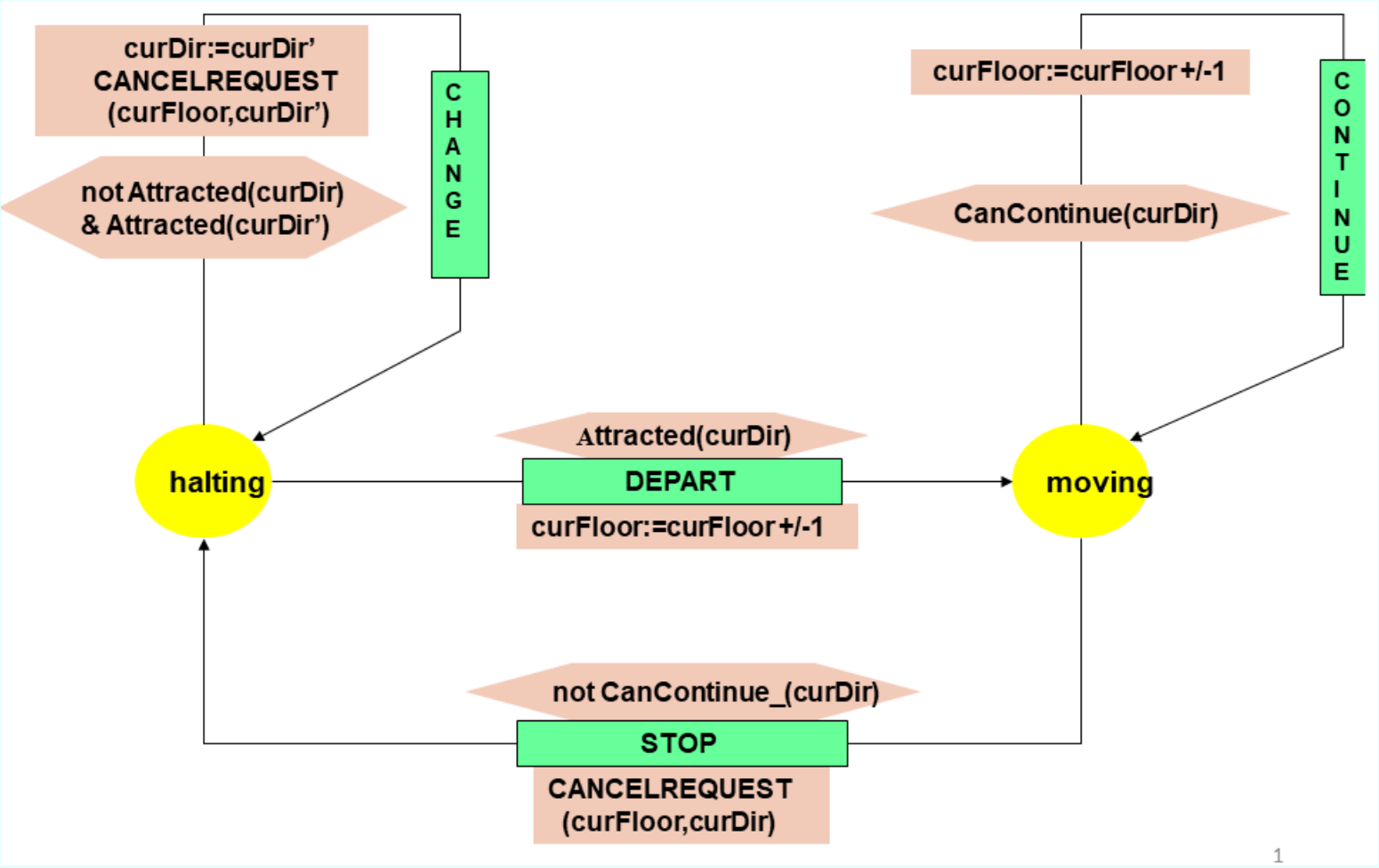
$HasToDeliverAt(floor)$ **iff** $Pressed(button(floor))$

$IsCalledFromTo(floor, dir)$ **iff** $Pressed(button(floor, dir))$

These locations are:

- read (and reset/cancelled) by lifts
- set/pressed by users (the environment)

Behavioral ground model (Lift)



with four actions Depart, Continue, Stop, Change[1]

[1] Figure modified from AsmBook, © 2003 Springer-Verlag Berlin Heidelberg, reused with permission

Communication with environment (reading input)

$Attracted(up)$ **iff forsome** $floor > curFloor$ -- same with down, $<$

$HasToDeliverAt(floor)$ **or** -- internal $button(floor)$ request

forsome $dir \in \{up, down\}$ $IsCalledFromTo(floor, dir)$

-- external $button(floor, dir)$ request

$CanContinue(dir)$ **iff** $Attracted(dir)$ **and**

not $HasToDeliverAt(curFloor$ **and** -- no internal request

not $IsCalledFromTo(curFloor, curDir)$ -- no external request

-- priority: keep direction of travel

NB. Keeping the direction of travel as long as $Attracted(dir)$ easens the 'well functioning' (correctness) proof

- namely that each request is served (i.e. each $Pressed$ $button(floor)$ or $button(floor, dir)$ triggers a lift to move to $floor$).

Resetting requests once they have been served

$\textsc{CancelRequest}(floor, dir) =$

$HasToDeliverAt(floor) := false$ -- 'cancel' $button(floor)$

$IsCalledFromTo(floor, dir) := false$ -- 'cancel' $button(floor, dir)$

Wlog assume that initially $mode = halting$, $curFloor = bottom$, $curDir = up$ and $HasToDeliverAt, IsCalledFromTo$ everywhere false.

Constraint: $HasToDeliverAt(curFloor) = false$ when $mode = halting$

Constraint: $IsCalledFromTo$ false for $(bottom, down), (top, up)$ and for $(curFloor, curDir)$ when $mode = halting$.

Analysis of possible runs (of one lift)

- L1. Non-empty runs (from the initial state) have the form
  $(\textsc{Depart}\ \textsc{Continue}^*\ \textsc{Stop})^+$
  $(\textsc{Change}\ (\textsc{Depart}\ \textsc{Continue}^*\ \textsc{Stop})^*)^*$
- L2. Moving from any reachable state, the lift moves floor by floor to the farest point of attraction in its direction of travel where, after at most $|Floor|$ steps, it $\textsc{Stop}$s and then either terminates - namely iff it is not attracted in any direction - or it $\textsc{Change}$s direction and moves into the oppositve direction.
- L3. When moving to the farest point of attraction in its direction of travel, the lift $\textsc{Stop}$s at each floor where it is attracted, wrt its direction of travel, and turns off the (internal) delivery request and the (external) call from that floor to go into the current direction of travel. When it $\textsc{Change}$s, it turns off the (external) call from its current floor to go into the new direction of travel .

Correctness proof

- To show:
  - All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel.
  - All requests for lifts from floors must be serviced eventually, with all floors given equal priority.
- Proof follows from the three lemmas above, since every internal request from within a lift, and every external request, make the lift being eventually attracted in the requested direction.
- NB. The proof does not exclude real-life situations with crowded lifts, where requests may be satisfied logically, but the lack of capacity prevents people from entering the lift. This problem has no logical solution, and should be solved providing more capacity (larger bandwith)

Adding emergency trigger: a conservative refinement

- Add out-of-service entry rules
- Restrict LIFT by error guard

if $mode \in \{halting, moving\}$ **and** $Pressed(emergency)$

then

$mode := outOfService$

EMITWARNING

else LIFT

Write ERRORHANDLER to be $Triggered$ in $mode = outOfService$ and such that when the error is repaired, it calls back the LIFT (presumably in $mode = halting$ or in its initial state).

NB. This refinement type has been used extensively for modeling Java and the JVM, in particular for exception handling. See:

- R. Stärk, J. Schmid, E. Börger Java and the Java Virtual Machine: Definition, Verification, Validation

Introducing scheduling for multiple lifts

- Problem: all lifts attracted by every external call from a floor to a direction.
- Solution: a scheduler assigns ONE Lift to each external call.
- Model refinement idea: guarantee that $IsCalledFromTo(floor, dir)$ becomes true only for the scheduled lift:
  - $IsCalledFromTo$ becomes a scheduler/lift interface
    - not any more defined by $Pressed(button(floor, dir))$
  - a new shared predicate $ExternalCall(from, to)$ is introduced for the communication between scheduler and environment.

The scheduler will make its choice among the lifts which are $AppropriateFor$ the requesting $floor$ and the direction

- e.g. near to the $floor$, ready to move in direction (in particular not $outOfService$), possibly not crowded, etc.

To express this computation we use a (here not furthermore specified) $select$ion function.

A LiftPool managed by a LiftScheduler

LiftPool = -- synchronized form

  forall $l \in Lift$ Lift_l -- instance Lift_l of Lift

  LiftScheduler

where

  LiftScheduler =

    if $TriggerFrom(floor, dir)$ **then**

      let $lift = select(\{l \in Lift \mid AppropriateFor(l, floor, dir)\})$

        $IsCalledFromTo_{lift}(floor, dir) := true$

        $\text{Consume}(TriggerFrom(floor, dir))$

  $TriggerFrom(floor, dir)$ **iff** $Pressed(button(floor, dir))$

NB. Since $IsCalledFromTo$ is shared with LiftScheduler and not any more defined by $Pressed(button(floor, dir))$, one should add to CancelRequest to also reset $button(floor, dir)$ (and maybe to inform the scheduler that the assigned task has been performed).

Adding optimization requirements for LiftPool

- *schedule non-crowded lifts*: given the needed info on being $Crowded$ (e.g. via weight sensors), it suffices to restrict the $select$ion
  - by including into $AppropriateFor(l, floor, dir)$ the constraint that **not** $Crowded(l)$
- *lifts reserved for a section* $[n, m] = \{n, n+1, \ldots, m\}$ and 1:
  - setting $HasToDeliverAt_{lift}(floor) = false$ for each other $floor$ (since for them no internal request buttons are available in such $lift$s)
  - refining the LiftScheduler $select$ion for each section $button(floor, dir)$ request
    - i.e. $floor \in \{1\} \cup [n, m]$ except $dir = up/down$ if $floor = n/1$)

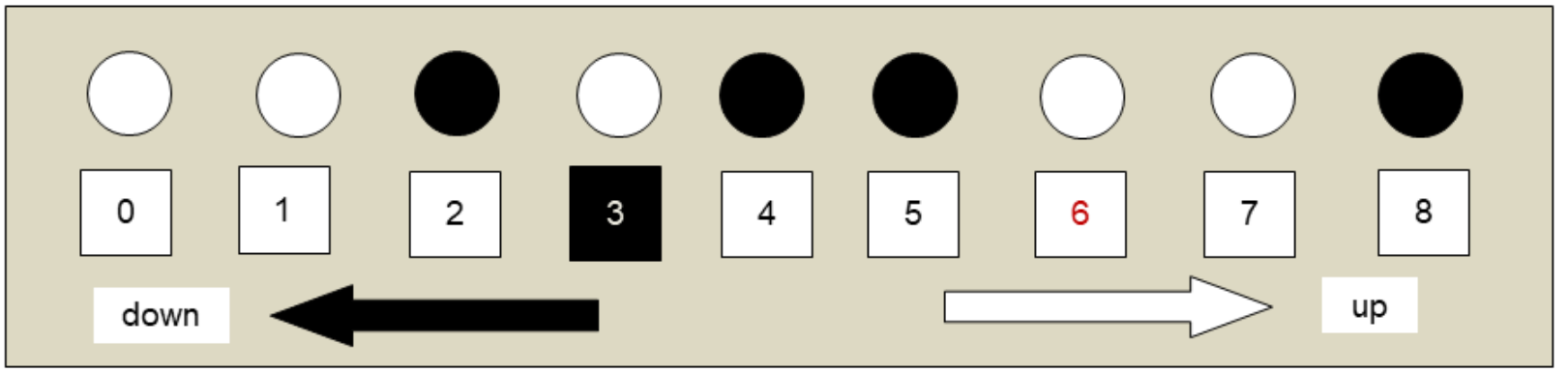
    to assign such requests to a $LiftReservedFor(l, [n, m])$
    - by including into $AppropriateFor(l, floor, dir)$ for each section $button(floor, dir)$ that $l \in LiftReservedFor(l, [n, m])$

NB.The correctness property is relativized to the served section.

Additional info panel requirement

(From M. Jackson 2001, Sect.7.2,7.3.): Design an info panel in the lobby, to show waiting guests where the lift is at any time, so that they will know how long they can expect to wait until it arrives.

- The panel has two lamps *for each floor*.
  - There is a *square lamp* to show that the lift is at the floor, and a *round lamp* to show that there is a request outstanding for the lift to visit the floor.
- In addition there are *two arrow-shaped lamps* to indicate the direction of travel.



Data refinement for info panel requirement

To model the display requirement, which is formulated only for one lift, it suffices to add appropriate derived locations to represent the lamps:

$currentFloorDisplay(f) = on$ **iff** $f = curFloor$

$currentDirectionDisplay(d) = on$ **iff** $d = curDir$

$currentRequestDisplay(f) = on$ **iff** $HasToDeliverAt(f)$ **or**

forsome $dir \in Direction \;\; IsCalledFromTo(f, dir)$

Obviously, the $currentRequestDisplay$ could be further refined to indicate also the direction for which the lift $IsCalledFromTo$.

NB. This is a pure data refinement.

Exercise. Add opening and closing doors as (atomic/durative) action with error handling in case doors do not open/close.

References to related models

- E. Börger, R. Stärk: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer-Verlag 2003 (Ch.2.3)

For a solution in B which inspired the above ASMs see

- J.-R.Abrial: The B-Book . Cambridge University Press 1996 (Sect. 8.3.)

For the Lift info panel requirement see

- M. Jackson: Problem Frames. Analyzing and structuring software development problems. Addison-Wesley 2001

For a Petri net solution see section 4, in particular Fig. 26 pg. 87, of

- W. Reisig: Petri Nets in Software Engineering. In: W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Applications and Relationships to other Models of Concurrency. Springer LNCS 255 (1987) pp.63-96.

Copyright Notice