Verified Design of an Automated Parking Garage

Aad Mathijssen   A. Johannes Pretorius

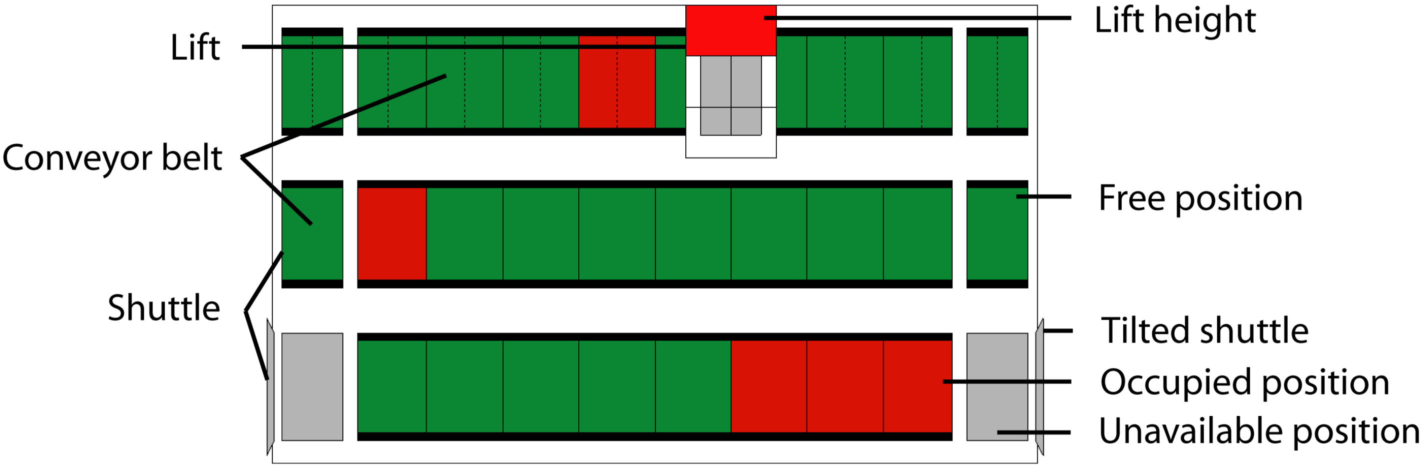
1st February 2006

LaQuSo Laboratory for Quality Software

Introduction

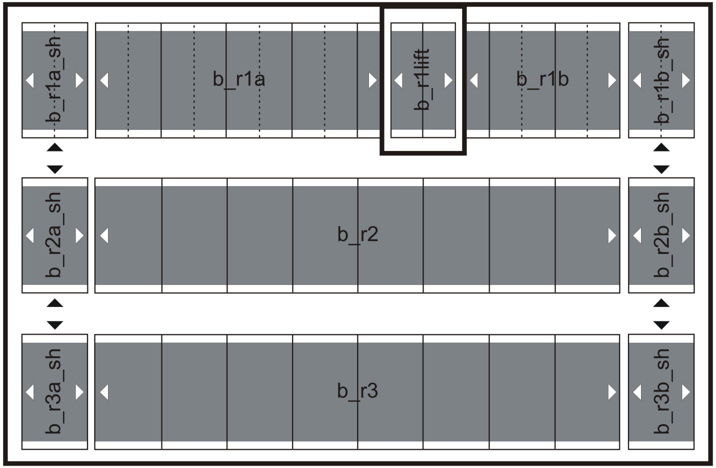
Problem description (1)

Design software for the following system:



Problem description (2)

Design software for the following system:



- 30 parking spots, maximum 29 occupied
- awkward lift position

Approach (1)

Design the software in such a way that *safety* is guaranteed.

Approach (2)

After formulating functional requirements, do *not* start implementing immediately.

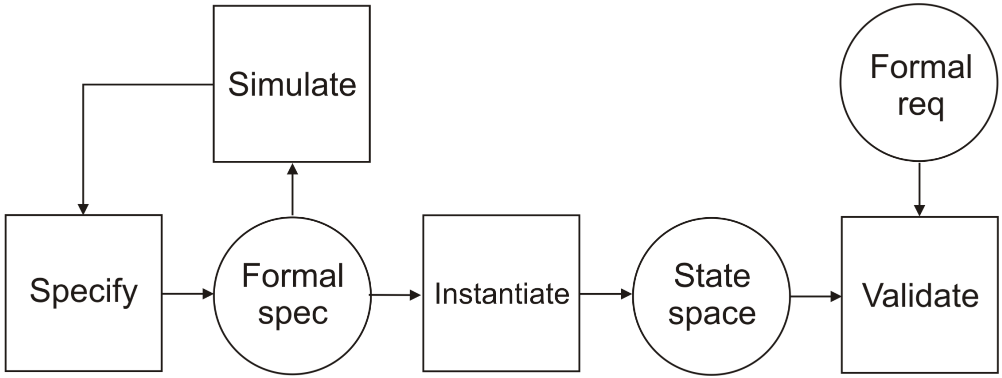
Design a *model* of the software:

- gain *insight* in the system
- detect *errors* in the proposed design
- foundation for *implementation*

Interactions are of primary concern: model *behaviour*

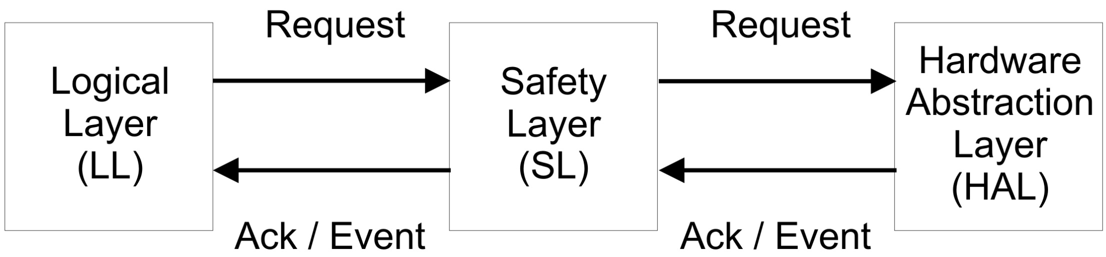
Approach (3)

Pipeline:



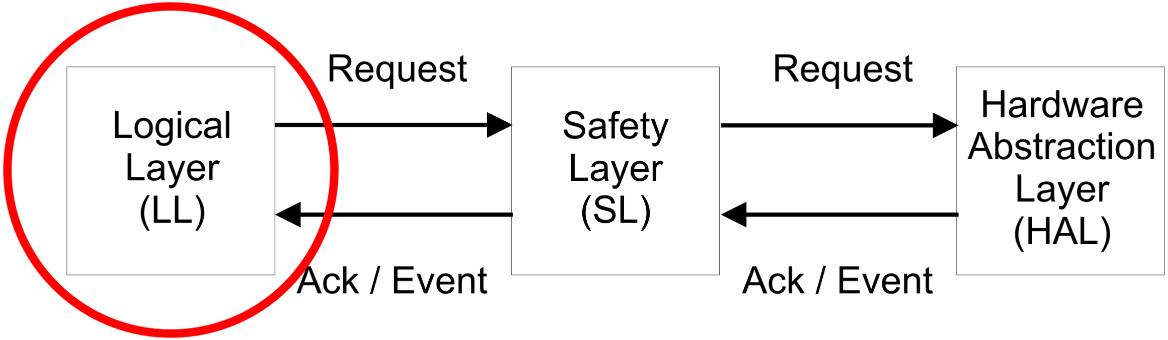
Conceptual system design: architecture

Distinction between three layers in order to maintain separation of concerns:



Conceptual system design: architecture

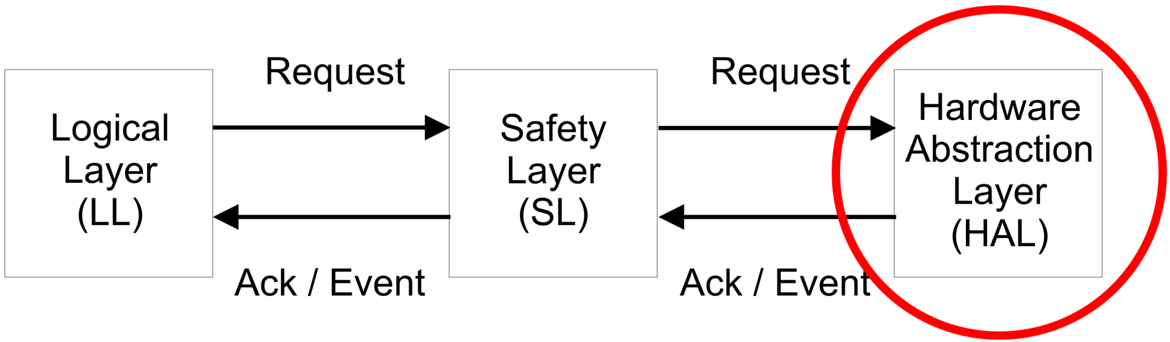
Distinction between three layers in order to maintain separation of concerns:



logical layer: the parking/retrieval algorithm

Conceptual system design: architecture

Distinction between three layers in order to maintain separation of concerns:

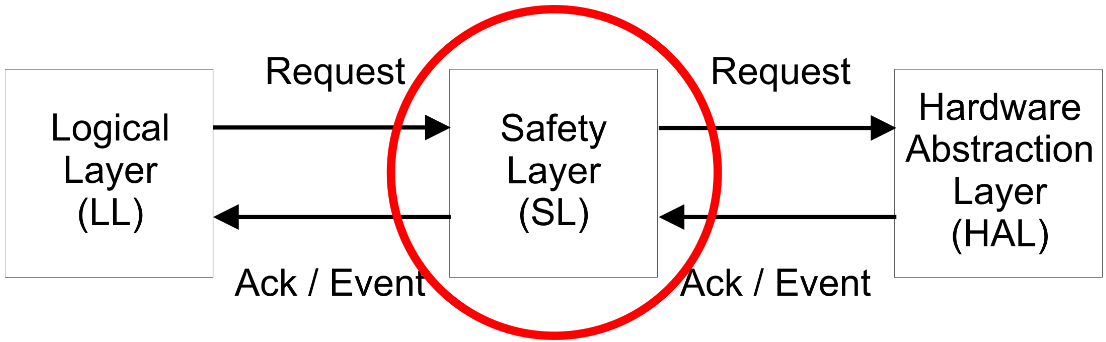


hardware abstraction layer:

- receive and execute instructions; provide feedback on results
- issue events to the safety layer

Conceptual system design: architecture

Distinction between three layers in order to maintain separation of concerns:



safety layer:

- pass messsages between the logical and hardware layer
- **only** if they are safe, deny otherwise

Conceptual system design: data

The following data is communicated between the layers:

- $Event$: addition/removal of cars to/from the system
- $InstructionSet$: instructions that are to be executed *concurrently* by the HAL
- $Instruction$: single instruction that the hardware should execute:
  - $move\_belts(bs, d, ms)$
  - $move\_shuttles(shs, o, d)$
  - $tilt\_shuttle(p, o)$
  - $move\_lift(h)$
  - $rotate\_lift$
- $Result$: indicates the result of executing of a set of instructions

Conceptual system design: actions

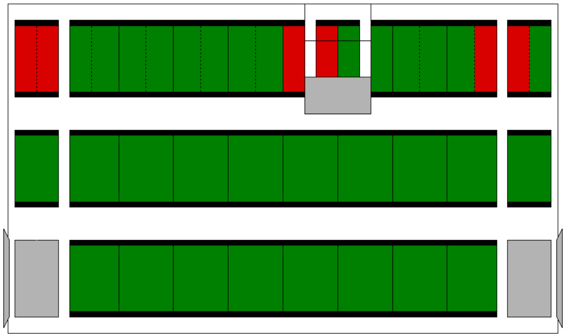
The following actions facilitate communication between the layers:

- $occur(e)$: occurrence of an event e
- $req(is)$: request of an instruction set is
- $ack\_req(is)$: acknowledgement of a request of an instruction set is
- $deny\_req(is)$: deny of a request of an instruction set is
- $ack\_exec(is, r)$: acknowledgement of execution of instruction set is with result r

Safety requirements

Some examples:

1. If a car is moved between belts, both belts should move in the same direction.
2. Cars should not be able to move into walls.
3. When moving shuttles, cars may not be damaged.
4. When moving the lift, cars may not be damaged.



Specification

Focus on safety layer.

Formalise conditions under which it is allowed to execute a set of instructions based on the current state.

A set of instructions is is allowed if:

1. is specifies at least one instruction
2. the instructions in is do not overlap, i.e. the *areas* on which the instructions operate are pairwise disjoint
3. each instruction in is is allowed

Specification: allowed instructions

The instruction *move_belts(bs: BeltSet, d: Direction, ms: MoveSize)* is allowed if:

1. *bs* specifies at least one conveyor belt.
2. All conveyor belts in *bs* directly border each other (this also implies that they must be in the same row).
3. All conveyor belts in *bs* are available (in particular, this applies to belts on the lift and on shuttles).
4. At least one position of size *ms* must be free at the end of the set of belts specified, this free position should be on the side indicated by *d*.
5. In the case that the specified belts are in row r1, there must be no car suspended halfway between the two outer belts of *bs* and their neighbours, if any.

Simulation

Simulation enables us to:

- test the quality of our specification
- construct and analyse potentially dangerous scenarios

Infeasible using initial specification. Reductions were needed:

- abstract from sets of instructions by focusing on single instructions on only

To make simulation more effective:

- abstract from requests and acknowledgements; instead, it is assumed that instructions are executed successfully by the HAL
- build a visualization plugin

Observations

Major complicating factors:

- due to lift position, cars are able to move in half positions
- shuttles can be tilted

Consequences:

- components are much more intertwined
  e.g. a car can be on both the lift and the bordering conveyor belt
- more checks are needed
- complex checks are needed

Verification (1)

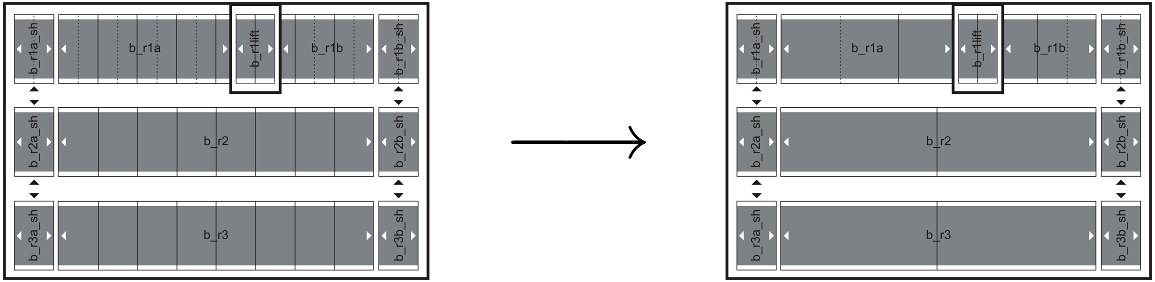
Verification:

- guarantees requirements are fulfilled for each possible system state
- requirements need to be formalised
- model checking is space and time consuming

Unfeasible with the original model: 640 billion states ($6,4 * 10^{11}$)

Verification (2)

Solution: restrict the number of *positions*:



Result: 3,3 million $(3,3 * 10^6)$ states and 98 million $(9,8 * 10^7)$ transitions

Feasible

Verification (3)

Approach:

- augment specification with *error* actions that are triggered when a requirement is not fulfilled
- check the state space on the existence of the *error* actions

Result: no *error* actions were found

Tech Specs

Specification language used: mCRL2 (successor to μCRL)

Specification: 991 lines of mCRL2 code

Verification: 217 lines of mCRL2 code

Visualization: 1583 lines of C++ code

Verification time (real time):

- 5 hours on a cluster of 34 CPUs (3 GHz CPU, 2 GB RAM)
- 35 hours on a single PC (3 GHz CPU, 4 GB RAM)

Time spent: approximately 500 man hour

Conclusions (1)

For systems that interact with their environment, focus on *behaviour*.

Model the behaviour:

- gain *insight* in the system
- detect *errors* in the design
- foundation for *implementation*

Conclusions (2)

Simulation: *confidence* in safety of our model

Visualization:

- *speeds up* simulation
- revealed a number of *errors* in the model
- enhances *communication*

Verification: *prove* safety of our model