Requirements, Behaviours and Software Engineering

Michael Jackson
The Open University
jacksonma@acm.org

RE 2015
Ottawa
28 August 2015

Daniel, Didar and Vincenzo:
Thank you for inviting me!

Everyone:
Thank you for coming to my talk!

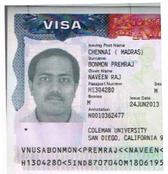
Requirements state what is wanted

- We are about to commission, or build, or buy, or somehow acquire something
    - The requirements state what is wanted, expected or demanded from it

Requirements state what is wanted

US State Department (for visa photo)

- Colour, head 22mm to 38mm high, taken last 6 months, full-face, usual dress, eyes open, no sunglasses, no hat.



VISA

Issuing Post Name
CHENNAI (MADRAS)
Surname
BONMON PREMRAJ
Given Name
NAVEEN RAJ
Passport Number
H1304280
Sex
M
Entries
M
Issue Date
24JUN2013
Annotation
N0010362477

*

COLEMAN UNIVERSITY
SAN DIEGO, CALIFORNIA 9

VNUSABONMON<PREMRAJ<<NAVEEN<
H1304280<5IND8707040M1806195

Computer store customer

- PC laptop, 13” screen, 4GB RAM, 1TB HDD, min 3GHz, USB3 ports, WiFi, 8hr battery, max 1.5Kg, Windows 8.

Ice cream parlour customer

- Medium sugar cone, one scoop crema, one chocolate, no sauce, no sprinkles.

Car dealer customer

- 5-dr hatchback, 1.6l diesel, 50mpg, automatic, trunk 15cu.ft, sunroof, built-in satnav, alloy wheels, 4WD.

So what’s the problem?

- What’s the requirements engineering problem?
  - Why do we need ”Requirements for the Masses”?
- Are the masses dissatisfied?
  - Are they not getting what they want?
- I imagine mass demonstrations …

Requirements for the masses?

- Are the people in these crowds US visa applicants, or customers of computer stores, ice cream parlours, and car dealers?
- No! It's easy to get the photo, laptop, ice creams or cars you want
  - These things are products of normal (standard) design
- The demonstrators are customers of software engineering
  - Most software products involve radical (innovative) design

Normal and radical engineering

Normal:
Toyota
1992

"The engineer … knows at the outset how the device in question works, ... its customary features, and ... properly designed, ... [can accomplish the desired task]."*

- Normal engineering allows exact requirements by standard parameters
  - Standard designs, reliably built, satisfying well understood needs

Radical:
Karl Benz
1885

"The device ... is largely unknown. The designer has never seen such a device before … the problem is to design something that will function well enough to warrant further development."*

- Radical engineering allows vague hopes rather than exact requirements
  - Design is uncertain, outcome unreliable, needs poorly understood
  - Result is difficulty in stating and satisfying requirements

Walter G Vincenti;
What Engineers Know and How They Know It*

What kind of software engineering?

- Engineering OF Software
  - Problems of symbolic computations and symbolic results
  - The system comprises only the computer (Dijkstra even discarded input and output)
- Engineering WITH and BY Software
  - Problems of specifying software to evoke or ensure desired behaviours in the human and physical world
  - The system comprises both the computer and the human and physical world of the problem

Engineering WITH software concerns behaviours in the world

Radiation therapy

Passenger lift

Rotterdam barrier

Car parking

Flight control

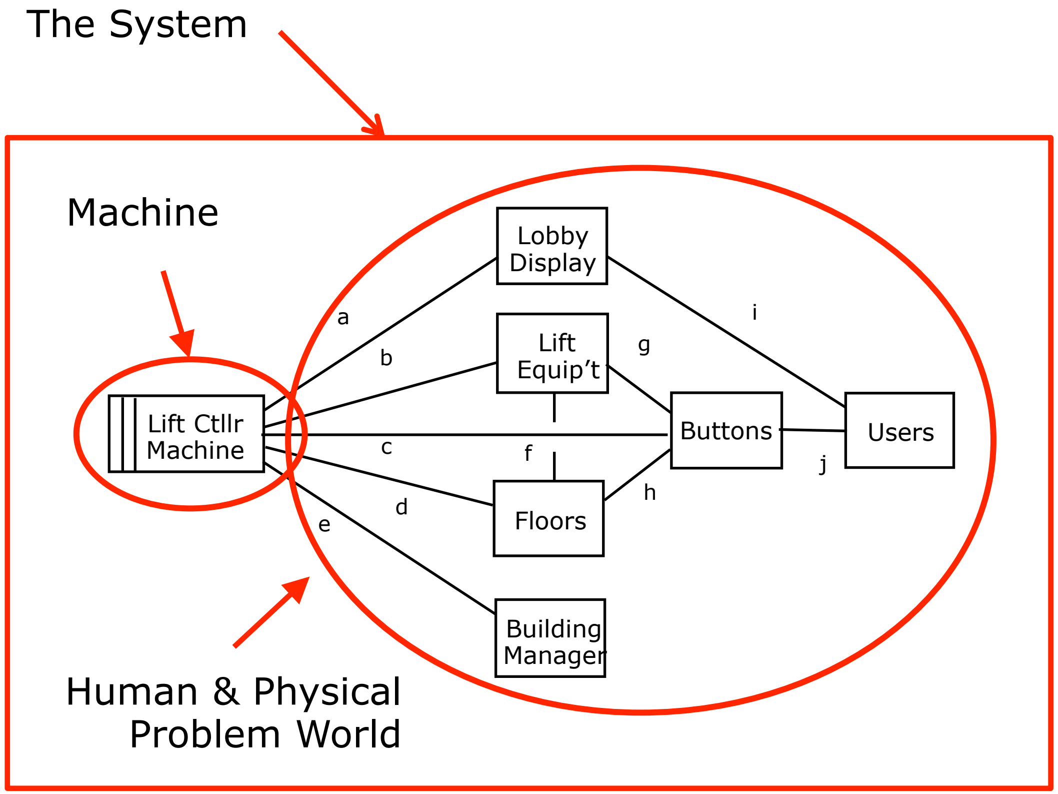
Cruise control

Industrial press

Vending machine

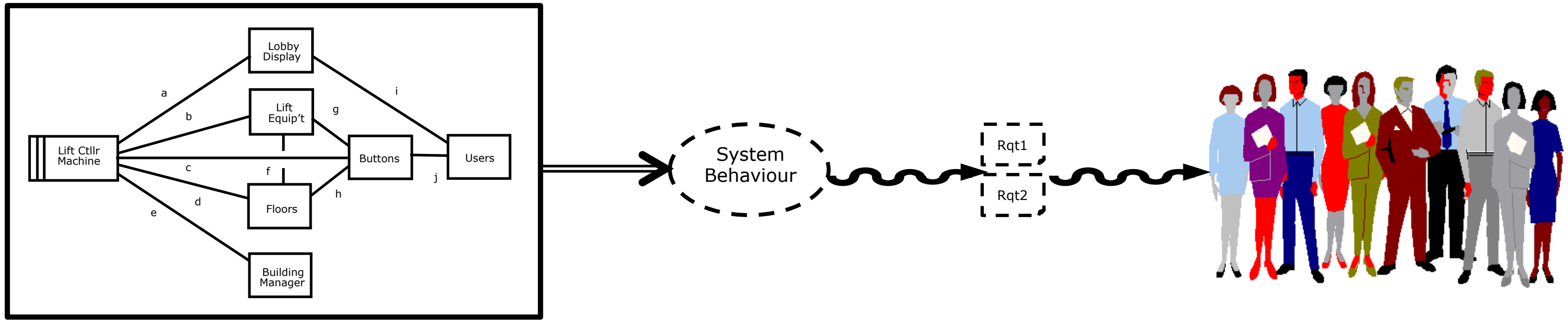
Medical Records

Lending Library



The system behaviour is everything that can happen in the given problem world while it is interacting with the specified software

What is the requirements engineering context here?



The System-to-be

The Stakeholders

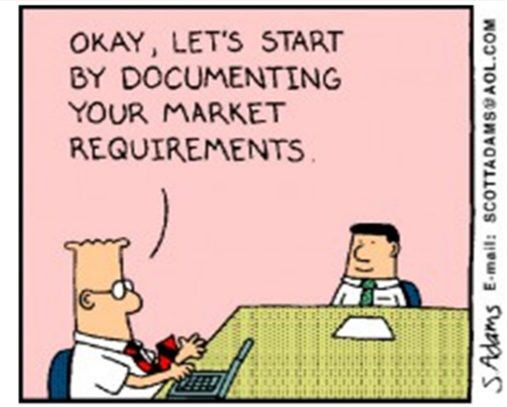
- A specified Machine M guarantees a system behaviour B, cooperating with given properties W of a closed, formalised problem world
- System behaviour B satisfies requirements so far as possible: some may be informal or lie outside the problem world
- A clear distinction of tasks and interests here:
  - Stakeholders state requirements
  - Developers specify machine M
- The tasks are unavoidably interactive and iterative
  - Dilbert and his manager know why!

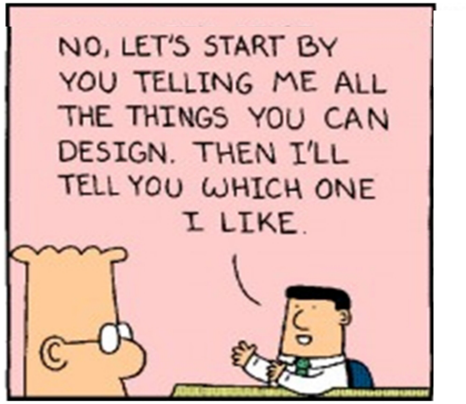
Dilbert and his manager

OKAY, LET'S START BY DOCUMENTING YOUR MARKET REQUIREMENTS.
S. Adams E-mail: SCOTTADAMS@AOL.COM

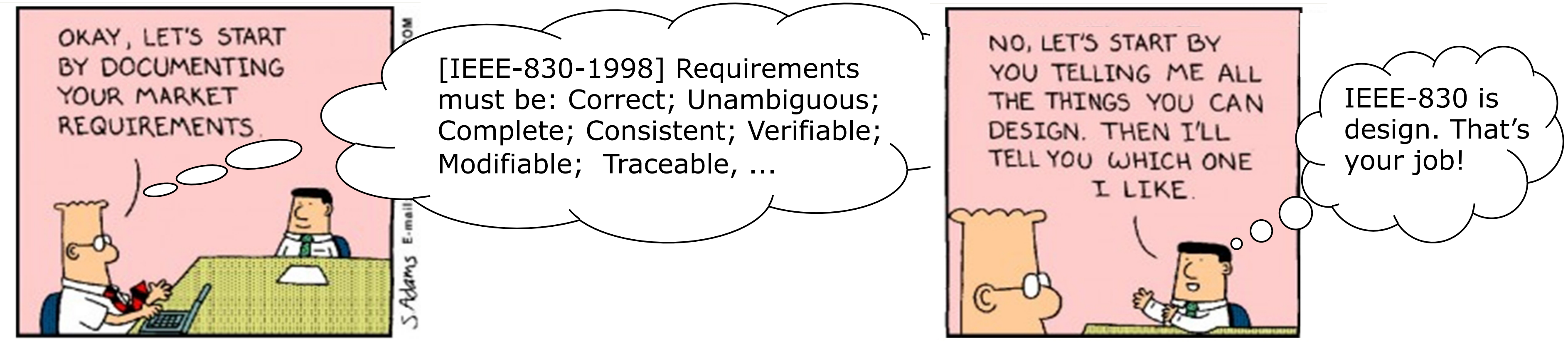
(borrowed from a talk by Jyotirmoy V Deshmukh)

Dilbert and his manager


OKAY, LET'S START BY DOCUMENTING YOUR MARKET REQUIREMENTS.
S. Adams E-mail: SCOTTADAMS@AOL.COM


NO, LET'S START BY YOU TELLING ME ALL THE THINGS YOU CAN DESIGN. THEN I'LL TELL YOU WHICH ONE I LIKE.

Behaviour: a meeting point for developers and stakeholders


OKAY, LET'S START BY DOCUMENTING YOUR MARKET REQUIREMENTS.
S. Adams
NO, LET'S START BY YOU TELLING ME ALL THE THINGS YOU CAN DESIGN. THEN I'LL TELL YOU WHICH ONE I LIKE.

[IEEE-830-1998] Requirements must be: Correct; Unambiguous; Complete; Consistent; Verifiable; Modifiable; Traceable, ...

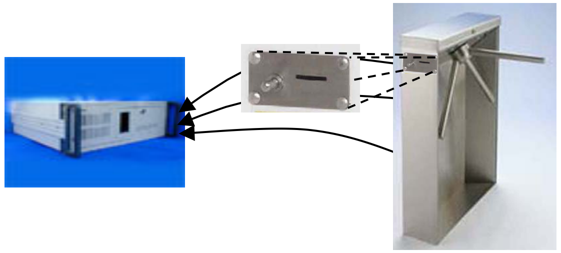
IEEE-830 is design. That's your job!

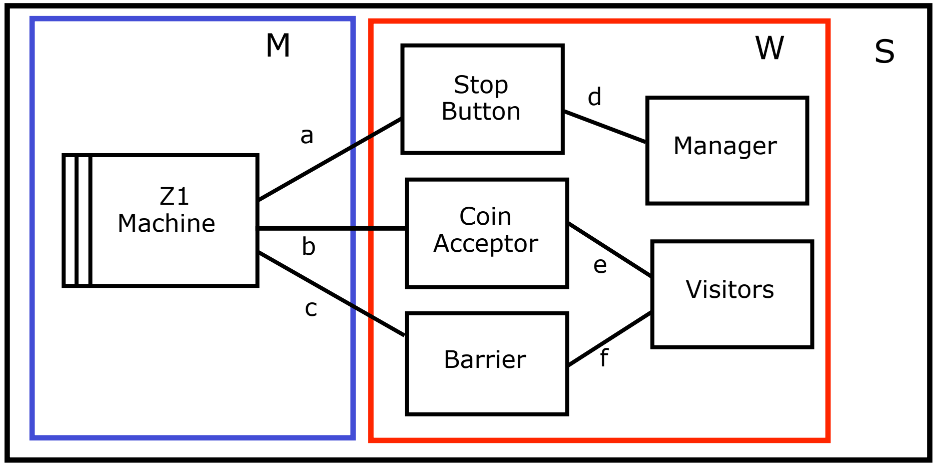
- IEEE-830 is not an ideal: it was a mistake
  - "Requirements should form a detailed arm's-length contract"
  - Unnecessary for normal products; impossible for radical products
- Dilbert's manager is right: IEEE-830 requirements are detailed designs
  - Why should stakeholders (and requirements engineers) do design?
  - State transitions masquerading as requirements are a bad idea!
    (an automotive product line manufacturer had 200,000 requirements)
- The system behaviour is not itself a requirement
  - Most requirements are desired properties and effects of the behaviour

The Zoo Visitor control problem

- Illustration from a small example problem*
  - First, the system (machine + problem world)
  - Second, the [initial] requirements
  - Third, an interactive, iterative development ..
    .. leading to a little behaviour structuring ..
    .. into three behaviours ...
  - .. which must then be recombined

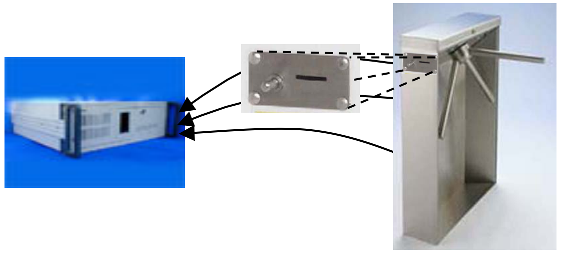
Zoo visitor problem: machine and problem world





- The problem world is closed and reliably formalisable
- The problem world includes human participants
- Properties of B must be calculable from the (eventual) machine specification and the given problem domain properties

Zoo visitor problem: initial requirements



Requirements

R1: “No visitor enters without paying fee (1 coin)”

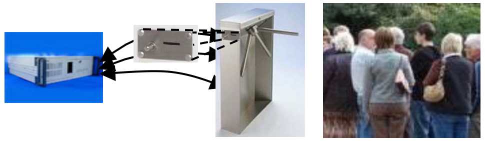
R2: “Here’s what visitors do to enter” (‘use-cases’)

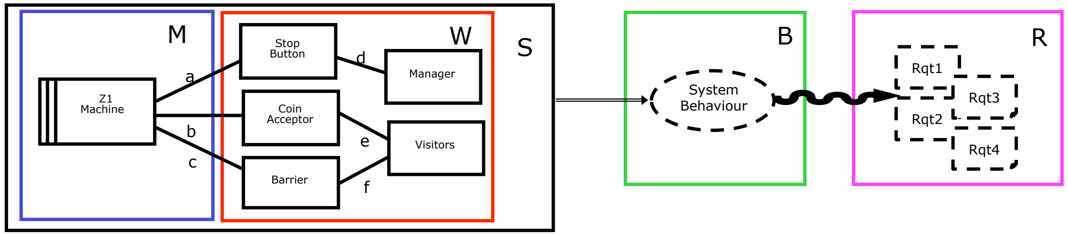
R3: “Turnstile is quick and easy to use”

R4: “Entry system positively attracts new visitors”

R5: “Pressing the Stop button ends system operation”

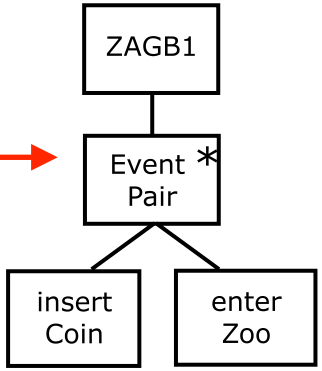
Development: functional requirement, abstract goal behaviour





Development approach

1. Identify the (sole) simple functional requirement
   (R1: "No visitor enters without paying fee")
2. Propose a simple abstract goal behaviour (AGB)
3. Describe given properties of problem domains
4. Specify m/c ensuring a system behaviour B => AGB
5. Check B with stakeholders against all requirements
6. Iterate over 2,3,4,5 as necessary



(a) ‘insert coin’ and ‘enter Zoo’ are poor abstractions

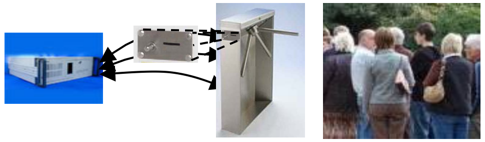
(b) How and when does the behaviour stop?

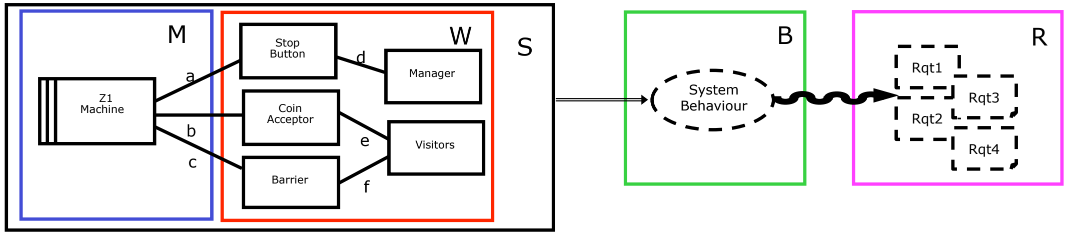
(c) Alternation of payment and entry vs R2 use-case?

(d) Mutual exclusion of payment and entry vs R3, R4

(e) Machine Z1 is too complex to specify and program

Iteration in specifying the machine and the system behaviour





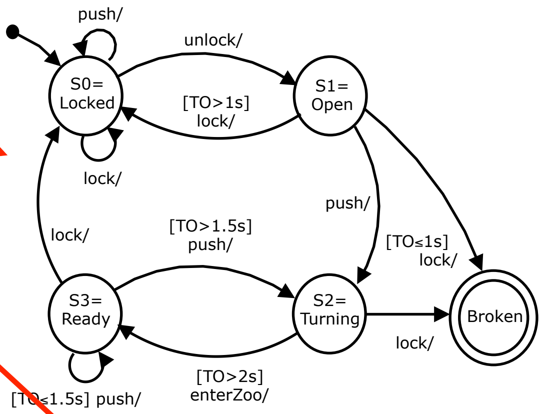
(a) ‘insert coin’ and ‘enter Zoo’ are poor abstractions
 * Complication and delay from Coin Acceptor & Barrier

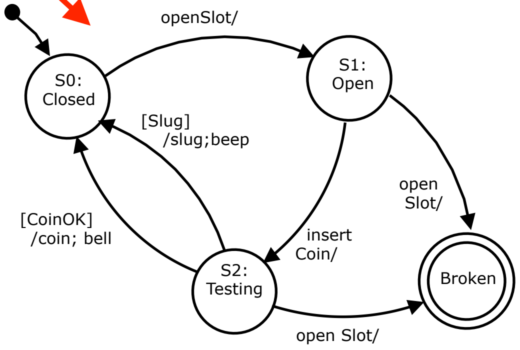
(b) How and when does the behaviour stop?
 * Termination is a standard ‘problem concern’

(c) Alternation of payment and entry vs R2 use-case
 * While pupils enter at Barrier, Teacher inserts many coins, building up a convenient excess of coins

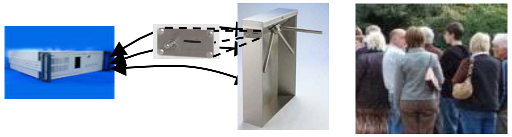
(d) Mutual exclusion of payment and entry vs R3, R4
 * Visitors are frustrated by waiting to insert a coin while an entry is in progress: Barrier and Acceptor must be able to operate concurrently

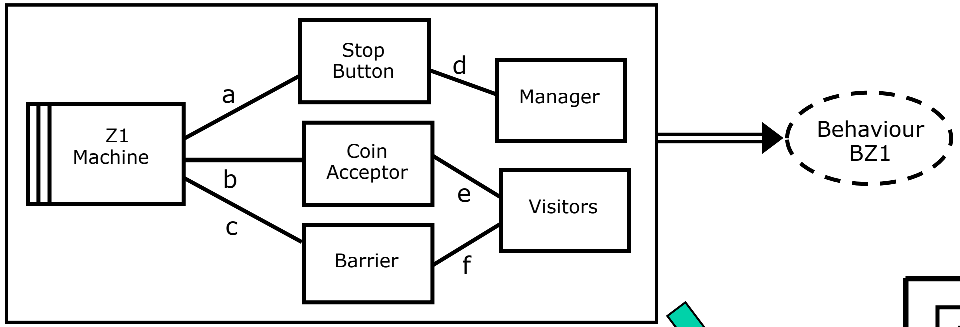
(e) System is now too complex for a simple problem
 * Concurrency demands explicit decomposition to two ‘subproblems’ and their combination





Decomposition into constituent behaviours

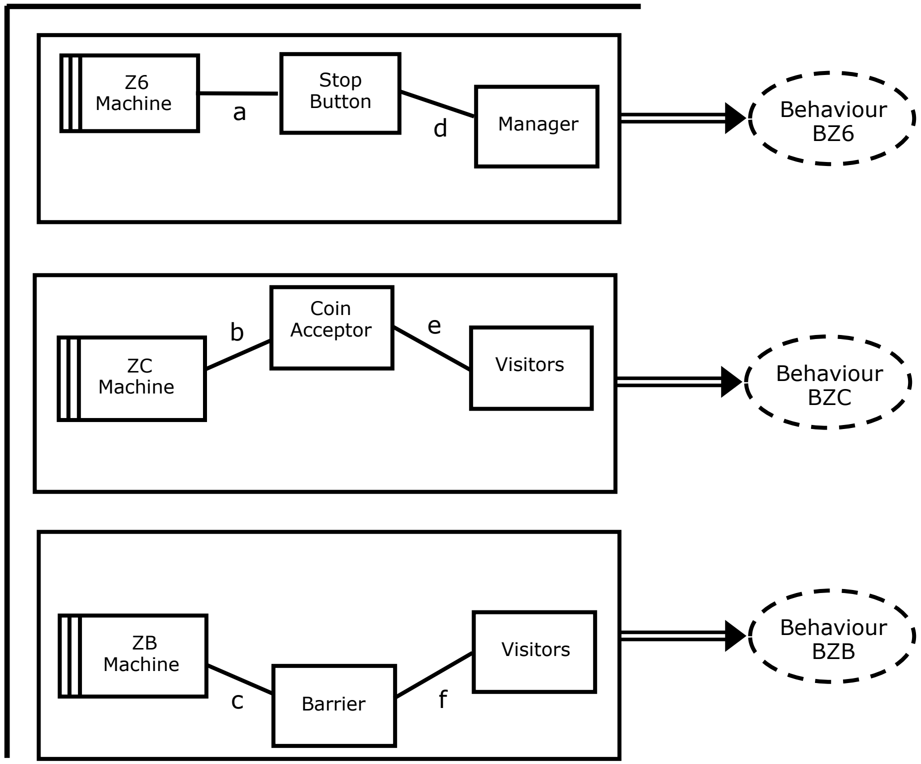




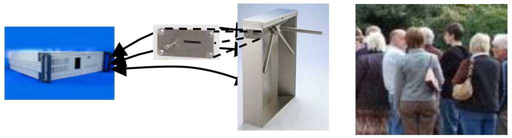
Behaviour BZ1 (Machine Z1) is decomposed into 3 behaviours:

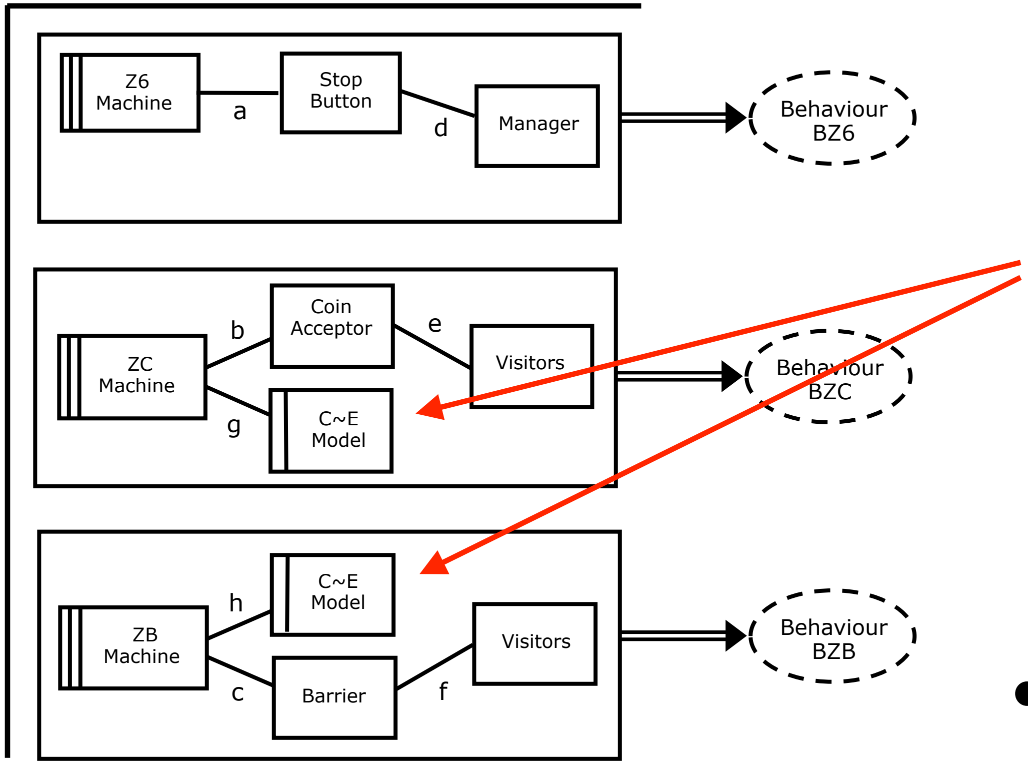
Behaviour BZ6 (Machine Z6);
Behaviour BZC (Machine ZC);
Behaviour BZB (Machine ZB)

What mechanism is needed to control and coordinate the three machines?



Behaviour control and a designed domain

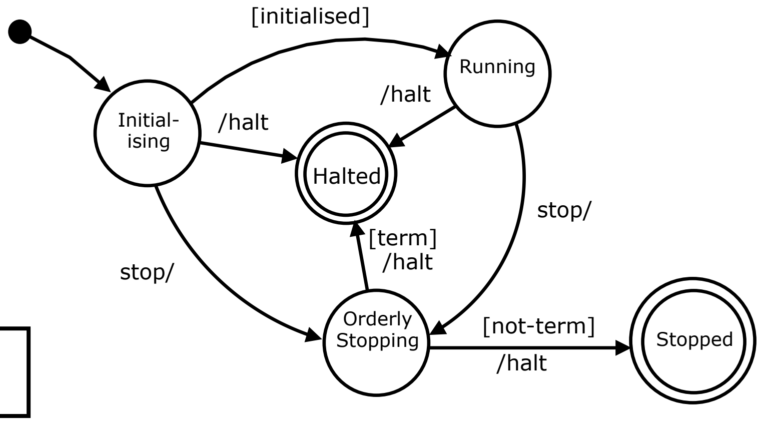




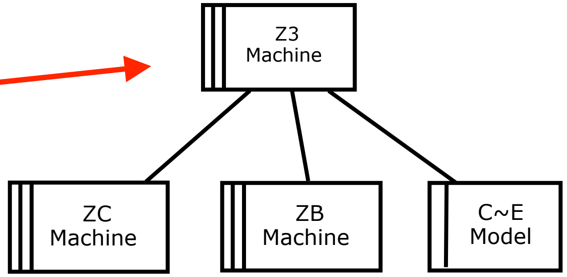
Machine Z6 controls ZC and ZB by a standard control protocol

The Z6 local variable C~E (the coin surplus) is ‘promoted’ to a problem domain for ZC and ZB

Behaviour worlds are closed and control is separate from content



Behaviour control in tree of machine instantiations



The Zoo Visitor control problem: some key points

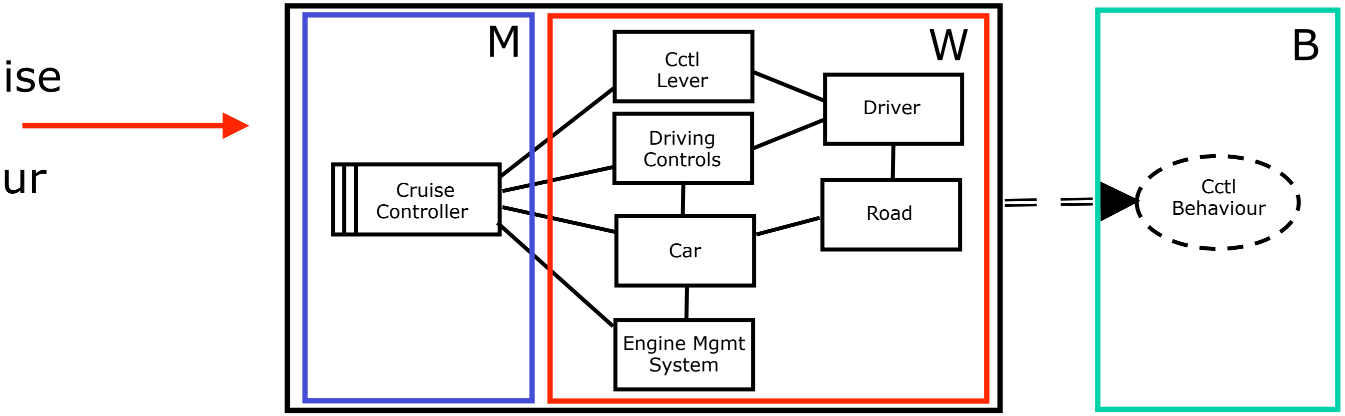
- Engineering BY Software
  - We are programming system M+W, not system M
  - W low-level properties have high-level effects
- It is a discipline of sequential programs (plus concurrency)
  - The discipline addresses standard concerns, eg: initialisation, termination, breakage, totality, surprise, …
- Here is the characterisation of a behaviour:
  - B is a program for M+W with a comprehensible purpose
  - An execution of M is an instantiation of B

- Components for behaviour design are themselves behaviours

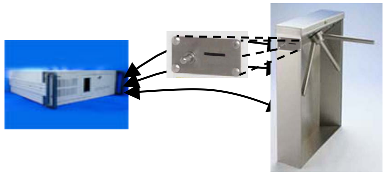
Complex behaviour is constituted of many simpler behaviours

- This car is moving under driver control
  - ABS is monitoring wheel speeds ready for braking
  - Air conditioning is cooling the car cabin
  - Stop-Start is not running
  - Automatic Parking is not running
  - Cruise Control is maintaining driver's chosen speed
  - Lane Departure Warning is watching lane markings
  - Speed Restriction is limiting speed to 110kph
  - Active Suspension is smoothing and stabilising ride
  - … …

The Cruise Control behaviour



What makes the Zoo problem small and simple?

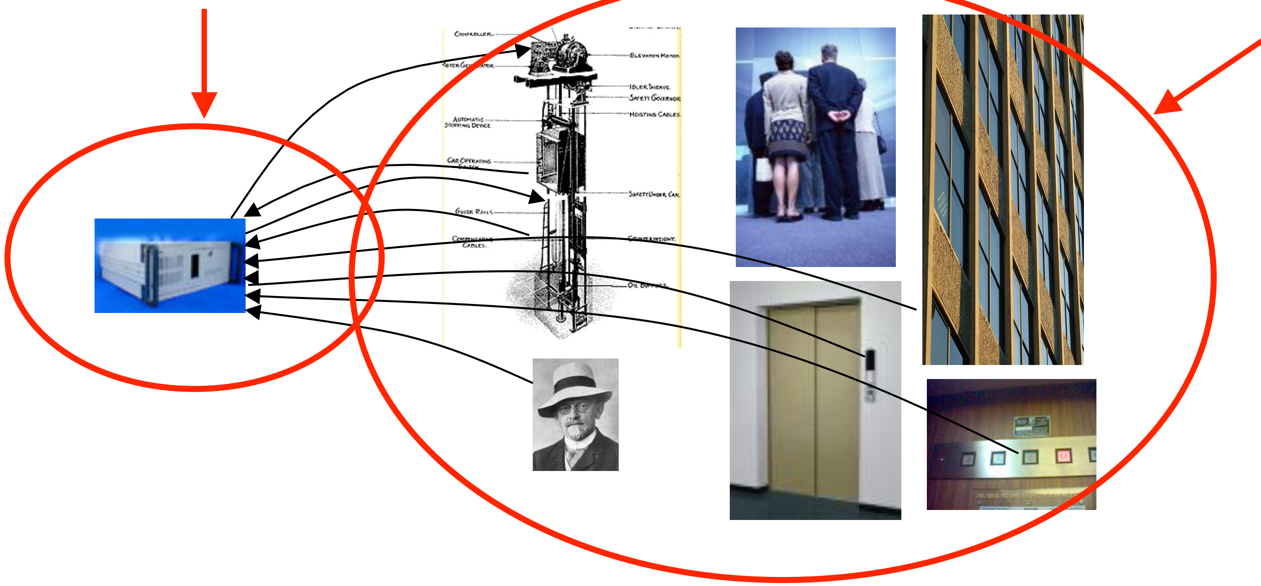


- Very limited functionality
  - Few stakeholders, few requirements
  - One function only: eg no reports, no maintenance, ...
- Constant simple domain properties, globally assumed
  - No sumo wrestlers
  - No ‘fault-tolerance’
  - No fires, earthquakes etc
- Extreme simplicity of machine/behaviour tree
  - Each behaviour is instantiated once, unconditionally (continuing after Stop is a new execution instance)
- Totally non-critical system
  - Narrow OE (operating envelope)
  - On failure: refund fees/start again/call engineer
- A larger problem will relax some of these limitations

A lift system for a building

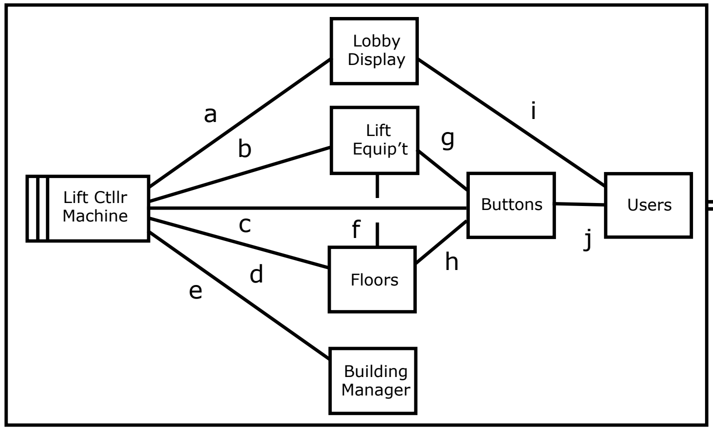
The Machine

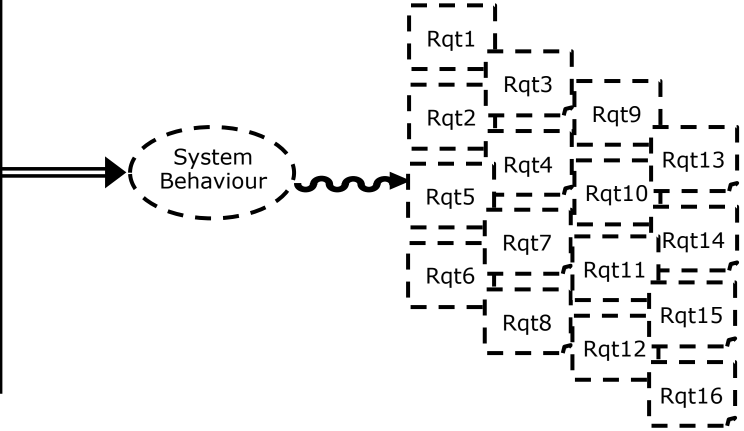
The Problem World



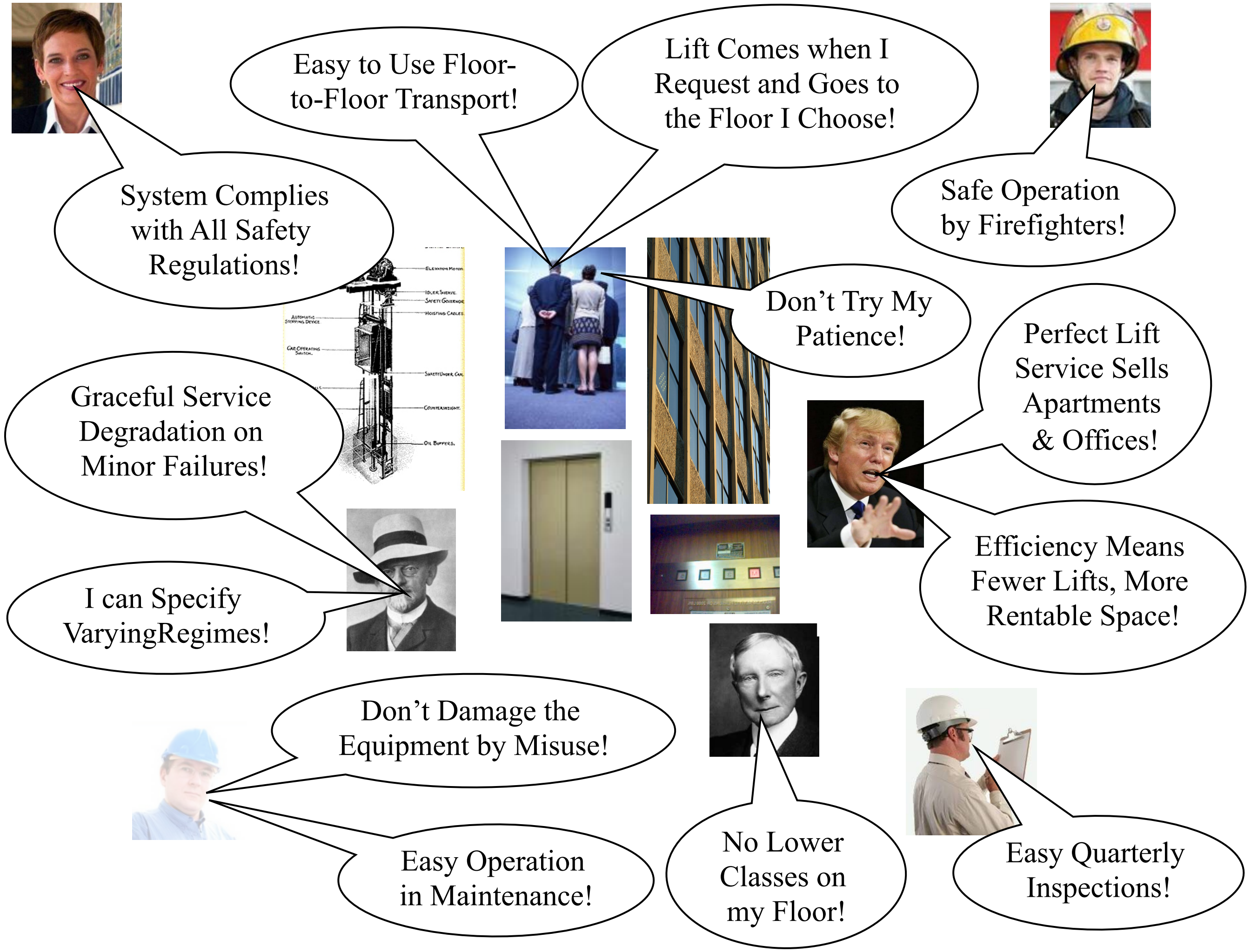
A large system having ..

- * Many stakeholders
- * Many requirements
- * Many behaviours
- * Complex behaviours
- * Some critical functions
- * Complex duty cycle
- * Wider OE





Some stakeholders and their requirements



System Complies with All Safety Regulations!

Easy to Use Floor-to-Floor Transport!

Lift Comes when I Request and Goes to the Floor I Choose!

Safe Operation by Firefighters!

Don't Try My Patience!

Perfect Lift Service Sells Apartments & Offices!

Graceful Service Degradation on Minor Failures!

Efficiency Means Fewer Lifts, More Rentable Space!

I can Specify VaryingRegimes!

Don't Damage the Equipment by Misuse!

Easy Operation in Maintenance!

No Lower Classes on my Floor!

Easy Quarterly Inspections!

Developing system behaviour starting from requirements

Where’s the AGB?

* There are many

Identify candidates

* From requirements

* From regions of operating envelope

Explore designs

* Simplified, isolated

Top-down design

* Children from parent

Bottom-up design

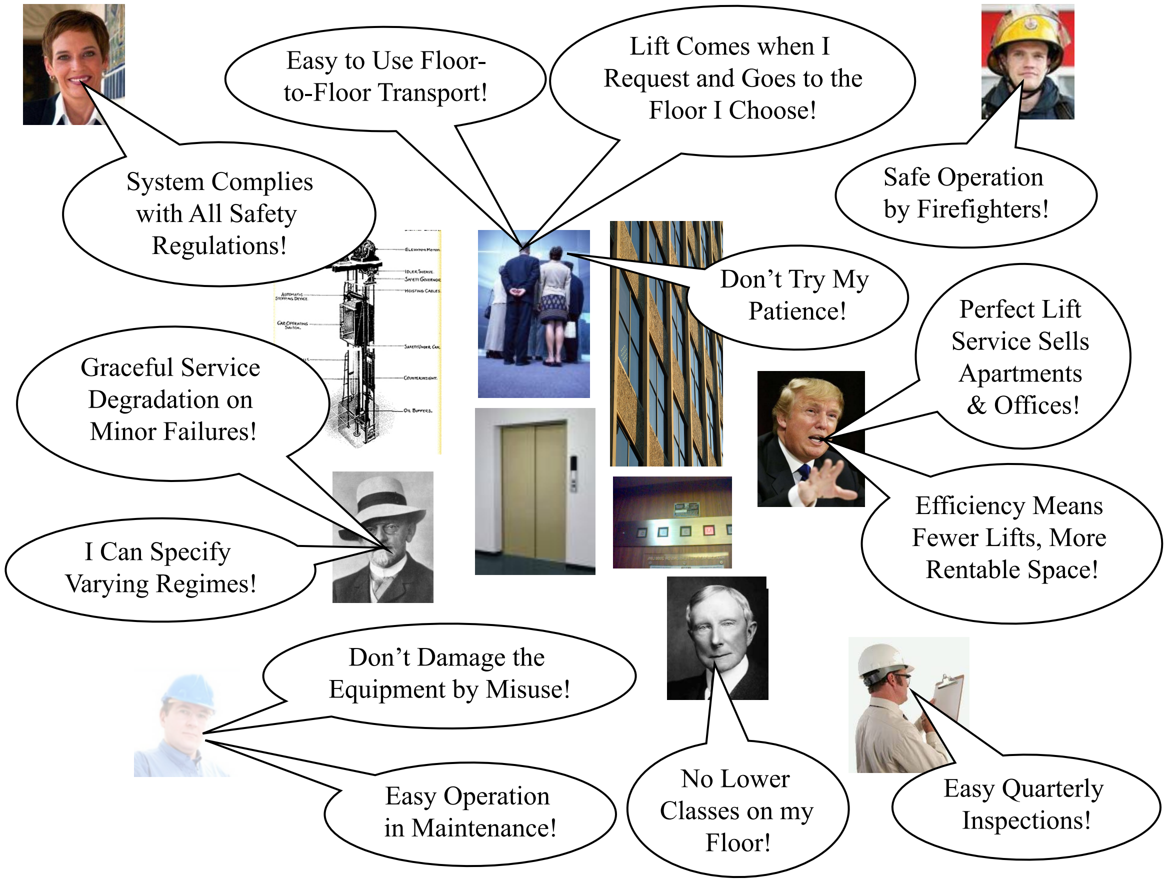
* Parent from children

Some new candidates

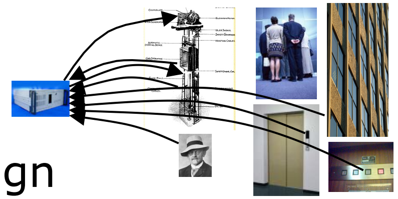
* From combination

Stepwise complication

* With possible abort


Easy to Use Floor-to-Floor Transport!
Lift Comes when I Request and Goes to the Floor I Choose!
System Complies with All Safety Regulations!
Safe Operation by Firefighters!
Don’t Try My Patience!
Perfect Lift Service Sells Apartments & Offices!
Graceful Service Degradation on Minor Failures!
Efficiency Means Fewer Lifts, More Rentable Space!
I Can Specify Varying Regimes!
Don’t Damage the Equipment by Misuse!
Easy Operation in Maintenance!
No Lower Classes on my Floor!
Easy Quarterly Inspections!

Some candidate constituent behaviours



1. From requirements directly

* NLS: Normal Lift Service

* FLS: Firefighter Lift Service

* ITM: Inspection Test Mode

* MEO: Maintenance Engineer Op’n

* EPR: Edit Lift Priority Regime

* TFR: Tycoon Floor Restriction

* …

2. From early requirements study

* FFP: Free Fall Prevention

* MLD: Maintain Lobby Display

* OTP: Overload Travel Prevention

* FRS1: Failure Reduced Svc Level 1

* FRS2: Failure Reduced Svc Level 2

* MPR: Manage Lift Priority Regimes

* …

3. From top-down design

* ODN: Open Doors Normal

* CDN: Close Doors Normal

* ODF: Open Doors Firefighter

* CDF: Close Doors Firefighter

* NQM: Maintain Normal R’qst Model

* LFM: Maintain Lift Fault Model

* CSM: Maintain Car State Model

* …

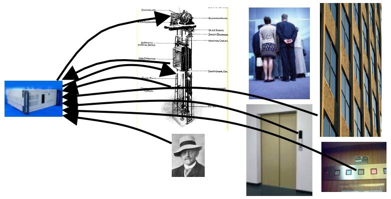
4. From combining constituents

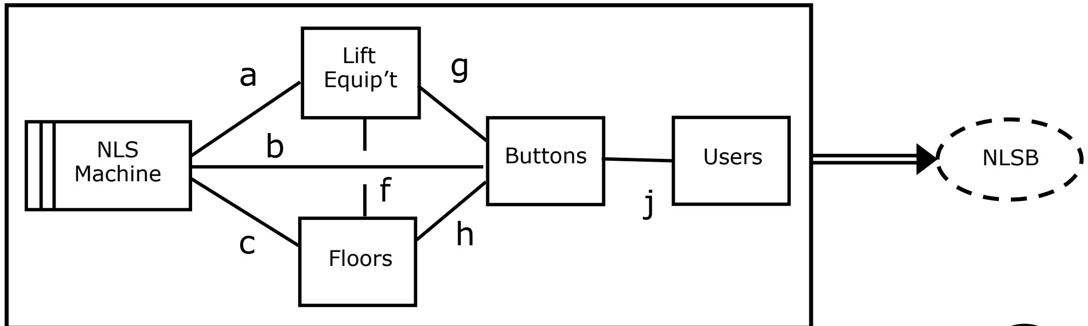
* PIL: Passenger Immediate Landing

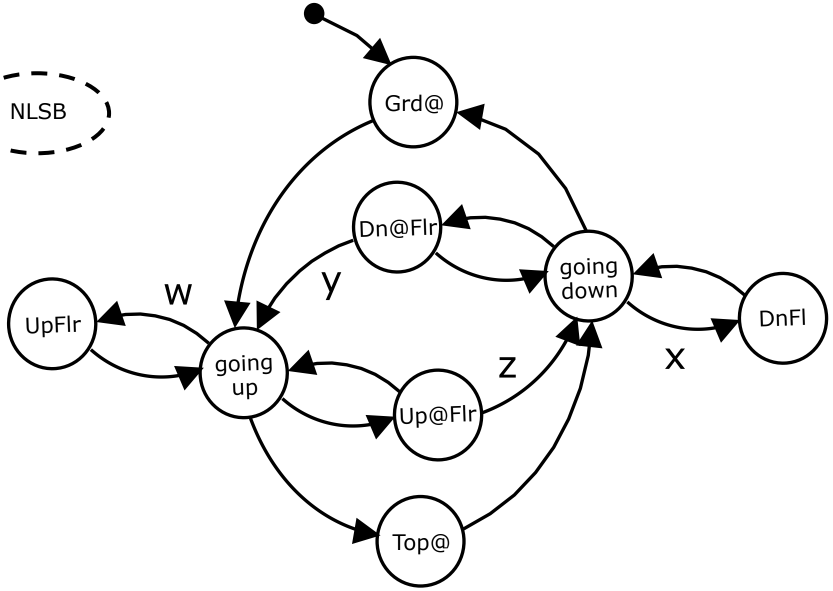
* PGL: Passenger Ground Floor Parking

* …

AGB for NLS (Normal Lift Service)



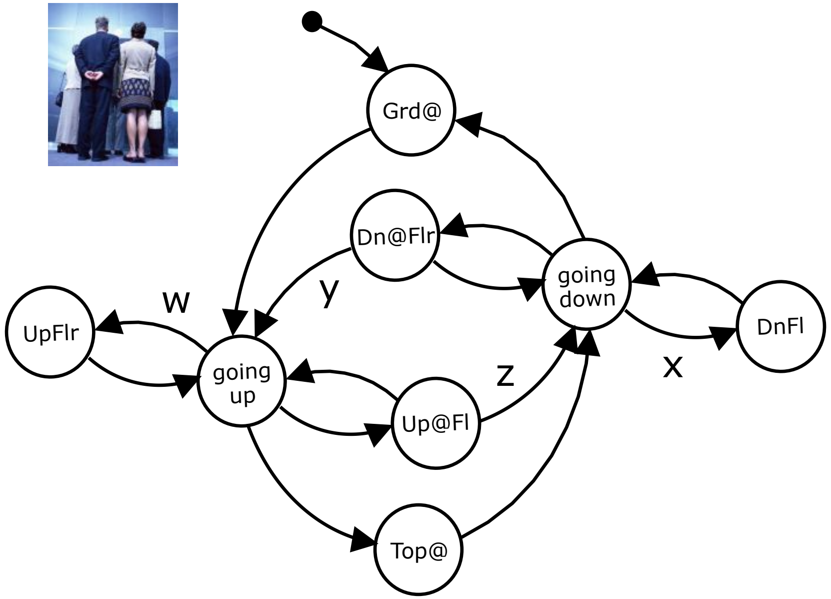


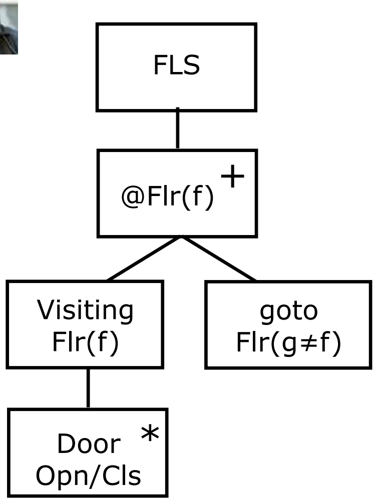


- Normal Lift Service Requirement: "Afford (passenger) use-cases"
- AGB: alternate up/down episodes
  * Choices (by NLS machine):
  * Skip UpFloor (w)
  * Skip DnFloor (x)
  * Reverse at Up@Flr (z)
  * Reverse at Dn@Flr (y)
- Simplifications
  * fixed standard policy (regime)
  * no equipment faults (yet)

- Top-down design suggests:
  * ODN, CDN (open, close doors)
  * NQM (maintain request model)

NLS and FLS behaviours: affording and obeying use cases





- Normal Lift Service Requirement is:

  “Afford (passenger) use-case”
- Shared physical resource
- Passenger identities are ignored

- Firefighter Lift Service Requirement is:

  “ Obey (operator) commands
- Dedicated physical resource
- Operator identity is ignored

Incremental complication for emerging behavioural properties

Incremental complication for NLS

- Initially: isolated, full simplicity
  - ODN, CDN: atomic, reliable
  - Regime variants ignored
- Deviations for minor failures
  - ODN, CDN delay or failure
  - Sensor failure at some floor[s]
  - Possible halt terminating NLS
- Concerns for a behaviour
  - Initialisation
  - Breakage
  - Termination
  - Surprise
  - …

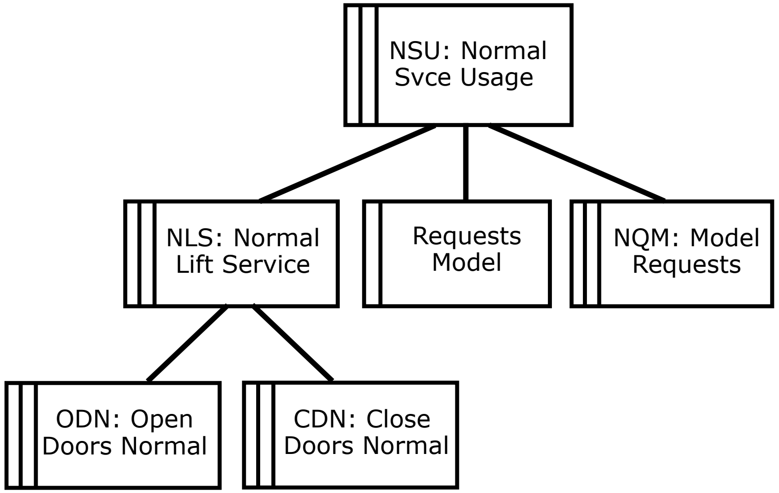
Design for combinations

- Behaviours’ time relationships
  - TFR, MLD with FLS?
  - ODN, ODF are different?
  - When can MPR be active?
  - EPR only within MPR?
  - Requests model persistence?

Concerns for combining

- Interference
- Direct conflict
- Sharing common resource
- Switching
- Interleaving (eg lift regimes)
- Terminating cooperation (eg Zoo)
- …

Sketching a subtree of behaviour instances



NSU: Normal Svce Usage

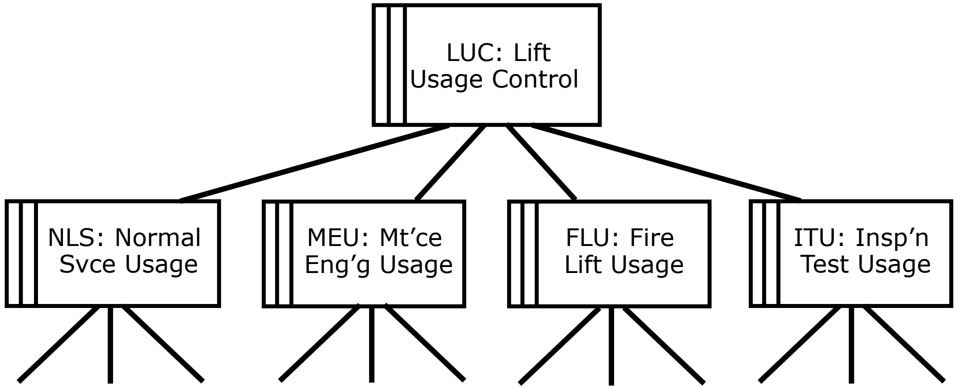
NLS: Normal Lift Service

Requests Model

NQM: Model Requests

ODN: Open Doors Normal

CDN: Close Doors Normal



LUC: Lift Usage Control

NLS: Normal Svce Usage

MEU: Mt’ce Eng’g Usage

FLU: Fire Lift Usage

ITU: Insp’n Test Usage

Subtree sketch for parent of NLS

* Persistence of Requests Model ?

* Why ODN/CDN and ODF/CDF?

* Should MLD be in this tree?

* How to fit in priority regimes?

Subtree sketch for major usages

* Is this how they fit together?

* What more do we need to know?

* What conditions to instantiate?

* How is each terminated?

* Where do MPR and EPR fit in?

What is this subtree’s parent?

You can identify these questions roughly, but the devil is in the details!

Benefits of a behaviour view

- A behaviour view applies at a smaller and a larger scale
  - At both scales it interfaces ‘Requirements’ and ‘Design’ tasks
    - At the smaller scale it informs design of a simple behaviour
    - At the larger scale it is a tool for structuring system function
- The smaller scale
  - Criteria of simplicity aid developer and stakeholder comprehension
  - Criteria of simplicity guide identification of constituent behaviours
  - ‘Subproblem concerns’ provide a checklist of failures to avoid
  - The relationship of software to problem world behaviour is formalised
- The larger scale
  - The machine/behaviour tree defines composite operational modes
  - The machine/behaviour tree maps to operating envelope regions
  - The machine/behaviour tree identifies interaction of constituents

Final thoughts

- Critical cyber-physical systems pose two major challenges
    - Huge functional complexity
        - Behaviour structure addresses functional complexity
    - Physicality of the problem world
        - Distinct behaviours may assume distinct given properties

- Behaviour structure exploits 50 years of program design knowledge
    - That’s not to be wasted!

Thank you!

Michael Jackson
The Open University
jacksonma@acm.org

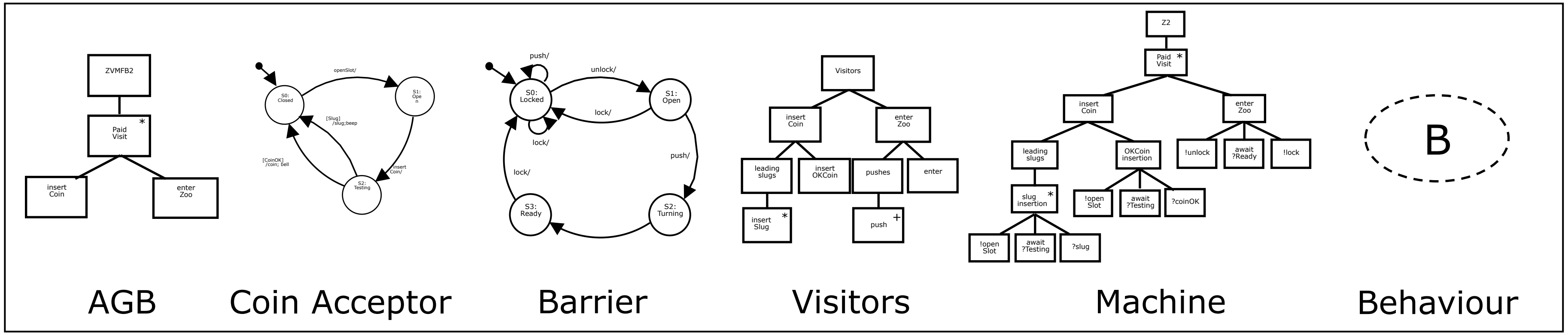
Additional Topics (12 slides)

Top-down works only for [almost completely] normal design

> “Richard Feynman ... noted the inevitability of more failures and embarrassing surprises if NASA did not change ... the way its big projects were designed. He called the procedure ... ‘top-down design’ and contrasted it with sensible ‘bottom-up’ design that has been normal engineering practice for centuries.”

Eugene S Ferguson; Engineering and the Mind’s Eye, pp188

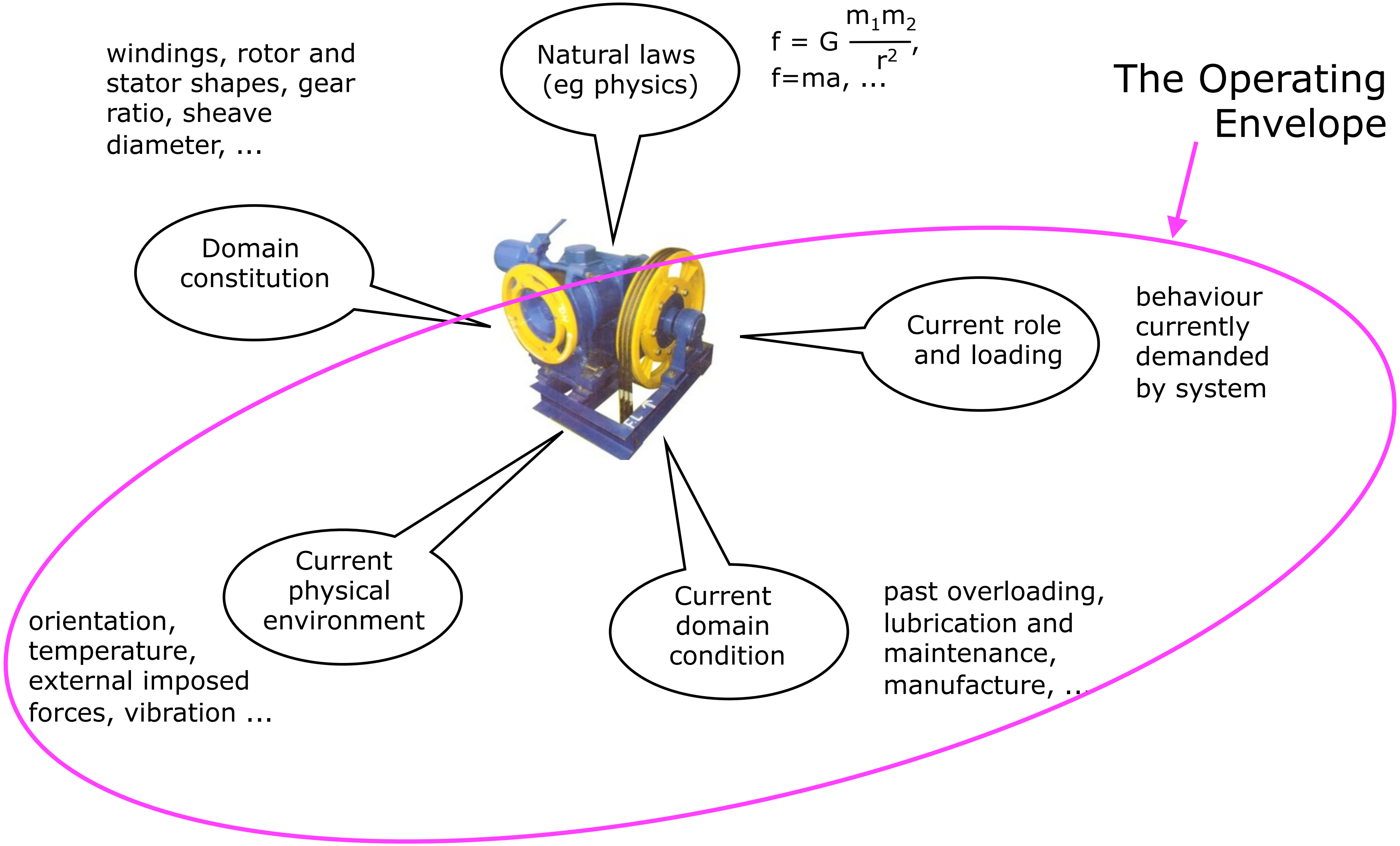
- What’s the difficulty?



AGB Coin Acceptor Barrier Visitors Machine Behaviour

- Properties of reality emerge at all levels, especially the most concrete
  - So we can’t assume that AGB can be refined to a feasible and desirable B
    - We intend AGB to be a property of the B that emerges from M and W
- Varying domain properties (fault-tolerance &c) add a very large complication

Properties of a physical domain and the operating envelope



windings, rotor and stator shapes, gear ratio, sheave diameter, ...

Natural laws (eg physics)

$f = G\frac{m_1 m_2}{r^2}$,
f=ma, ...

The Operating Envelope

Domain constitution

Current role and loading

behaviour currently demanded by system

Current physical environment

Current domain condition

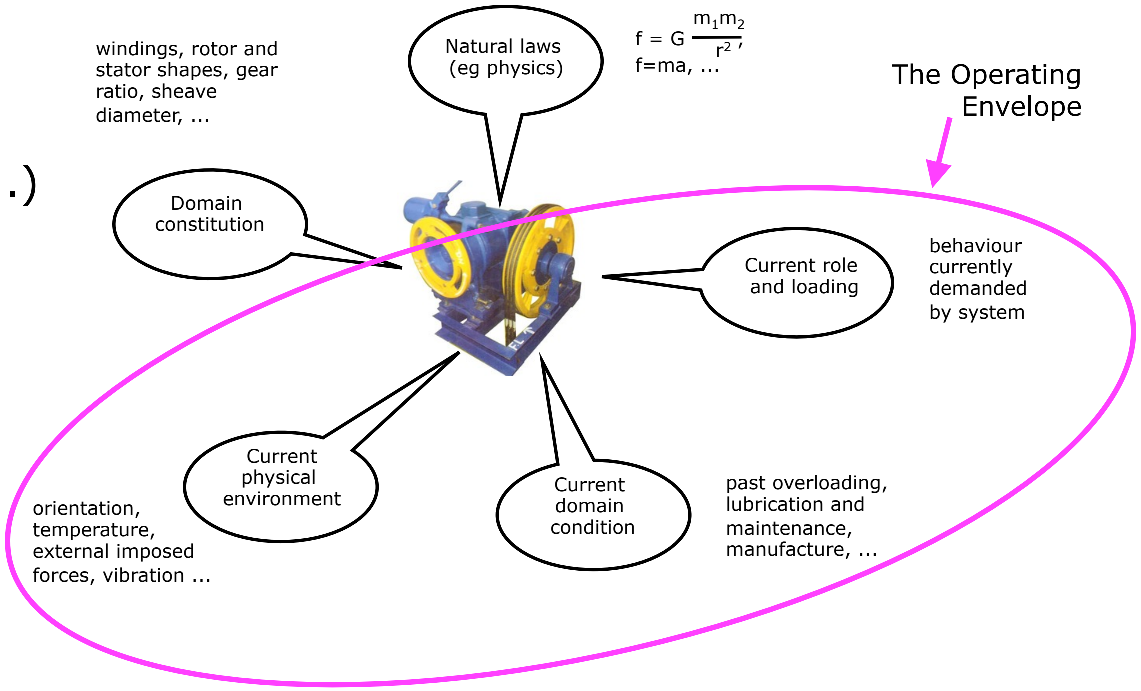
orientation, temperature, external imposed forces, vibration ...

past overloading, lubrication and maintenance, manufacture, ...

Restricting or regionalising the operating envelope

Restricting

- ‘Normal use’ or stipulate:
  - * environmental conditions (eg temp, power supply, …)
  - * maintenance schedule etc
  - * operating conditions
- Assume constant properties, ignore domain failures, disclaim all responsibility


windings, rotor and stator shapes, gear ratio, sheave diameter, …
Natural laws (eg physics)
$f = G \frac{m_1 m_2}{r^2}$,
f=ma, …
The Operating Envelope
Domain constitution
Current role and loading
behaviour currently demanded by system
Current physical environment
orientation, temperature, external imposed forces, vibration …
Current domain condition
past overloading, lubrication and maintenance, manufacture, …

Regionalising domain properties

* By system function

* By environmental conditions

* By current domain conditions

- Domain properties correspond to behaviours, varying by region

Note: regionalising is never complete

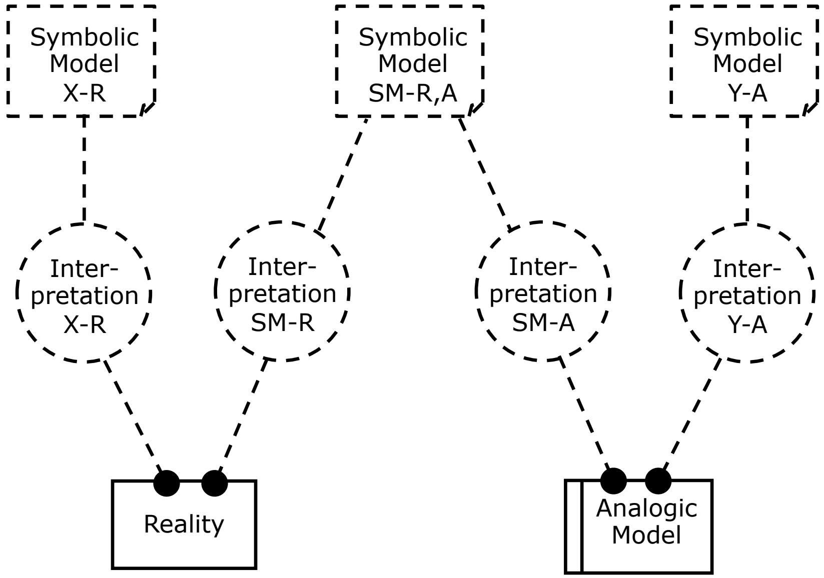
- The operating envelope is always restricted in some dimensions

Mastering complexity and physicality

- Cyber-physical systems present two large challenges
  - Functional complexity
  - Physical non-formality
- To address functional complexity we need structure
  - Components separate concerns and difficulties
  - Structure recombines simplicity into complexity
- Behaviour structure clarifies system state
  - Which behaviour instances are current?
  - What is the state of each current behaviour?
- Behaviour structure also addresses non-formality
  - Domain properties vary with varying conditions
  - Distinct behaviours rely on different properties

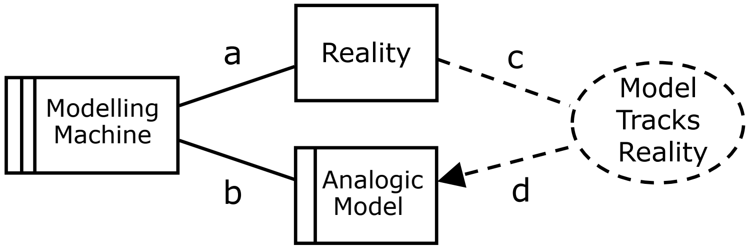
Interpretations for analogic software models

Symbolic models represent subjects linguistically (eg FSM, PDE, CSP, ...)
Analogic models represent subjects by analogy (eg electricity by water flow)

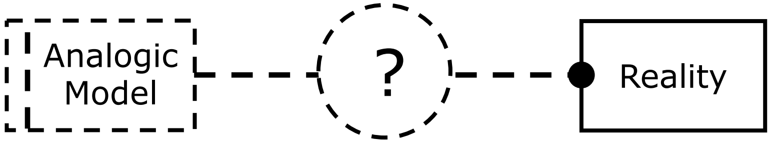


Symbolic and analogic models with their interpretations (designations)

Symbolic models X-R and Y-A are (or should be!) irrelevant to the analogy



In building an analogic model:
{a},{b},{c},{d} are distinct



Eliding the modelling interpretations is very tempting, very common, and very misleading

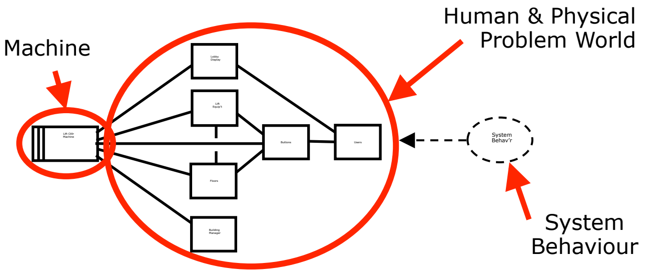
What counts as a behaviour? As a requirement?

- A stimulus-response pair like this does not count as a behaviour
- It's a program fragment (one transition arc)
  - * A single instruction is not a program!
  - * The purpose depends on unstated context

* Heading Select mode shall be selected when the HDG switch is pressed on the FCP

* Thank you, Mats Heimdahl

- A behaviour is associated with the program evoking it in the human and physical world
- 1 run of program ~ 1 instance of behaviour
- The run has an extended, unbroken duration
- A behaviour has a coherent intelligible purpose



- A requirement is a desired property or effect of a behaviour (or behaviours)
  - A requirement may be about the problem world
  - A requirement may be outside the problem world
  - A requirement may be formal or informal

Understanding a behaviour fragment (masquerading as a requirement)

Heading Select mode shall be selected when the HDG switch is pressed on the FCP

All right? A part of a sequential process—ie some behaviour!

Mats Heimdahl; *Let's Not Forget Validation*;
Position Paper for VSTTE Workshop, Zurich 2005.

Understanding a behaviour fragment (masquerading as a requirement)

Heading Select mode shall be selected when the HDG switch is pressed on the FCP

All right? A part of a sequential process—ie some behaviour!

If Heading Select mode is not selected, Heading Select mode shall be selected when the HDG switch is pressed on the FCP

Oh! Here's a condition. What if Heading Select mode is already selected? Beep? Just ignored?

Mats Heimdahl; *Let's Not Forget Validation*;
Position Paper for VSTTE Workshop, Zurich 2005.

Understanding a behaviour fragment (masquerading as a requirement)

Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

All right? A part of a sequential process—ie some behaviour!

If Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! Here's a condition. What if Heading Select mode is already selected? Beep? Just ignored?

If this side is active and
   Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! It seems this behaviour is parameterised: by {Left, Right}? What about danger of crosstalk?

Mats Heimdahl; *Let's Not Forget Validation*;
Position Paper for VSTTE Workshop, Zurich 2005.

Understanding a behaviour fragment (masquerading as a requirement)

Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

All right? A part of a sequential
process—ie some behaviour!

If Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! Here's a condition. What if
Heading Select mode is already
selected? Beep? Just ignored?

If this side is active and
Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! It seems this behaviour is
parameterised: by {Left, Right}?
What about danger of crosstalk?

If this side is active and
Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP
(providing no higher-priority event
occurs at the same time)

Ah! Resolution of a conflict with other
concurrent behaviours? I wonder
which take priority? And what
happens after the higher-priority
event? (Is selection just delayed or
permanently ignored?)

Mats Heimdahl; *Let's Not Forget Validation*;
Position Paper for VSTTE Workshop, Zurich 2005.

Understanding a behaviour fragment (masquerading as a requirement)

Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

All right? A part of a sequential process—ie some behaviour!

If Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! Here's a condition. What if Heading Select mode is already selected? Beep? Just ignored?

If this side is active and
Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP

Oh! It seems this behaviour is parameterised: by {Left, Right}?

If this side is active and
Heading Select mode is not selected,
Heading Select mode shall be selected when
the HDG switch is pressed on the FCP
(providing no higher-priority event
occurs at the same time)

Ah! Resolution of a conflict with other concurrent behaviours? I wonder which they could be? And what happens after the higher-priority event? (Is selection just delayed or permanently ignored?)

Mats Heimdahl; *Let's Not Forget Validation*;
Position Paper for VSTTE Workshop, Zurich 2005.

Do we understand it now, or is there more to be discovered?

Behavioural simplicity

The constituent behaviours and subproblems must be simple

What are the criteria of simplicity?

- Simple abstract goal behaviour (AGB)
- Simple abstract functional purpose
- One simple operational principle*
- One regular dynamic process structure
- Consistent properties of problem domains
- One region of the operating envelope
- ... ...

* Michael Polanyi; Personal Knowledge, pp328-329, University of Chicago Press, 1974

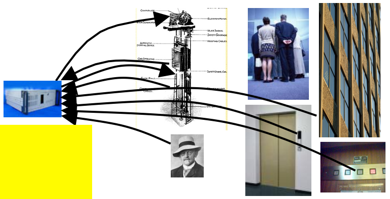
Problem Concerns

- Abortion: The world or machine aborts an interaction
- Abstraction failure: an abstractly defined phenomenon is not realisable in reality
- Approximation: The machine-world approximation is not faithful enough
- Completion: problem world frustrates completion of a composite operation (eg zoo entry)
- Creep: The machine-world approximation deteriorates with use
- Breakage: The machine breaks a problem domain
- Untimely response: A response occurs harmfully late (eg ‘resume’ in Cruise Control)
- Identities: Interacting with the wrong member of a set
- Information deficit: Machine can’t get information needed (demands model!)
- Information overload: Complexity of machine local variables justifies model domain
- Initialisation: Incompatible initial states of machine and world
- Overrun: Problem world goes too fast for the machine
- Races: Race conditions in the problem world
- Reliability: Problem domain properties are not satisfied
- Resource: Contention for scarce resources
- Surprise: Anomalous infraction of domain properties (eg change in DoB)
- Totality: Some problem world possibilities are ignored

Can I use these ideas selectively?

- Use system behaviour as a discriminant between requirements and design
- Use system behaviour as a key abstraction in early system development
- Study and analyse behaviour tree patterns
- Understand the persistence problem of a designed domain (eg a database)
- Repair requirement inadequacies by identifying constituent behaviours
- Avoid contaminating formal reasoning M,W|=B by informal concerns
- Use incremental complication
- Use loose decomposition
- Consider simplicity criteria
- Don’t fear multiple models of the same problem domain
- Don’t fear multiple occurrences of the same behaviour
- Can I stop at AGB for a subproblem?
- ‘problem concerns’ / ‘combination concerns’ are for one / many behaviours
- Behaviour tree is a map of danger points on ‘freed’ designed domains

PLC: Behaviours or requirements? OTP and FFP



OTP (Overload Travel Prevention) and FFP (Free Fall Prevention):
Are they behaviours or just requirements on several behaviours?

Overload Travel Prevention (OTP)

A buzzer sounds to alert passengers that the car is overloaded, the doors remain open and the car does not leave that floor until enough passengers exit the car.

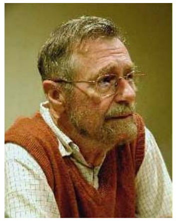
We may reasonably regard this is as a requirement to be satisfied in NLS (Normal Lift Service) when the doors are being closed for departure: it is a requirement on the door closing behaviour

Free Fall Prevention (FFP)

Lift speed is monitored, so that if the hoist cable breaks the emergency brake is applied, locking the car in the shaft and preventing any further movement.

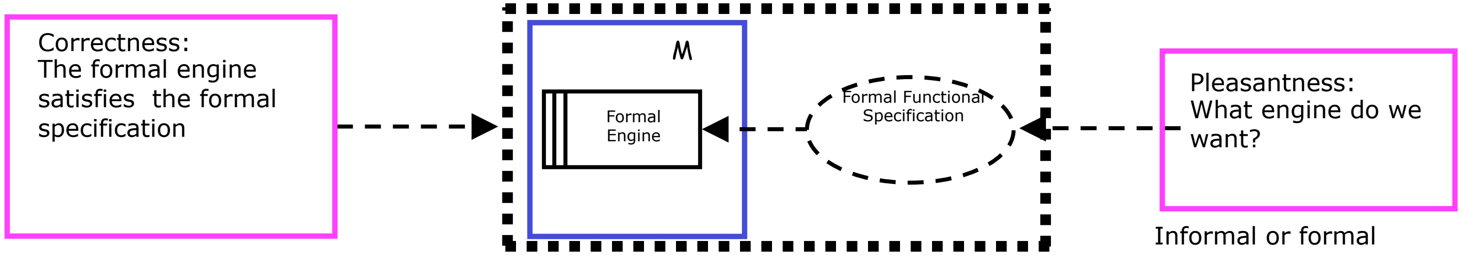
This seems certainly a distinct behaviour
It cannot be a requirement to be satisfied in other behaviours : Which ones?

Behaviours as a 'firewall' between requirements and software



> "The choice of functional specifications … may be far from obvious, but their role is clear: … to act as a logical 'firewall' between the 'pleasantness problem', … and the 'correctness problem' …"
>
> E W Dijkstra

For a program



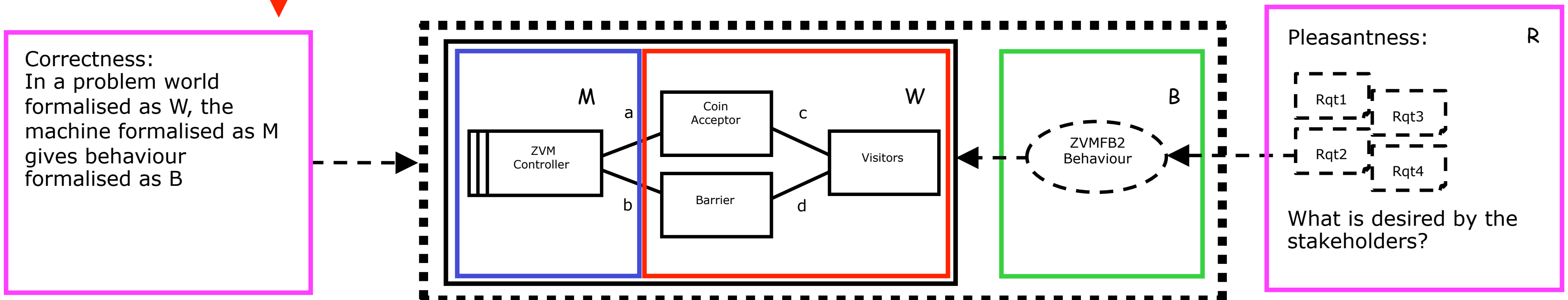
Correctness: The formal engine satisfies the formal specification

Pleasantness: What engine do we want?

Informal or formal

Formal, subject to physical computer

For a C-P system



Correctness: In a problem world formalised as W, the machine formalised as M gives behaviour formalised as B

Pleasantness: What is desired by the stakeholders?

Informal or formal

Formal, subject to physical computer and problem world

Lift system relationships among constituent behaviours

- What temporal relationships of behaviours are possible and desired?
- These questions are partly technical, and partly about requirements

FFP (Free Fall Protection): always active?

EPR (Edit Lift Pr'ty Regime): only nested within MPR (Manage Pr'ty Regimes)?

MPR (Manage Pr'ty Regimes): may be active at any time?

MLD (Maintain Lobby Display) and MLMM (Maintain Lift Mvt Model): are they coterminous?

MLD (Maintain Lobby Display): is active with FLS (Firefighter Lift Service)?

OTP (O’load Travel Prevention): applies to CDF (Close Door Firefighter)?

MLFM (Maintain Lift Fault Model): is active with FLS (Firefighter Lift Service)?

PIL (Passenger Immediate Landing): always activated on main power failure?

OTP (O’load Travel Prevention): applies to MEO (Maint’ce Engineer Oper’n)?

TFR (Tycoon Floor Restriction): active with FLS (Firefighter Lift Service)?

TFR (Tycoon Floor Restriction): active with MEO (Maint’ce Engineer Oper’n)?

Nine classes of software

- First, some terminology

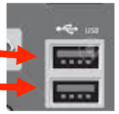
  Software = the program executed in the 'silicon package'

  Inputs = program inputs presented here

  Outputs = program outputs produced here

  Symbolic = to/from people via these

  Physical = via actuators/sensors like these



- The nine software classes

  0 GCD (Dijkstra)

  1 Read_Text module

  2 Read_Sensor module

  3 Write 1,000,000,000 primes

  4 Batch compiler, calculator

  5 BMEWS, train indicator

  6 Digitised C18 'automaton'

  7 Vending machine

  8 Automotive, Lift Control, ...

Outputs

Phys	6	7	8
Sym	3	4	5
–	0	1	2
	–	Sym	Phys

Inputs

Requirements validation for software engineering

Requirements on the process	Requirements on the developed product Satisfied? (Stakeholders, Developers)	
• Delivery costs and schedules • Development team make-up • Development method choice • Outsourced work control • Working hours • … …	Formal requirements on behaviour of the problem domains (Developers)	Formal requirements on behaviour outside the problem world (Stakeholders)
	Informal requirements on behaviour of the problem domains (Developers and Stakeholders)	Informal requirements on behaviour outside the problem world (Stakeholders)

X

Incremental complication for emerging behavioural properties

Incremental complication for NLS

* Initially: isolated, full simplicity
    * ODN, CDN: atomic, reliable
    * Regime variants ignored
* Deviations for minor failures
    * ODN, CDN delay or failure
    * Sensor failure at some floor[s]
    * Possible halt terminating NLS
* Concerns for a behaviour
    * Initialisation
    * Breakage
    * Termination
    * Surprise
    * …

Design for combinations

* Behaviours' time relationships
    * TFR, MLD with FLS?
    * CDN, ODN are different?
    * When can MPR be active?
    * EPR only within MPR?
    * Requests model persistence?

Concerns for combining

* Interference
* Direct conflict
* Sharing common resource
* Switching
* Interleaving (eg lift regimes)
* Terminating cooperation (eg Zoo)
* …

Benefits of a behaviour view

- A behaviour view applies at a smaller and a larger scale
  - At both scales it interfaces ‘Requirements’ and ‘Design’ tasks
    - At the smaller scale it informs design of a simple behaviour
    - At the larger scale it is a tool for structuring system function
- The smaller scale
  - Criteria of simplicity aid developer and stakeholder comprehension
  - Criteria of simplicity guide identification of constituent behaviours
  - ‘Subproblem concerns’ provide a checklist of failures to avoid
  - The relationship of software to problem world behaviour is formalised
- The larger scale
  - The machine/behaviour tree defines composite operational modes
  - The machine/behaviour tree maps to operating envelope regions
  - The machine/behaviour tree identifies interaction of constituents

Normal and radical engineering

- Only normal engineering products allow fully exact requirements
  - Standard designs match customer purposes and needs
  - Exact requirements specify parameters of standard design
- For a completely radical product
  - Design is uncertain: customer lacks basis of experience
  - Customer requirements are mostly vague uncertain hopes
- More radical design is likely to result in more dissatisfaction

Lift system relationships among constituent behaviours

- What temporal relationships of behaviours are possible and desired?
- These questions are partly technical, and partly about requirements

FFP (Free Fall Protection): always active?

EPR (Edit Lift Pr'ty Regime): only nested within MPR (Manage Pr'ty Regimes)?

MPR (Manage Pr'ty Regimes): may be active at any time?

MLD (Maintain Lobby Display) and MLMM (Maintain Lift Mvt Model): are they coterminous?

MLD (Maintain Lobby Display): is active with FLS (Firefighter Lift Service)?

OTP (O'load Travel Prevention): applies to CDF (Close Door Firefighter)?

MLFM (Maintain Lift Fault Model): is active with FLS (Firefighter Lift Service)?

PIL (Passenger Immediate Landing): always activated on main power failure?

OTP (O'load Travel Prevention): applies to MEO (Maint'ce Engineer Oper'n)?

TFR (Tycoon Floor Restriction): active with FLS (Firefighter Lift Service)?

TFR (Tycoon Floor Restriction): active with MEO (Maint'ce Engineer Oper'n)?

Some observations on behaviour-oriented development

- Critical cyber-physical systems pose two major challenges
  - Huge functional complexity
    - Behaviour structure addresses functional complexity
  - Physicality of the problem world
    - Distinct behaviours may assume distinct given properties
- Behaviour development specifies (not implements) a machine
  - For B, M is a program
  - For software development
    - M is not a specification of a software part
    - M is a part of a software specification
- Behaviour structure exploits 50 years of program design knowledge
  - Bounds the concerns to address at any one time
  - Relates each component to a comprehensible purpose
  - Allows systematic local search for potential failures