Behaviours and Requirements
for
Cyber-Physical Systems

A pre-formal view

(23 May, 45 mins with discussion)

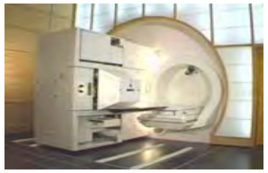
Michael Jackson
The Open University
jacksonma@acm.org

WG2.3 Meeting
Orlando, May 19—23 2014

Examples of cyber-physical systems

- ‘cyber’ = controlling
- ‘physical’ = concrete, not abstract

Radiation therapy



Flight control

Passenger lift

Automotive control

Rotterdam barrier

Industrial press

Car parking

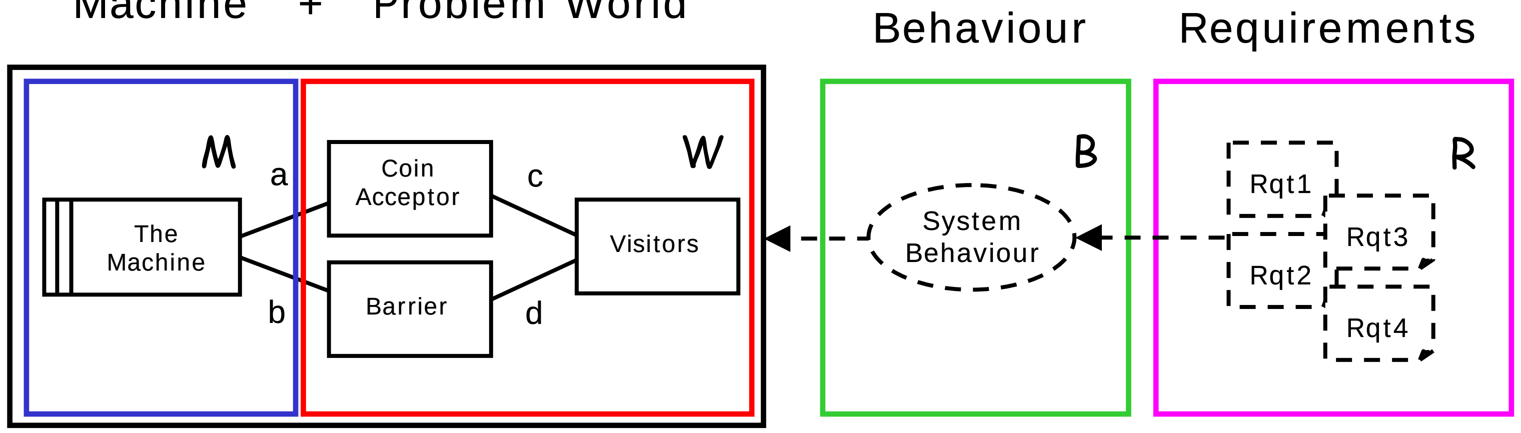
Vending machine

Some dimensions of variation between systems

Criticality, complexity, operating envelope, security, duty cycle, ...

A system and its behaviour

System = Machine + Problem World



The problem world is given, the (software) Machine to be developed

- Executing Machine M ensures behaviour B in problem world W
- Terminating or non-terminating
- Uninterrupted: no suspend/resume; multiple instances over time
- Behaviour is purposeful (cf automotive features)

The development problem:

- (a) Design a behaviour B satisfying R: that is $W,B \mid\sim R$
- (b) Specify a machine M ensuring B: that is $M,W \models B$

Behaviour complexity and constituent behaviours

A complex behaviour combines simpler ‘constituent behaviours’

Constituent behaviours are like automotive ‘driver-assist features’

Different constituent subsets are active at different times

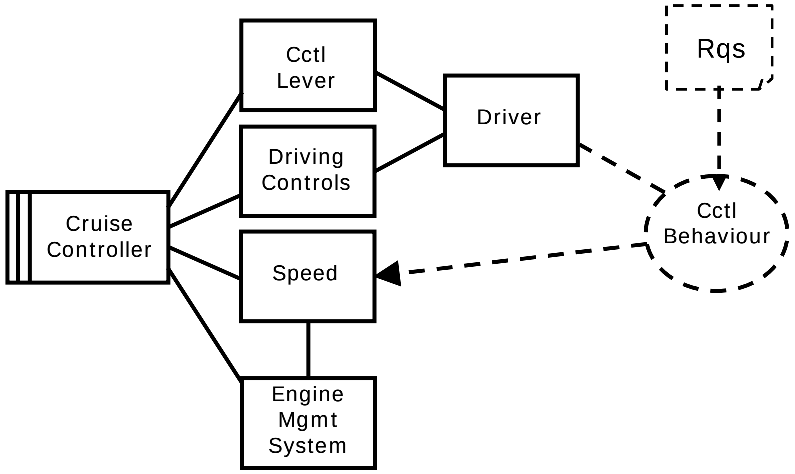
- ABS is monitoring wheel speeds
- Air conditioning is running
- Stop-Start is not running
- Automatic Parking is not running
- Cruise Control is maintaining the driver’s chosen speed
- Lane Departure Warning is watching the lane markings
- Speed Limiting is preventing speed from exceeding 110kph
- Active Suspension is monitoring and smoothing potholes
- …

Behaviour development identifies, designs and combines constituents

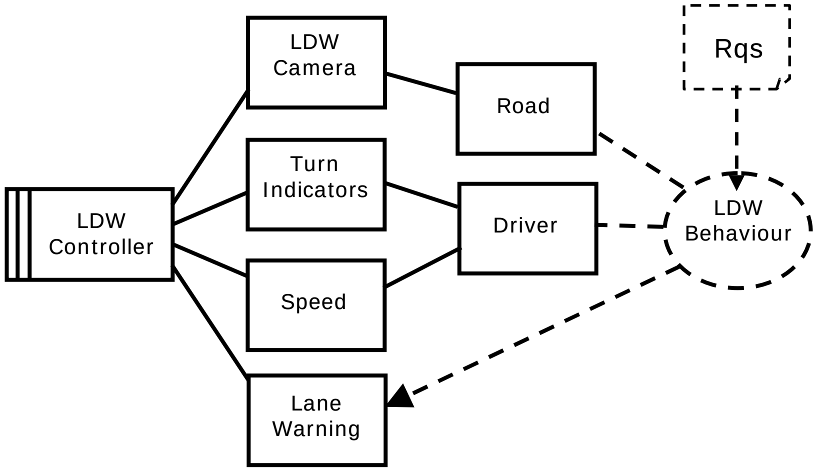
Constituent behaviours viewed as isolated systems

Constituent behaviours are designed in ‘subproblems’

Cruise Control to maintain
the driver’s chosen speed



Lane Departure Warning
to alert to lane indiscipline



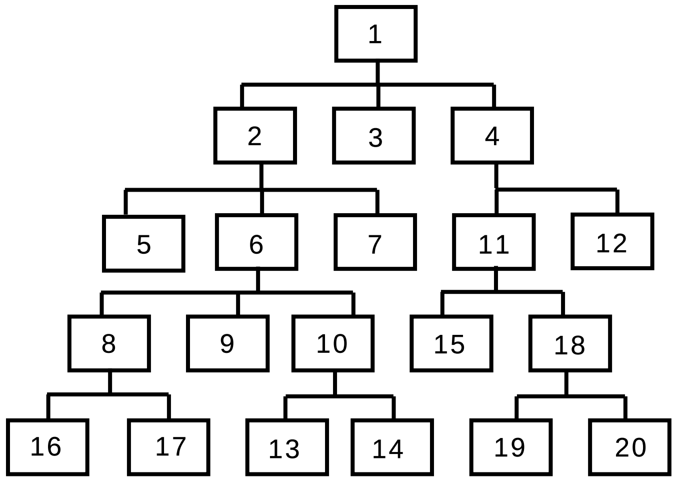
Suproblem = machine, problem world, behaviour, requirements

Like a problem, a subproblem defines a system

Combination with other behaviours is postponed

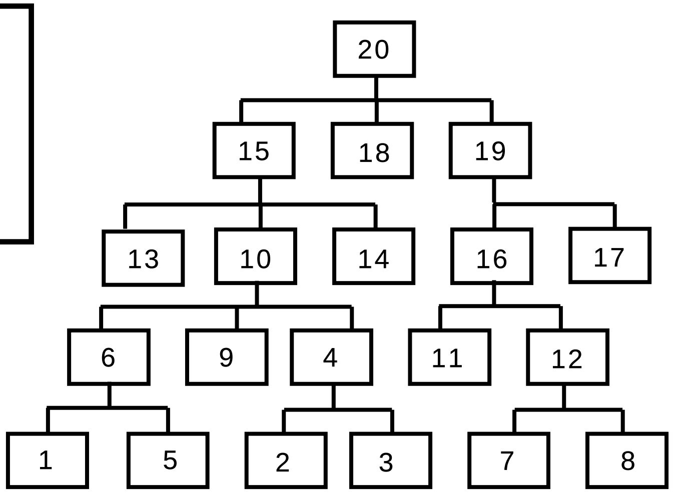
The art of dividing: top-down and bottom-up

Traditional software design assumed hierarchical component structure



Legend:

N = sequence of component identification & understanding



Pure top-down decomposition

Parent components are identified and understood before children

Parent's spec is essentially known; childrens' specs must refine it

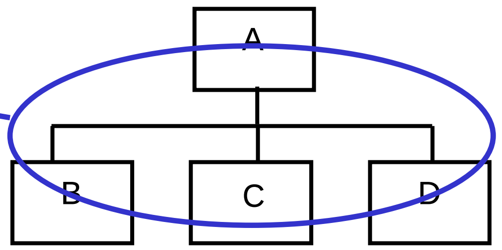
Pure bottom-up assembly

Child components are identified and understood before parents

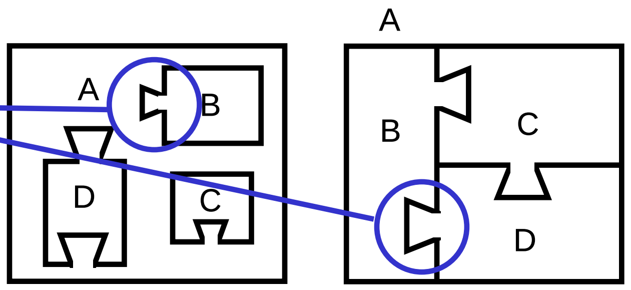
Parent's spec is essentially unknown; children are initially conjectural

The art of recombining: loose structuring

The component connections (eg procedure calls) in the structure define interactions among the components

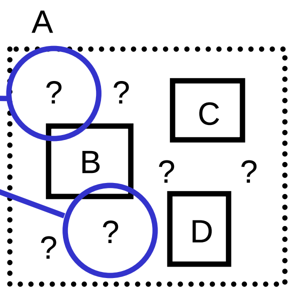


In tight structuring, connections of components are defined (for parent or child components) as the structure is developed



Tight embedded Tight jigsaw

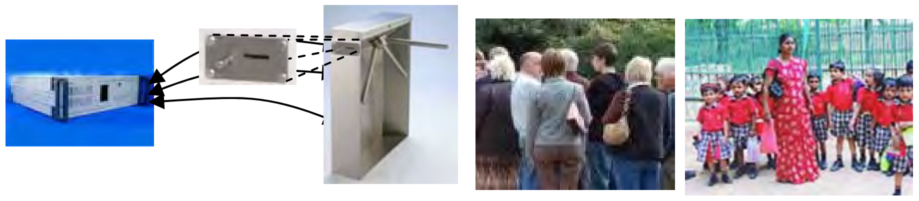
In loose structuring, component connections are analysed and specified in a distinct task, postponed until children are understood



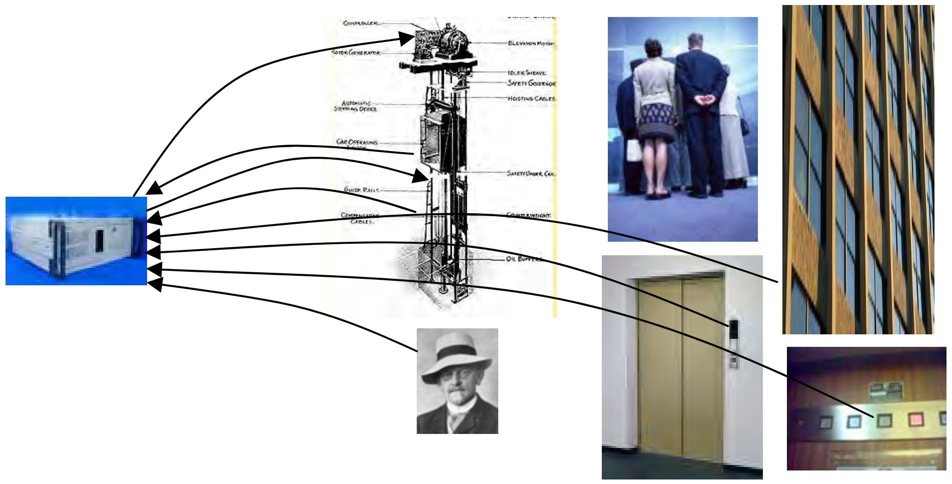
Loose

Two examples

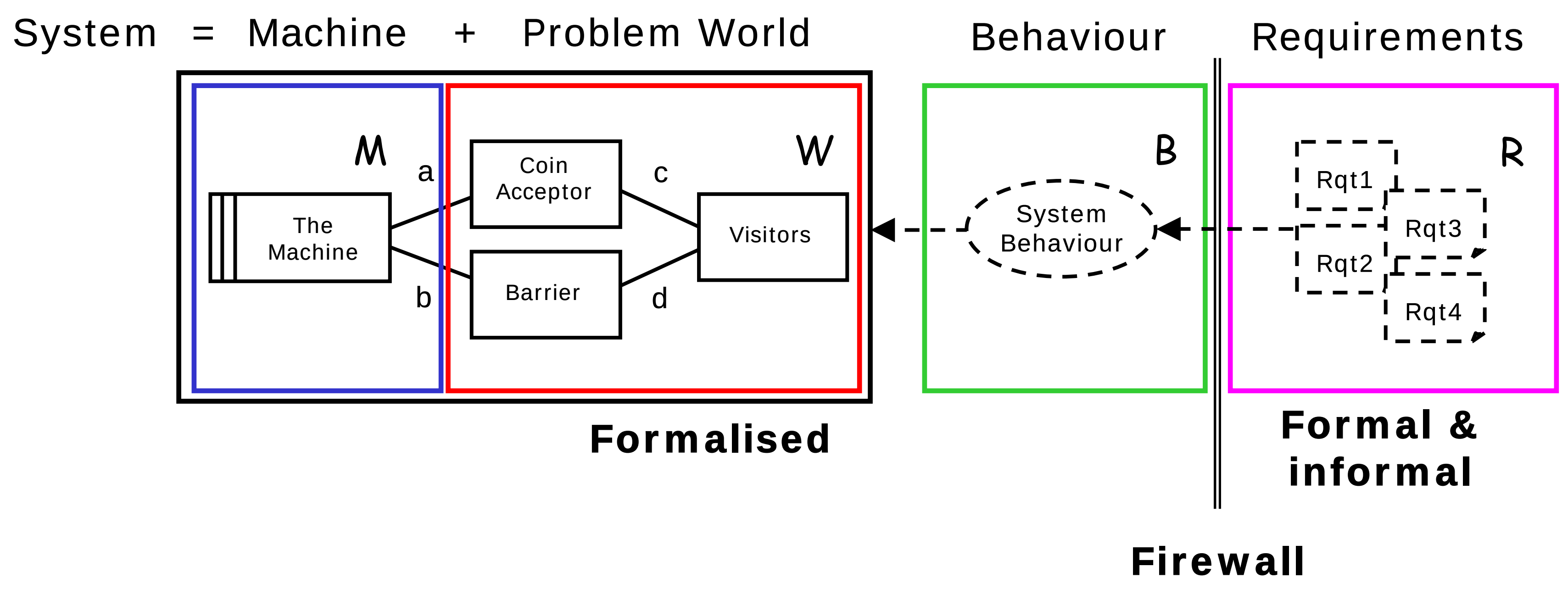
- Zoo Visitor Control
  - Some problem analysis detail, leading to …
  - Top-down identification of constituent behaviours and their combination



- Passenger Lift Control
  - Bottom-up identification of constituent behaviours and some top-down
  - Some combination detail



A firewall



Requirements are properties of system behaviour

- Most are informal and vague or overspecified
- Some lie well outside any possible problem world boundary
- So the 'system validity' goal "W,B |~ R" rests on informal judgments

The minimal 'system correctness' goal "M,W |= B" must be formal

- Rests on formal reasoning and a sound formalisation discipline

Summary and sales pitch

System behaviour is the primary product of development

* It captures exactly the significant effects of the system
* It is exactly the product of software execution

Behaviour is a complex assemblage of constituent behaviours

* 'Elementary' constituent behaviours should be simple
* Simplicity allows perspicuous treatment of concens

Each constituent behaviour is a quadruple (Mb,Wb,Bb,Rb)

* Mb,Wb,Bb,Rb are projections of M,W,B,R at various granularities
* Wb1,Wb2 may be inconsistent (eg: faulty, healthy properties)

Each behaviour Bb is associated with its machine Mb

* Problem world behaviours become 'manageable'
* The behaviour description is transformable to software design

Thank you