# Specification, Analysis and Verification of an Automated Parking Garage

Aad Mathijssen A. Johannes Pretorius

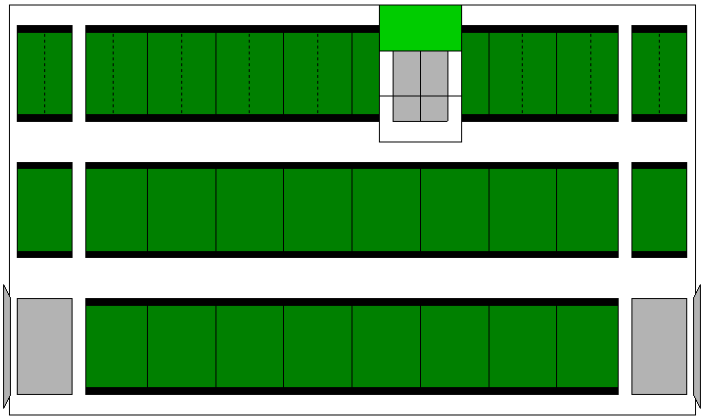
24th November 2005

LaQuSo
Laboratory for Quality Software

# Introduction

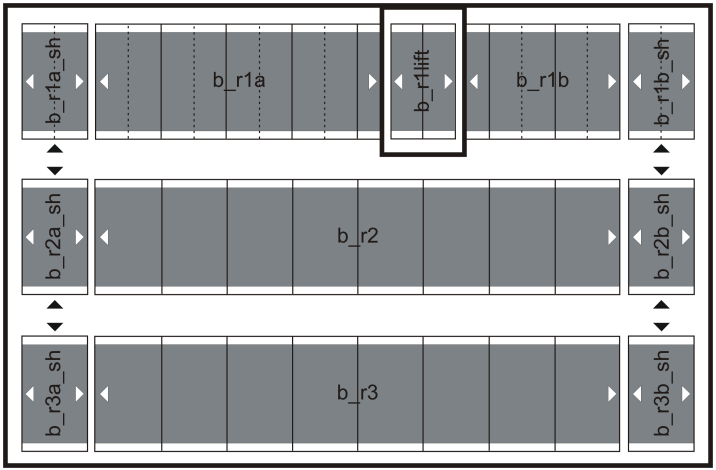
# Problem description (1)

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.

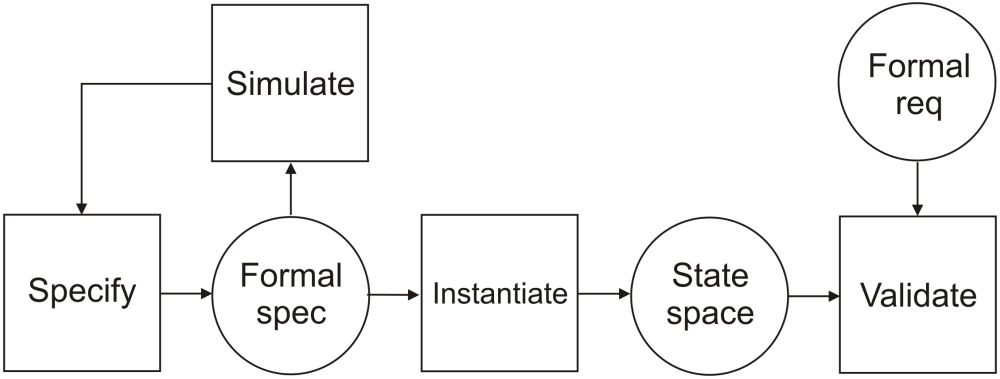
Design a *model* of the software:

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

Interactions are of primary concern: model *behaviour*

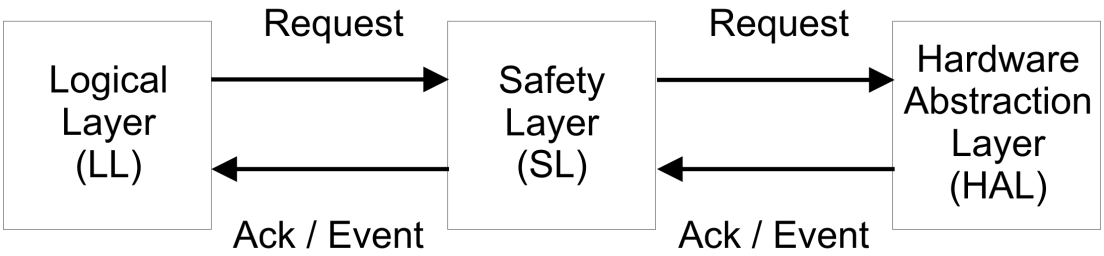
# Approach (3)

Pipeline:



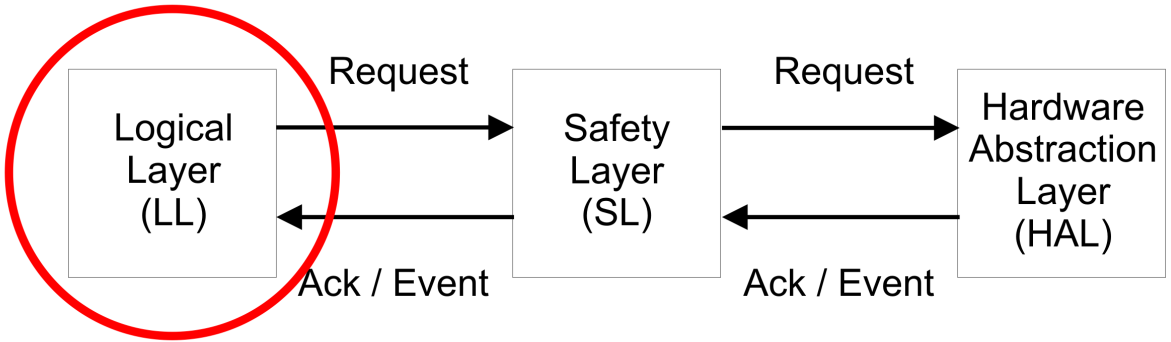
# Conceptual system design: architecture

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



# Conceptual system design: architecture

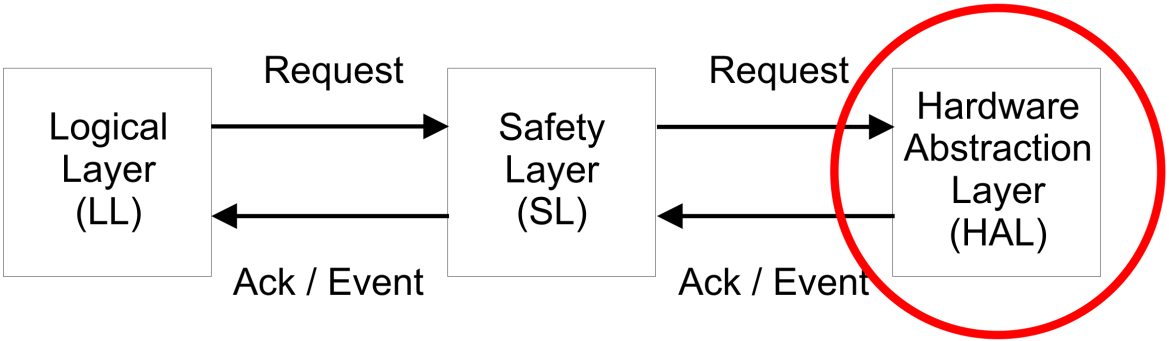
Distinction between three layers in order to maintain separation of concerns:




*logical layer*: the parking/retrieval algorithm

# Conceptual system design: architecture

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

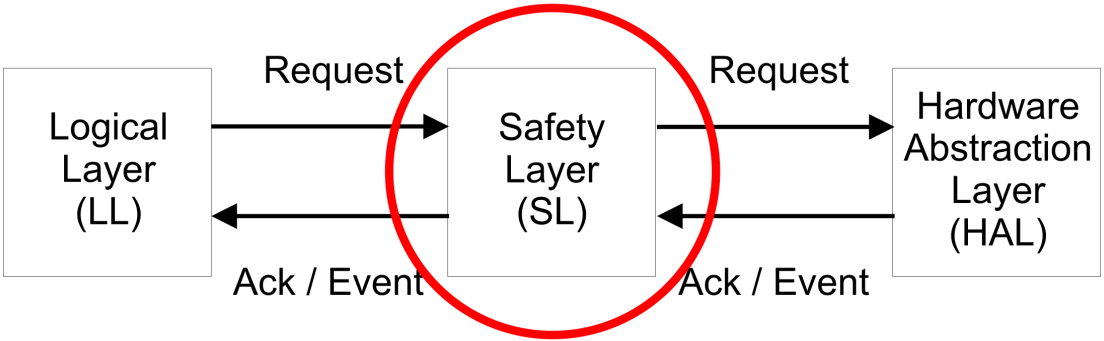



*hardware abstraction layer*:

- 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*:

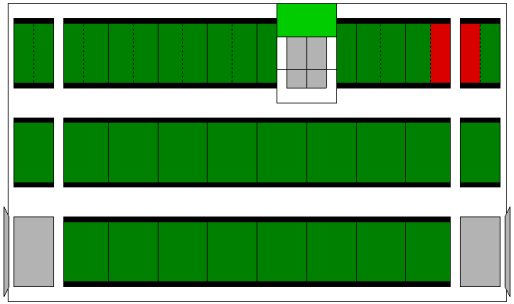
- pass messsages between the logical and hardware layer
- **only** if they are safe, deny otherwise

# Simulation

Simulation enables us to:

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

Custom made visualization plugin makes this much more effective



# Observations

Major complicating factors:

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

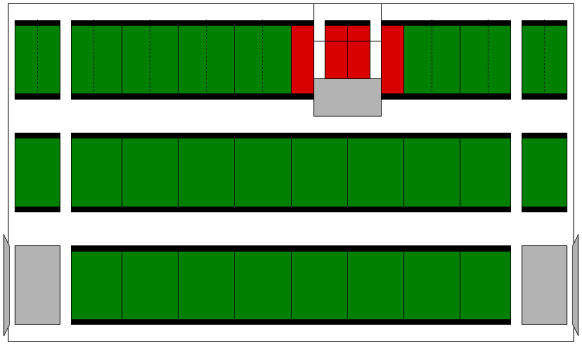
Consequences:

- 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

# Requirements Example

When we want to move the lift, a requirement is:

- there must be no car suspended between the lift and bordering conveyor belts



# Verification (1)

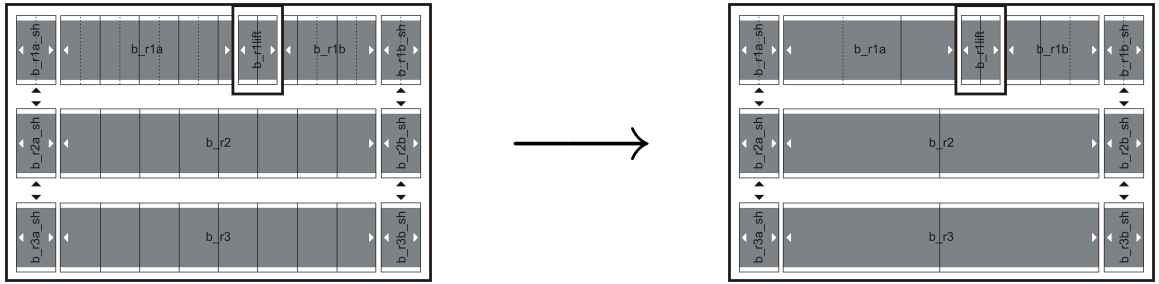
Verification:

- 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

# Tech Specs

Specification language: mCRL2 (Process Algebra + Data)

Specification: 991 lines of mCRL2 code

Verification: 217 lines of mCRL2 code

Visualization: 1583 lines of C++ code

Verification time (real time):

- 2 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