Problem Frames

A Tutorial

Michael Jackson
jacksonma@acm.org

14th Asia-Pacific SE Conference
Nagoya, December 4 2007
14:00—17:00

Problem frames: a tutorial

5 minutes — 1. Introductions
- Where is our starting point?

65 minutes — 2. Concepts (lightly, in breadth)
- Why think about problems?
- Basic problem frame ideas

10 minutes — Short break

75 minutes — 3. Example (in more depth)
- Analysis of one problem
- Composing the subproblems

10 minutes — Short break

15 minutes — 4. Practicalities
- Applying problem frames
- Some general principles
- Final discussion

Short PF bibliography

5 minutes

1. Introductions
   - Where is our starting point?

Introductions

- How new are problem frames to you?
- What kinds of system are you interested in?
- Have you read either of these books?

Problem Frames: Analyzing and Structuring Software Development Problems

Software Requirements & Specifications: a lexicon of practice, principles and prejudices

65 minutes

2. Concepts (lightly, in breadth)

- Why think about problems?
- Basic problem frame ideas

Why think about problems?

- Aren't programs much more fun?
  - Good programming and program design are "building the system right" (B Boehm)
- We need to "build the right system" (B Boehm)
  - Understanding requirements
  - Achieving dependability in the real world
- Understanding requirements = understanding problems
  - The software is the solution, not the problem
  - Exploring the problem reveals requirements
- Dependability demands intellectual structure
  - Problem analysis can reveal complexities early …
    - … when they can be addressed economically
  - Most failures are obvious in hindsight
    - Problem structuring helps foresight

Concepts: what are problem frames about?

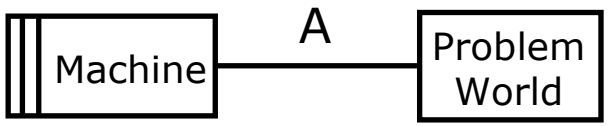
- Problem frames are
  - A framework for structuring problems in developing software-intensive systems
  - A framework for capturing experience
  - A framework for recognising concerns
  - A prompt for asking the right questions
- Problem frames are not
  - A development method or process
  - A formal calculus such as refinement
  - A new software development language
- Problem frames can be used ...
  - ... with many development processes
  - ... with many specific techniques
  - ... with many formal notations

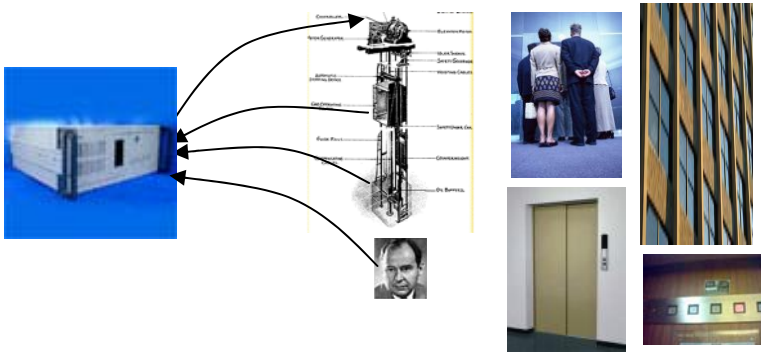
Concepts: basic problem frame ideas

1. The machine and the problem world
2. The problem world and the requirement
3. The machine as a problem solution
4. Subproblems as components
5. Introducing lexical domains
6. Identifying subproblem concerns
7. Standard subproblem classes
8. Composition concerns

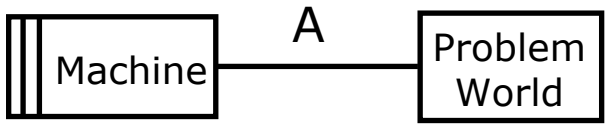
The machine and the problem world

machine problem world



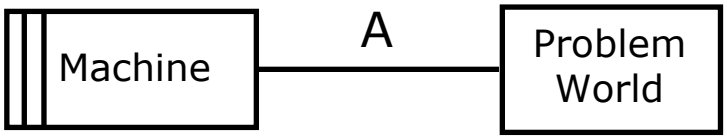


machine problem world



- The machine: what we must build (in software)
- The problem world: what the problem is about
  - The problem world is given
- A: the machine / problem world interface
  - A is an interface of shared phenomena
    - Phenomena: primarily events and states
    - eg: sensor[f], pressUpButton[f]

The machine and the problem world



- Car park control

- Administering a conference

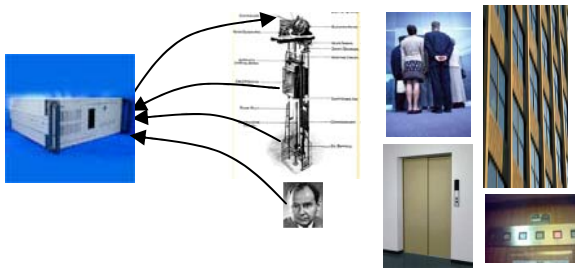
- Problem world contents depend on the requirement

The problem world and the requirement

machine problem world

- only members can borrow books
- reserved books available to collect
- reminders sent for overdue loans
- fees and fines are collected
- catalogue is up to date
- management reporting …
- … …

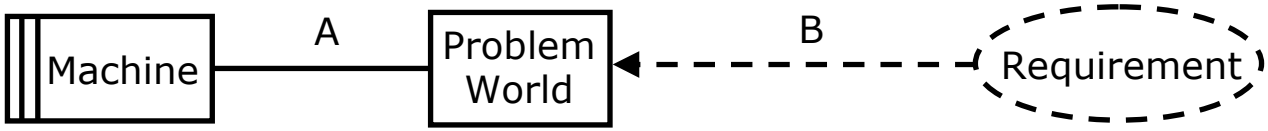
requirement



machine problem world

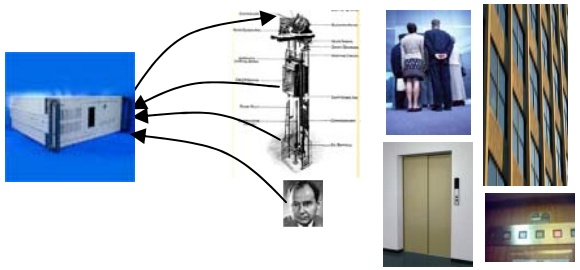
- lift comes on demand
- lift goes to requested floor
- safety against equipment failure
- gives efficient service
- current car location shown
- adjustable priorities …
- … …

requirement



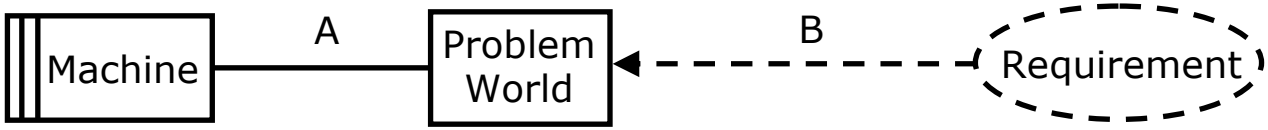
- The requirement is a condition on the problem world
  - Expressed in terms of phenomena B (B ≠ A)

Requirements and problem world properties



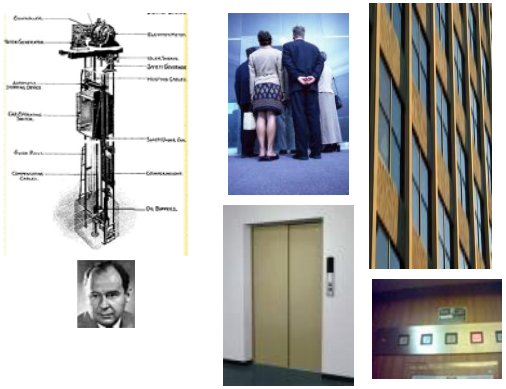
- lift comes on demand
- lift goes to requested floor
- safety against equipment failure
- gives efficient service
- current car location shown
- adjustable priorities …
- … …

machine problem world requirement



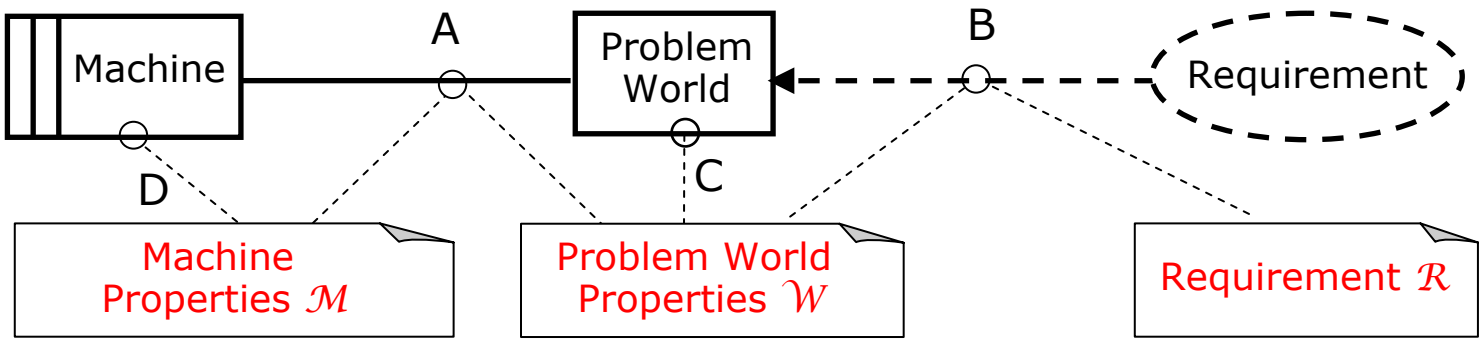
- The requirement is a condition on the problem world …
  - … which the installed machine must impose
  - eg: lift comes on demand
- The problem world also has given properties …
  - … independent of what the machine may do
  - eg: motor-on[direction=up] causes lift to rise

Requirements and problem world properties



- Requirement or problem world property?
  - Lift does not move if doors are open
  - Floor n can be reached from n-2 only via n-1
  - If up-arrow is lit lift will not depart downwards
  - Request button at floor is unlit when lift stops at floor
  - Floor doors and lift doors open and close together
  - If lift is within 15cm of floor, floor sensor is on
  - Pressing request button again has no effect
  - Only one floor is lit in lift position display in lobby

The machine as a problem solution

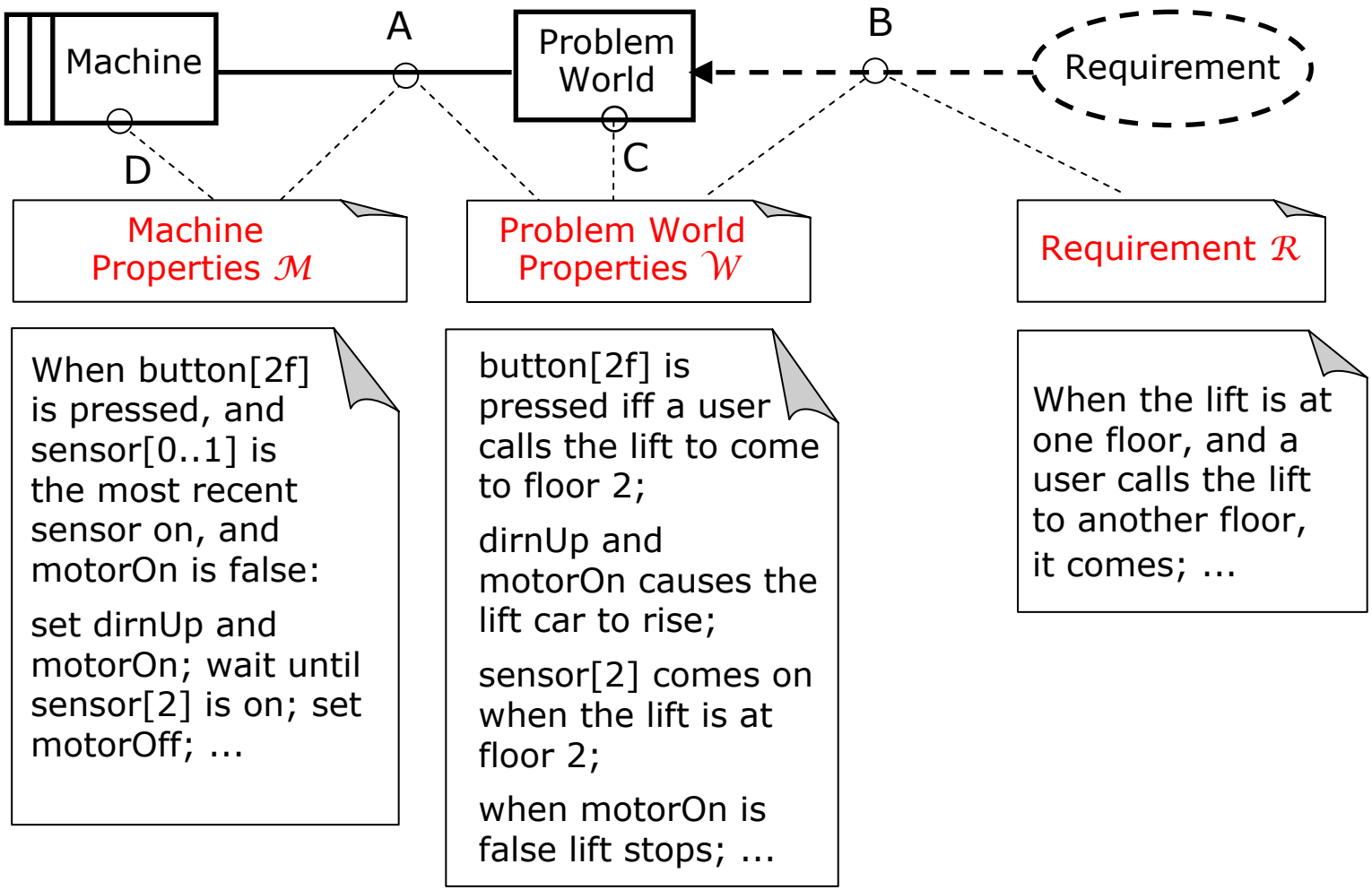


- Phenomena
  - A: phenomena shared between Machine and World
    - MotorOn, SensorOn[f], DoorMotorOn, ButtonPress[b], …
  - B: phenomena mentioned in the Requirement
    - CarArrives[f], DoorOpens, PassengerRequest[f], …
  - C, D: private (unshared) phenomena of Machine and World
- The Machine and the Problem World must cooperate …

$$\mathcal{M}, \mathcal{W} \models \mathcal{R}$$

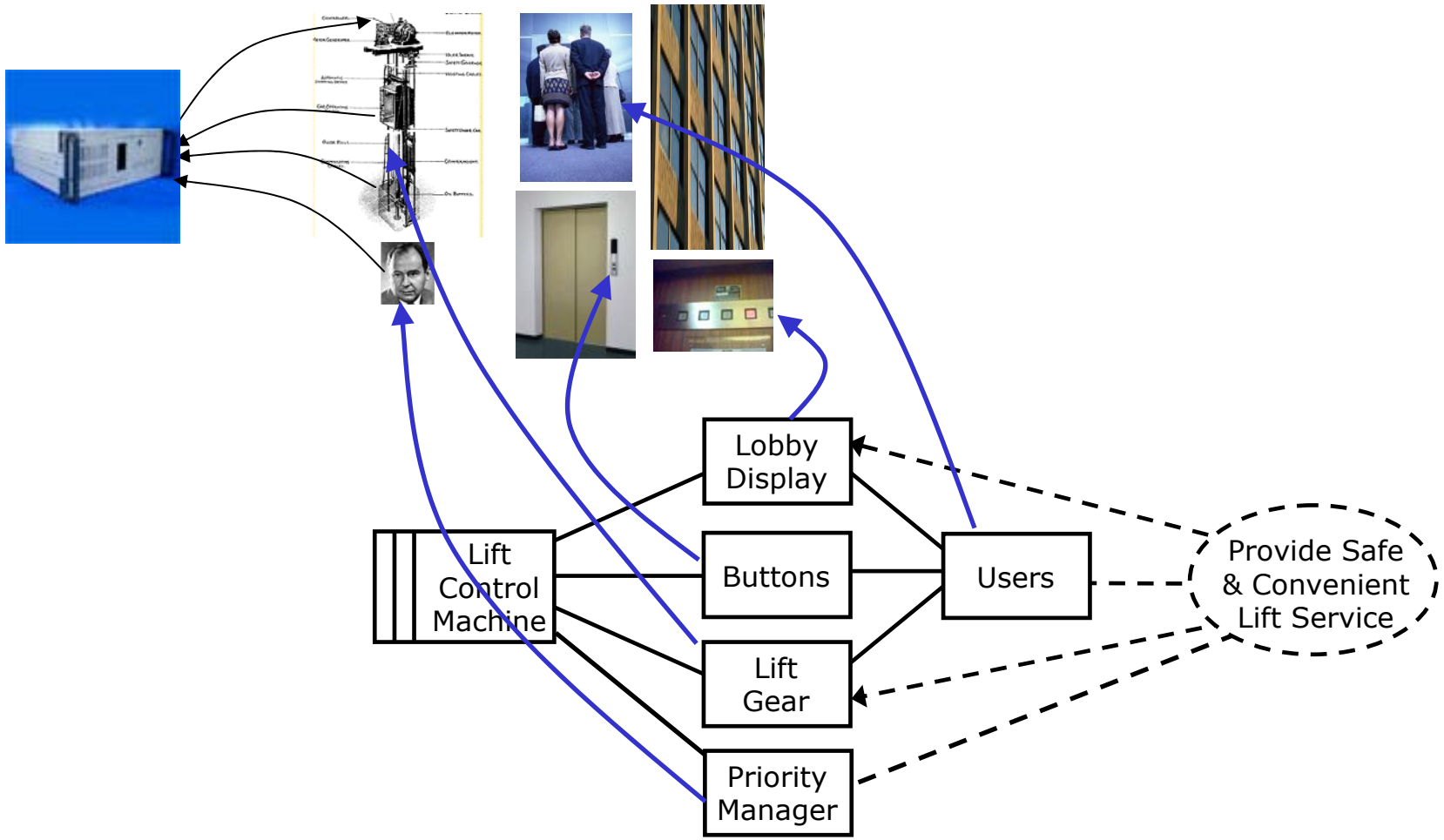
… to satisfy the requirement

$\mathcal{M},\mathcal{W},\mathcal{R}$: three fundamental descriptions



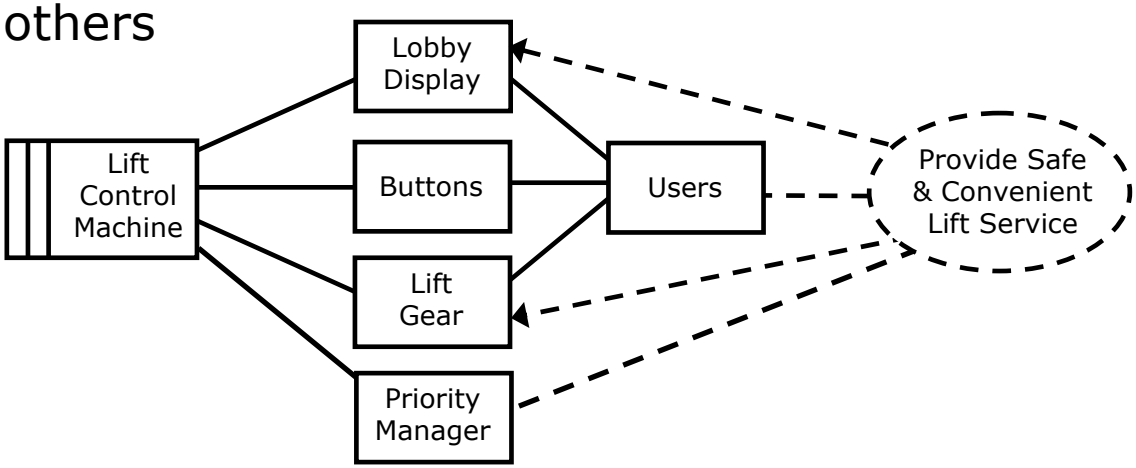
Decomposing the problem world

- We may decompose the problem world into domains

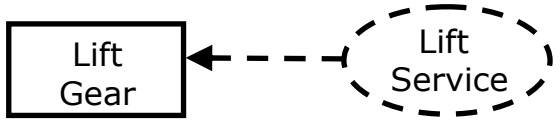


Decomposing the problem world

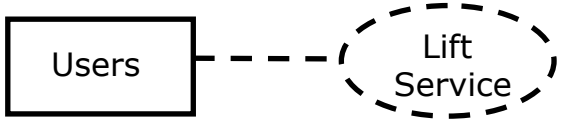
- In general, a requirement …
  - … constrains some problem world domains
  - … and not others



- Notation



- The requirement constrains ◄- - - - this domain



- The requirement only refers to - - - - this domain

The idea of a machine

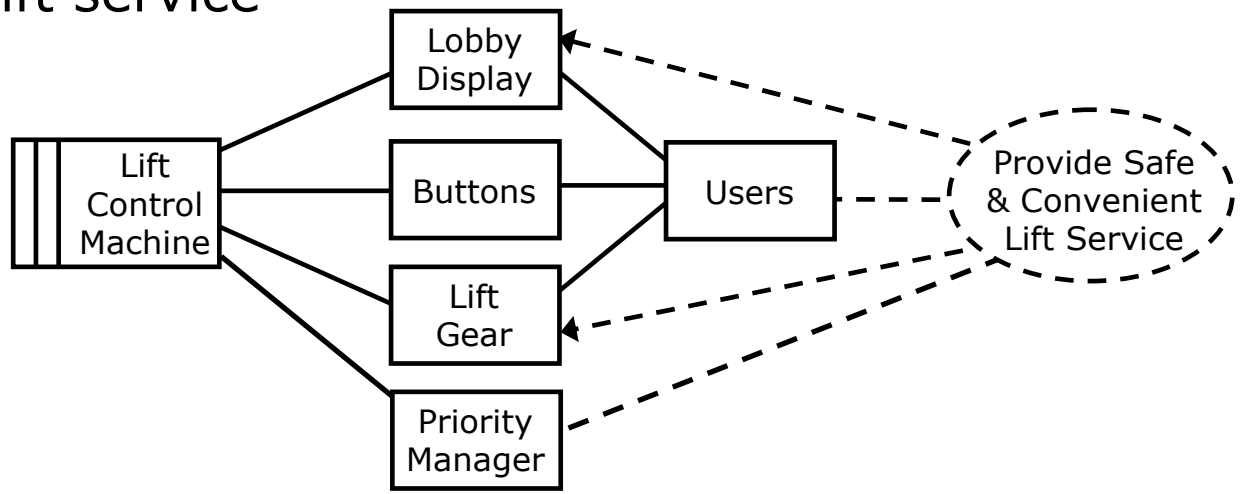
- Our idea of a machine …
  - … is relative to problems
  - In a particular problem:
    - the machine is what we must develop
    - the problem world is what is given
  - A machine is a partial view of software function
- Each problem has one machine
  - In the eventual system, a machine may represent …
    - … all or part of …
    - … one or more computers + software
- >1 machine means >1 problem
  - But the machine in one problem …
    … may be a problem domain in another problem
- We will decompose problems into subproblems
  - A subproblem is a smaller, simpler problem

Subproblems as components

- Granularity in problems, requirements, designs
  - System response to one stimulus
    "When button X pressed, turn on motor"
  - One use case
    "Each use of the lift follows this scenario …"
  - One object class
    "A [UML] object models one light unit"
  - One subproblem
    "Show lift position and requests on lobby display"
  - One system property or constraint
    "Two lifts never serve the same request"
- Problem frames emphasise subproblem granularity
  - Each subproblem has …
    - … a machine, problem world and requirement
  - Subproblems are our problem components
    - We defer the question: how to compose them?

Problem Decomposition

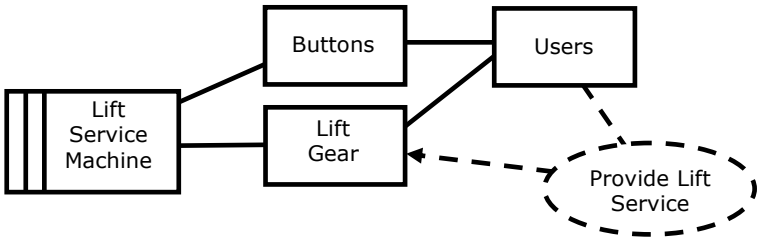
Problem:
Provide safe and convenient lift service



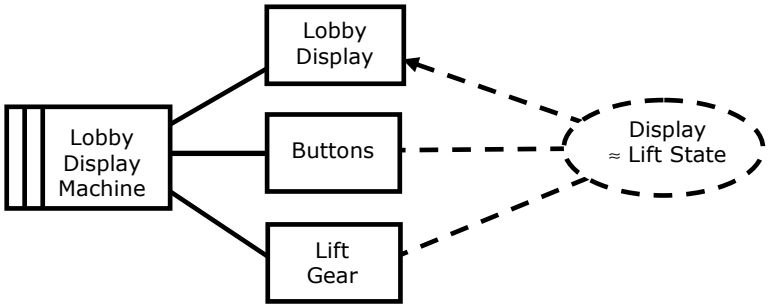
Subproblems as components

- A roughly sketched example

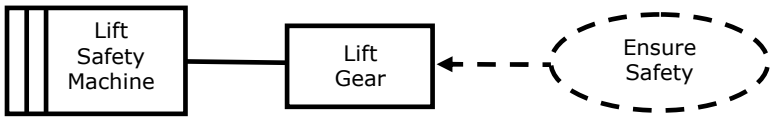
Provide lift service for user requests



Show requests and lift positions on lobby display



In event of malfunction apply emergency brake



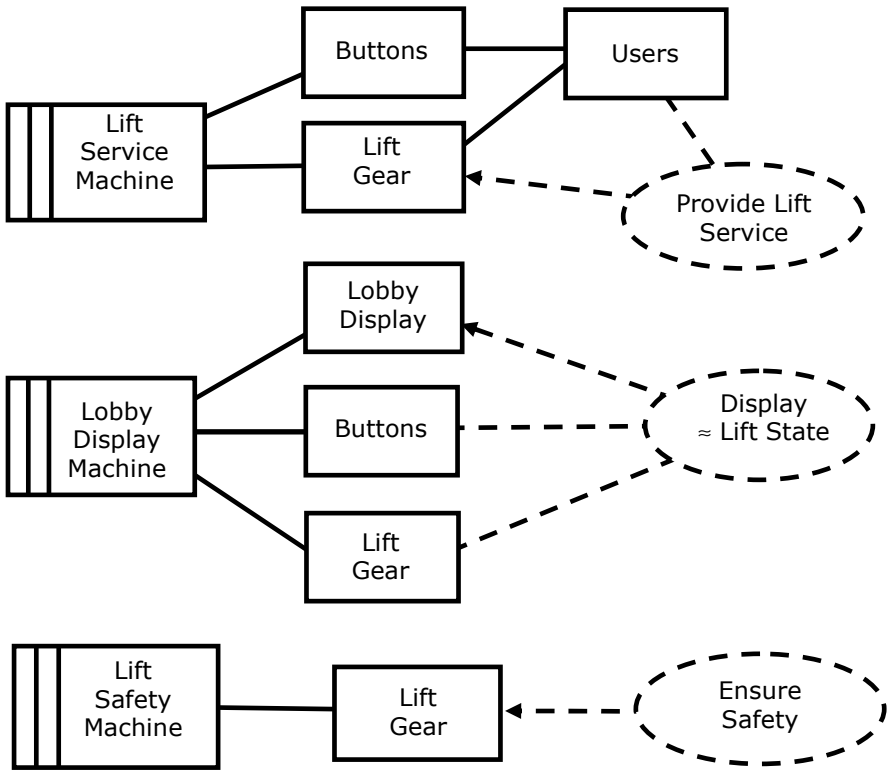
Subproblems as components

- Different subproblems may assume different properties of the same domain

Correctly functioning Lift Gear domain

Lift Gear domain with unspecified behaviour

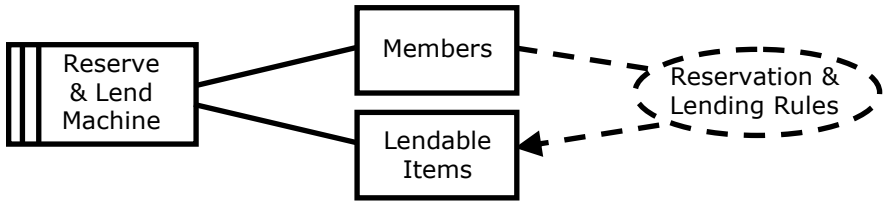
Possibly malfunctioning Lift Gear domain



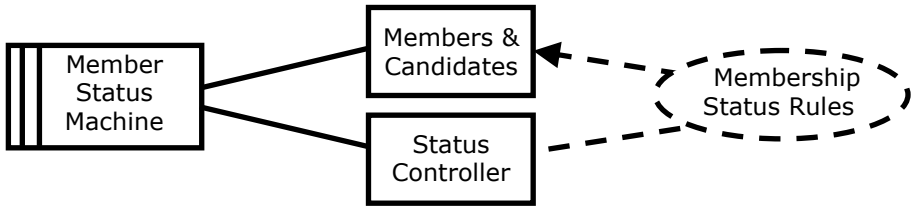
Subproblems as components

- Different subproblems may assume different properties of the same domain

For reservations and loans, library Membership is assumed to be static

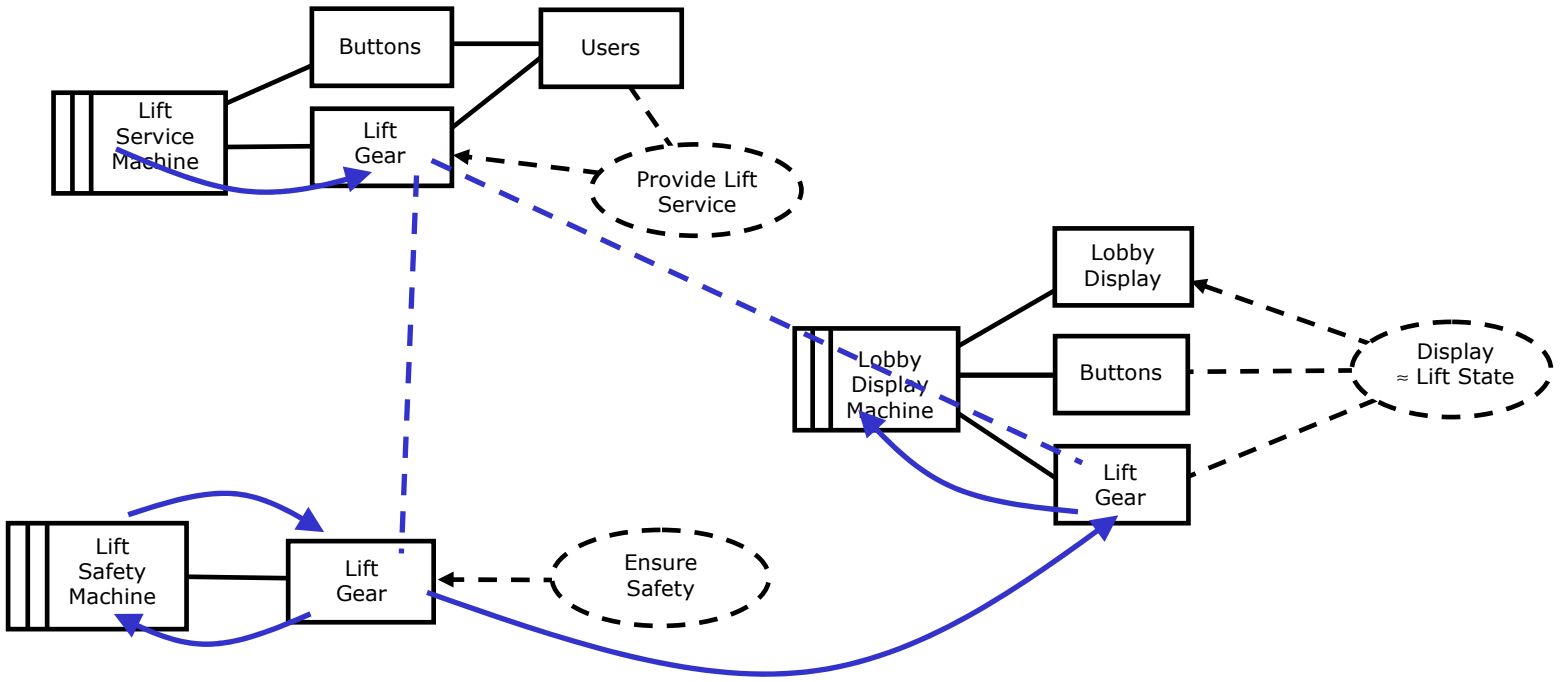


For managing member status, library Membership is assumed to be dynamic



Subproblems as components

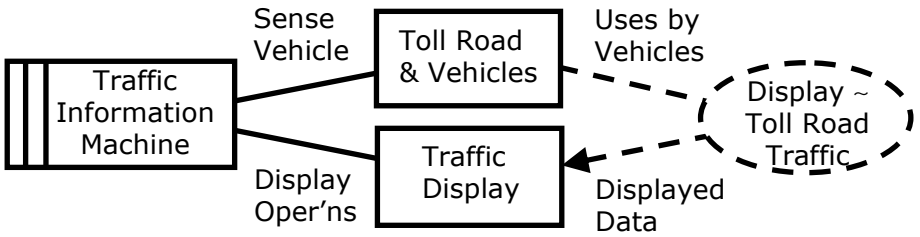
- Subproblem machines interact both directly (once composed) and indirectly through problem domains



Decomposition heuristics

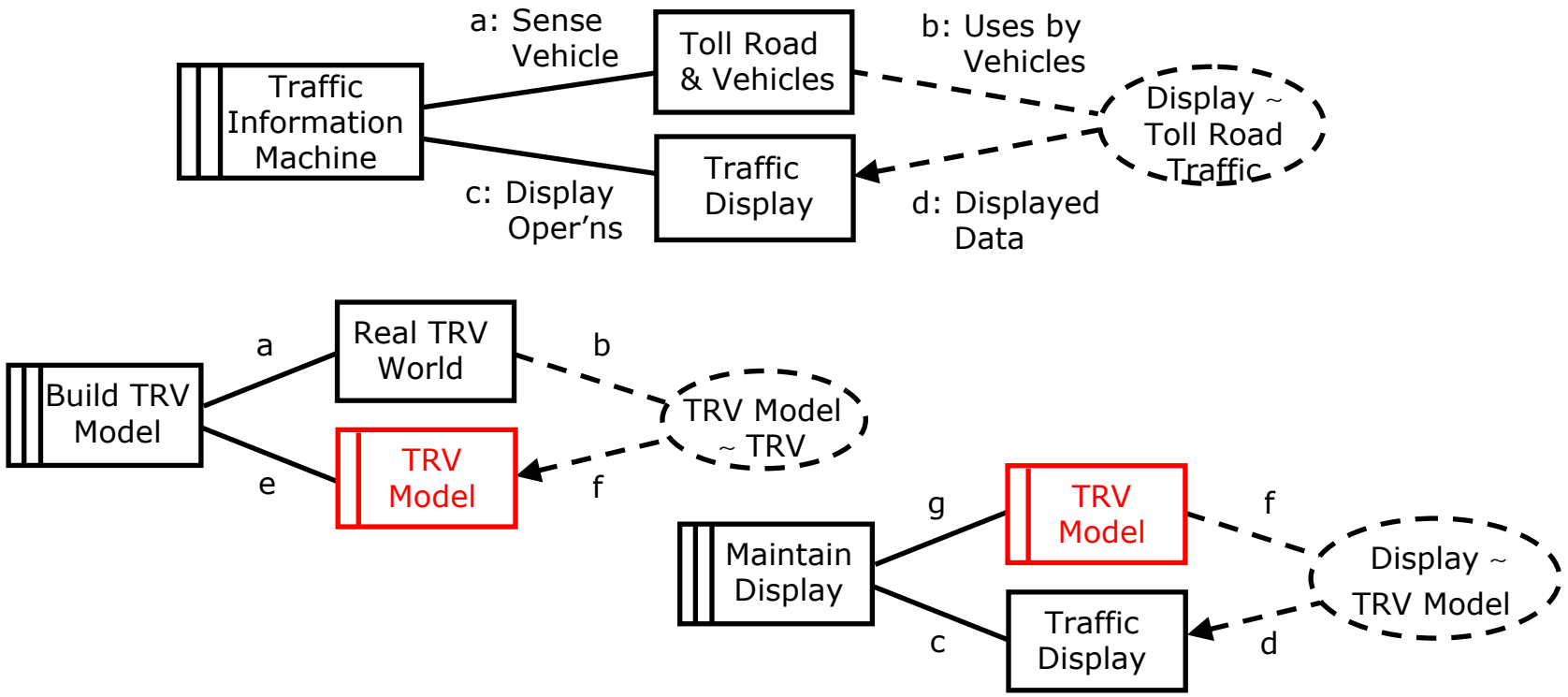
- A central purpose is maximum subproblem simplicity, to be achieved by assigning:
  - Different tempi to different subproblems
    - eg: membership and book borrowing
    - eg: scheduling trains and track maintenance
  - 'Faulty' and 'good' properties to different subproblems
    - eg: loans in good standing and loans in default
  - Different domain interactions to different subproblems
    - eg: book loans to members and to other libraries
  - Different domain views to different subproblems
    - eg: building and using a model domain
  - Different domain roles to different subproblems
    - eg: making a model of WorkPieces User
  - Functions of different criticality to different subproblems
    - eg: charging credit cards and sending reminders

Introducing a lexical domain



- Traffic Display machine with elaborate local variables
  - To remember each vehicle entry to toll road
  - To match each exit with corresponding entry
  - To count vehicles, uses, miles, etc …
  - … per road, road segment, entry/exit point, etc
- These local variables …
  - Constitute a model of the TR&V problem domain
  - Will probably be held in a database or in an assemblage of progrmaming objects whose persistence is assured by an object database
- They justify treatment as a distinct model domain with an accompanying problem decomposition

Introducing a lexical domain



- TRV Model ≠ Real TRV World
  - a ≠ g; b ≠ f
  - Model-only phenomena, eg: NULL values
- TRV Model ~ Real TRV World is always imperfect
  - Time lags, errors, abstraction, etc etc

Lexical domains more generally

- Model domains are one kind of lexical domain
  - Lexical domains extend availability of information in time and space
  - Lexical domains are physical (being part of the machine) but the information they carry is usually their chief significance
- A lexical domain may be:
  - A dynamic model of a part of the problem world
    - eg: TR&V Model
  - A static model of a part of the problem world
    - eg: Road Layout Model
  - A description of a requirement
    - eg: A railway timetable
  - A reification of a document
    - eg: A document in a WP program

Lexical and other domains generally

- We may identify three main kinds of domain
  - Lexical domains
    - Holding information in a physical form
    - eg: TR&V Model
  - Causal domains
    - Mechanical, electrical, inanimate natural
    - Obey causal laws (ultimately physics)
    - eg: Lift Gear
  - Biddable domains
    - Humans participating in problem world
    - Generally, do not obey causal laws
    - Can be requested to follow a procedure
    - eg: Library members, Lift users

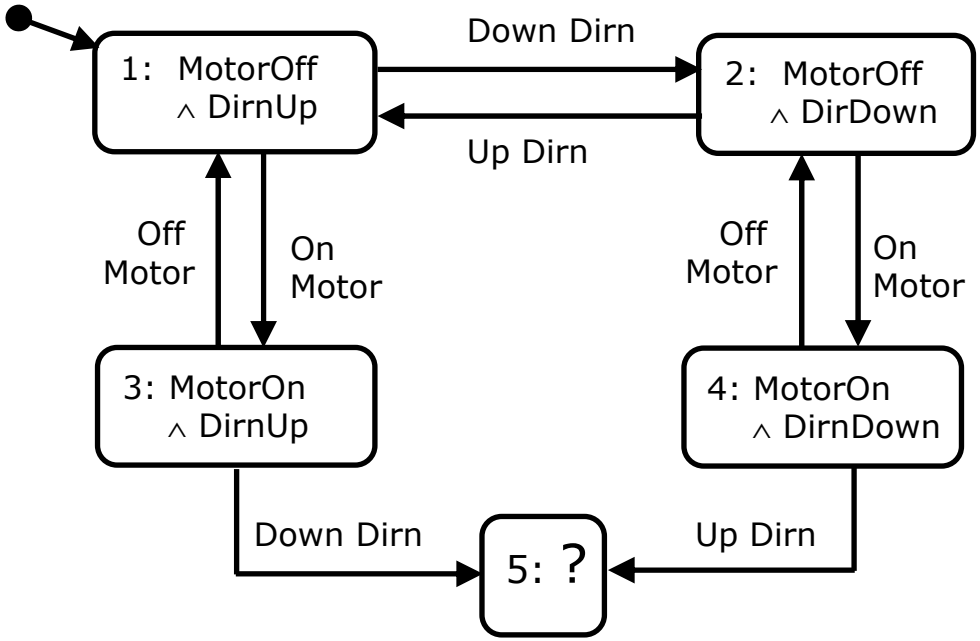
Addressing subproblem concerns

- A concern is any matter to which the developers must pay attention to obtain a good system
  - The basic ('frame') concern is $\mathcal{M}, \mathcal{W} \models \mathcal{R}$
  - There are also other more specific concerns that must be addressed to avoid serious failures
- Concerns are often about potential failures
  - We want to know, in each part of a development
    - What concerns can there be?
    - Which of them are important?
    - How can we address them effectively?
- We will distinguish
  - Subproblem concerns
  - Composition concerns

Some subproblem concerns

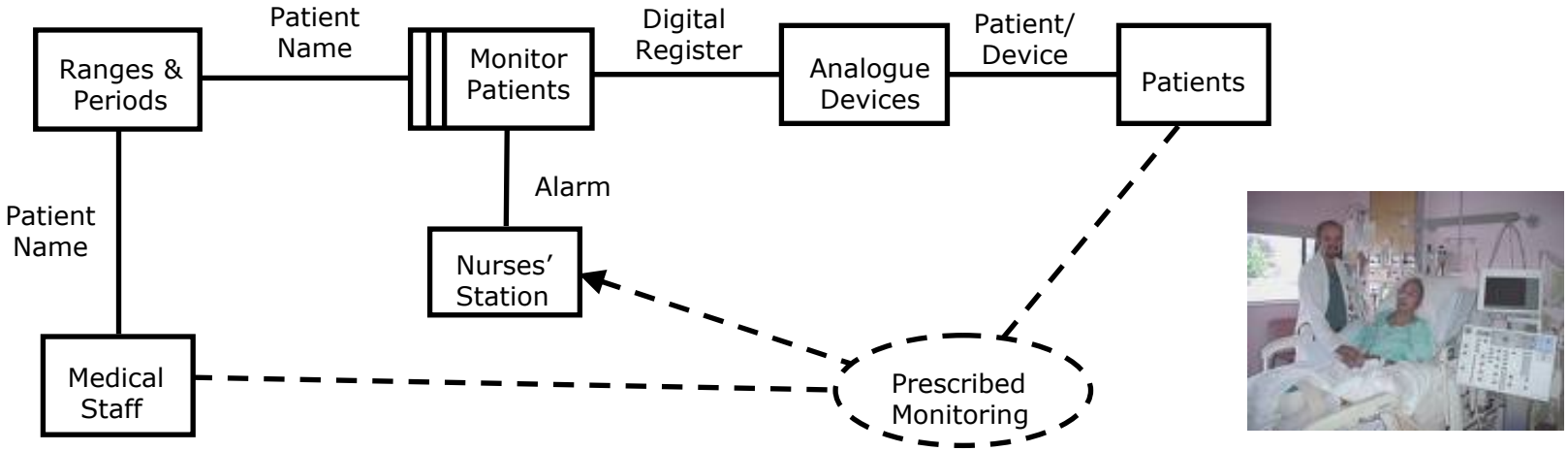
- Abortion
  - The world or machine aborts an interaction
- Breakage
  - The machine breaks a problem domain
- Initialisation
  - Incompatible initial states of machine and world
- Identities
  - Interacting with the wrong member of a set
- Reliability
  - Problem domain properties are not satisfied
- Totality
  - Some problem world conditions are ignored
- Overrun
  - Problem world goes too fast for the machine
- ...

Breakage: a subproblem concern



- "5: ?" is a 'broken state' of the Lift Gear
  - Behaviour in state 5 is entirely unpredictable
- Hence: Dirn must not be switched in MotorOn states

Identities: a subproblem concern

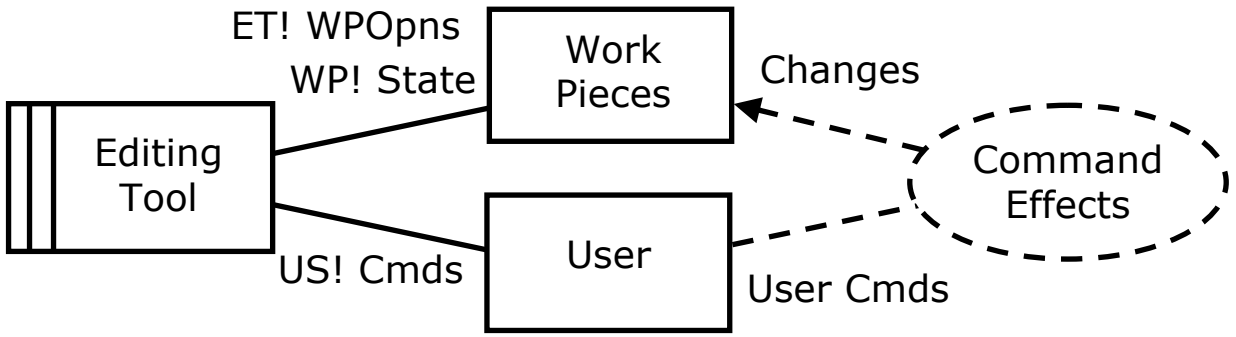


- How to identify monitored patient reliably?
  - Patients have names, and are in beds in ICU
  - Monitoring is specified by patient name
  - Devices (eg thermometers) are attached to patients
  - Devices are connected to machine's registers
- What is static, what is dynamic here?
- What can go wrong, and how?

Standard subproblem classes

- Why classify subproblems?
  - To focus attention on the relevant …
    - … notations and techniques
    - … concerns
- What determines classification?
  - Topology and types of problem domains
  - Type of requirement
- Examples of subproblem classes
  - Information display
    - eg: Toll Road Traffic problem
  - Required behaviour
    - eg: Provide Lift Service
  - Workpieces
    - eg: Word processing
  - Transformation
    - eg: Batch compiler

Workpieces: a standard subproblem class



- Simple Workpieces (no display!)
- The intuition:
  - Provide a tool for editing simple texts etc
- Problem domains
  - Workpieces: lexical, inert
  - User: biddable, active, autonomous
  - Command Effects: effects in WP of User commands
- Concerns
  - Completeness wrt User Cmds
  - Correctness (semantic rules not enforced by WP domain)

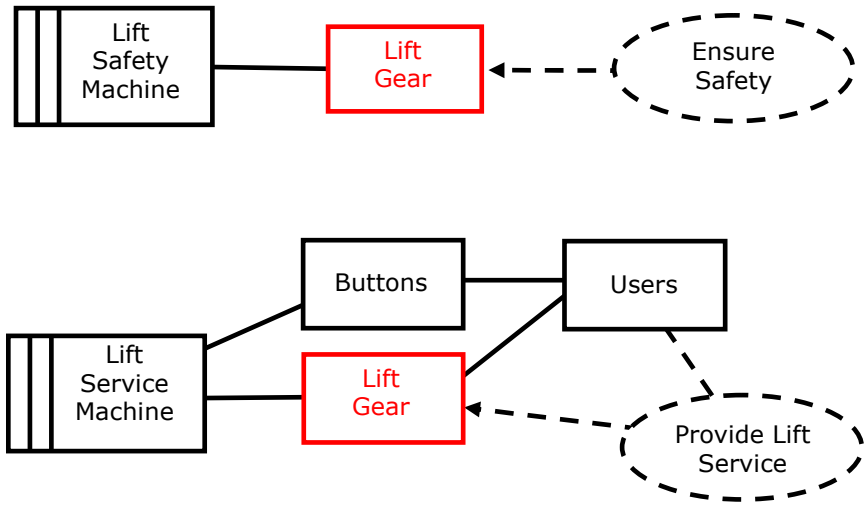
Composition concerns

- We consider subproblems in mutual isolation
  - eg: Lift Service vs. Lift Safety
  - eg: Build TR&V Model vs. Maintain TR&V Display
- Composition concerns arise in composing subproblems
  - Interference
  - Interleaving
  - Requirement conflict
  - Switching
  - Inconsistent domain properties
  - …
- By deferring composition concerns …
  - We see the subproblems more clearly
  - We see the composition concerns more clearly
  - We defer composition until components are known

Requirement conflict: a composition concern

- Requirement conflict
  - Two subproblems require inconsistent states or behaviours of the problem world

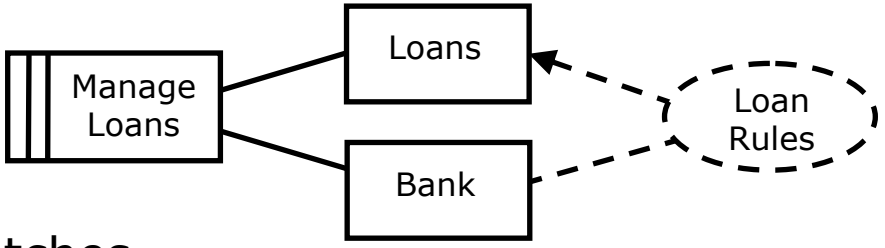
  - If a dangerous fault is detected, Lift Safety requires Emergency Brake on and MotorOff
  - If a User requests service, Lift Service requires MotorOn



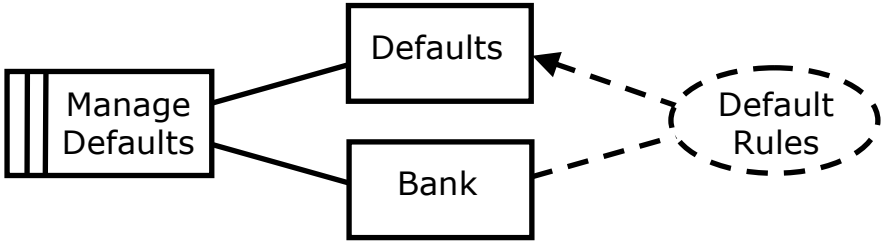
  - How do we expect to resolve this conflict?
    - Usually by requirement precedence
      - Lift Safety is more important than Lift Service

Switching: a composition concern

- Switching
  - A part of the problem world controlled under one regime must now be switched to control under another regime
  - What must happen when a loan customer defaults?



  - Management of the loan switches from ‘Loan Rules’ to ‘Default Rules’
  - The switch can take place only if the old process can be brought to a state in which the new process can validly start



10 minutes

Short break

75 minutes

3. Example (in more depth)

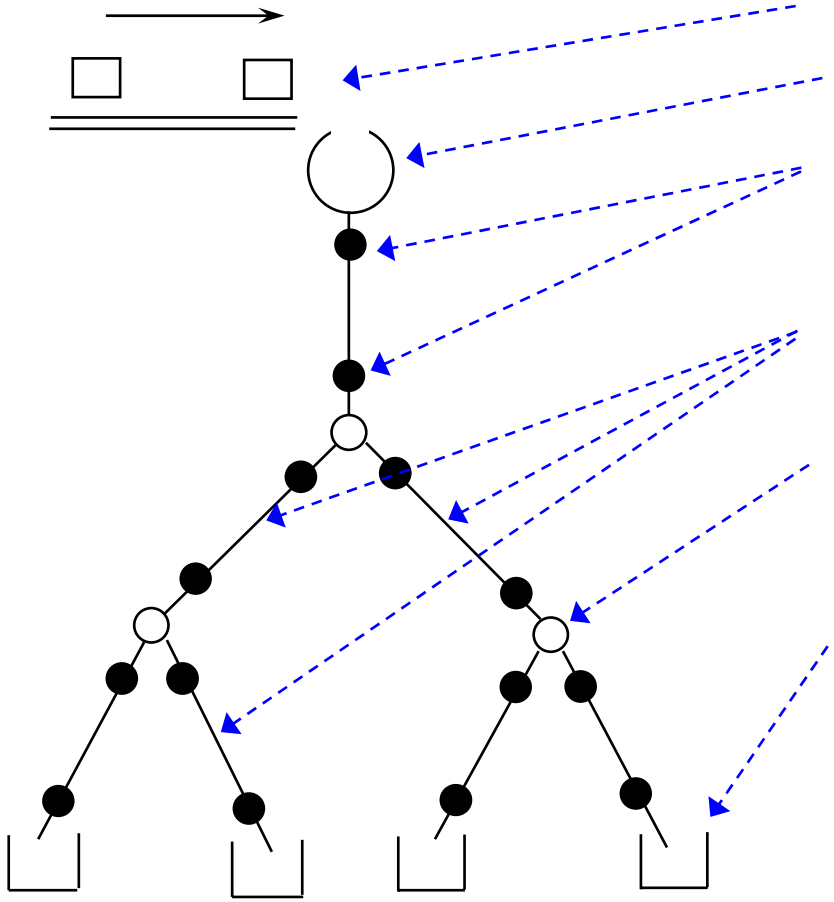
- Analysis of one problem
- Composing the subproblems

Package router control — 1

- A package router is a large machine used by delivery companies (eg Post Office, Fedex) to sort packages into bins according to bar-coded destination labels stuck to the packages.
- Packages are fed by a conveyor belt to a reading station where their destinations are read. They then slide by gravity down a tree of open pipes and binary switches, with bins at the leaves.
- The problem is to set the switches so that packages reach the correct bins. Misrouted packages are reported on a display station. An operator can issue commands to start or stop the feed conveyor.

Adapted from: *William Swartout and Robert Balzer; On the Inevitable Intertwining of Specification and Implementation; CACM 25 7, 438-440, July 1982*
[G Hommel; *Vergleich verschiedener Spezifikationsverfahren am Beispiel einer Paketverteilanlage*; Kernforschungszentrum Karlsruhe, TR, August 1980.]

Package router control — 2



feed conveyor

label reading station

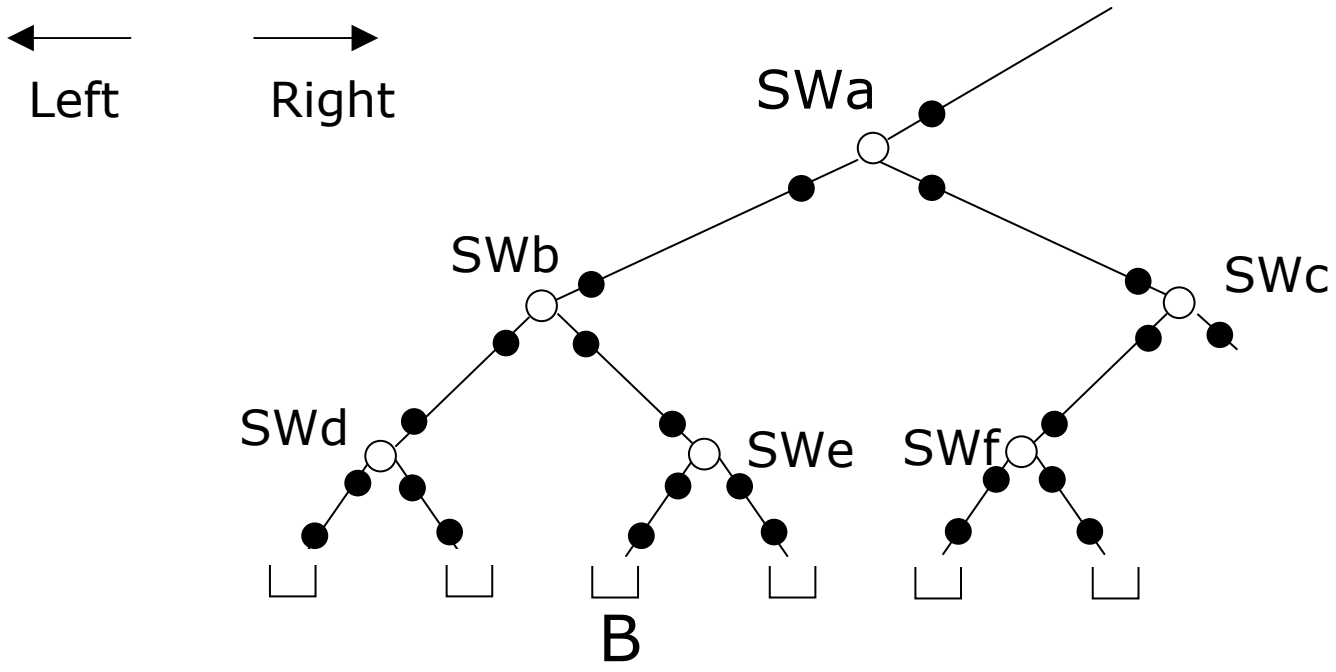
package sensors

open pipes

switch (not to be flipped unless empty)

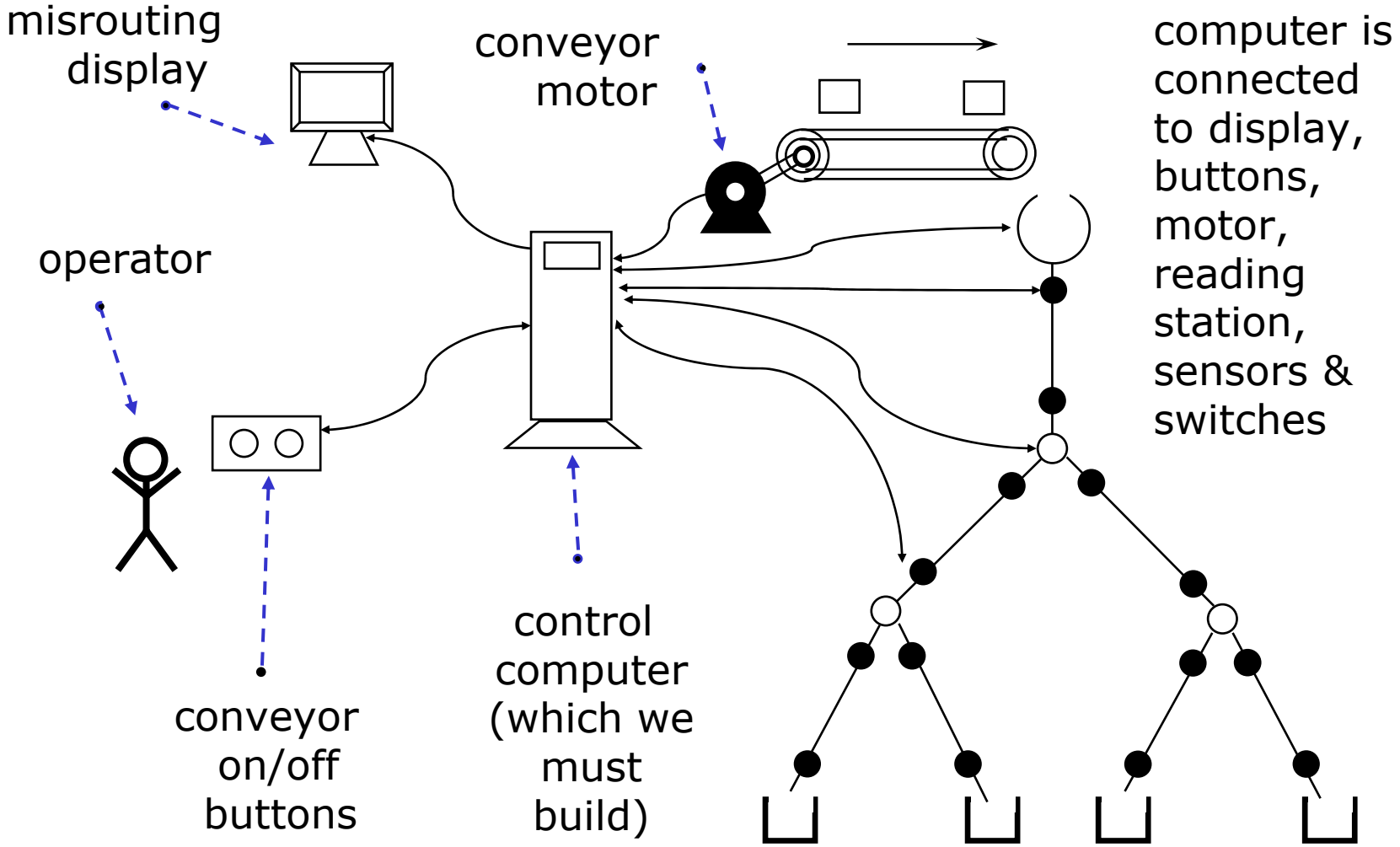
bin (corresponding to one or more destinations)

Package router control — 3

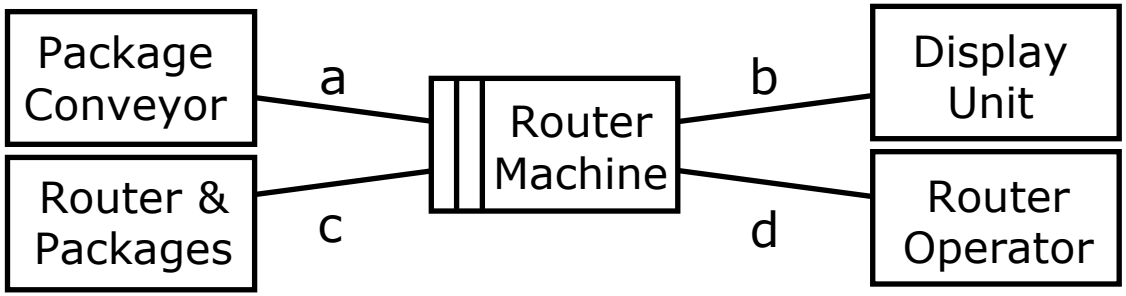


- Route at switch SWa to reach bin B:
  - If possible: Left at SWa, Right at SWb, Left at SWe
- Route at switch SWc to reach bin B:
  - No correct choice: package is already misrouted

Package router control — 4



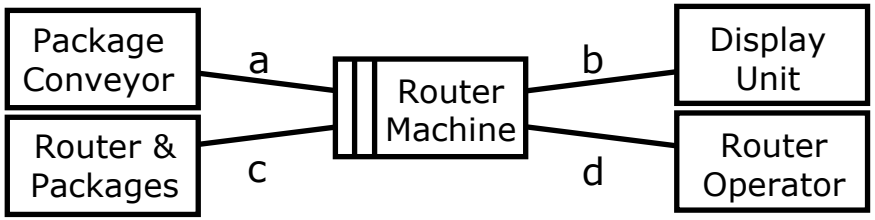
The machine and the world



a: RM! {OnC, OffC}
c: RM! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest),
SwPos[w], Sens[s]}

b: RM! {ShowMisrouting
(PkgId,Bin,Dest)}
d: RO! {OnBut, OffBut}

Designating the phenomena



a: RM!OnC ≈ event: the conveyor is switched on
RM!OffC ≈ event: the conveyor is switched off

---

c: RM!SwL[w] ≈ event: switch[w] is set to left
RM!SwR[w] ≈ event: switch[w] is set to right

RP!LRd(PkgId,Dest)
≈ event: package label is read giving id and destination
RP!SwPos[w] ≈ state: switch[w] is set to left (L) or right (R)
RP!Sens[s]} ≈ state: something is in front of sensor[s]

---

b: RM!ShowMisrouting(PkgId,Bin,Dest)
≈ event: message is displayed (id, bin, destination)

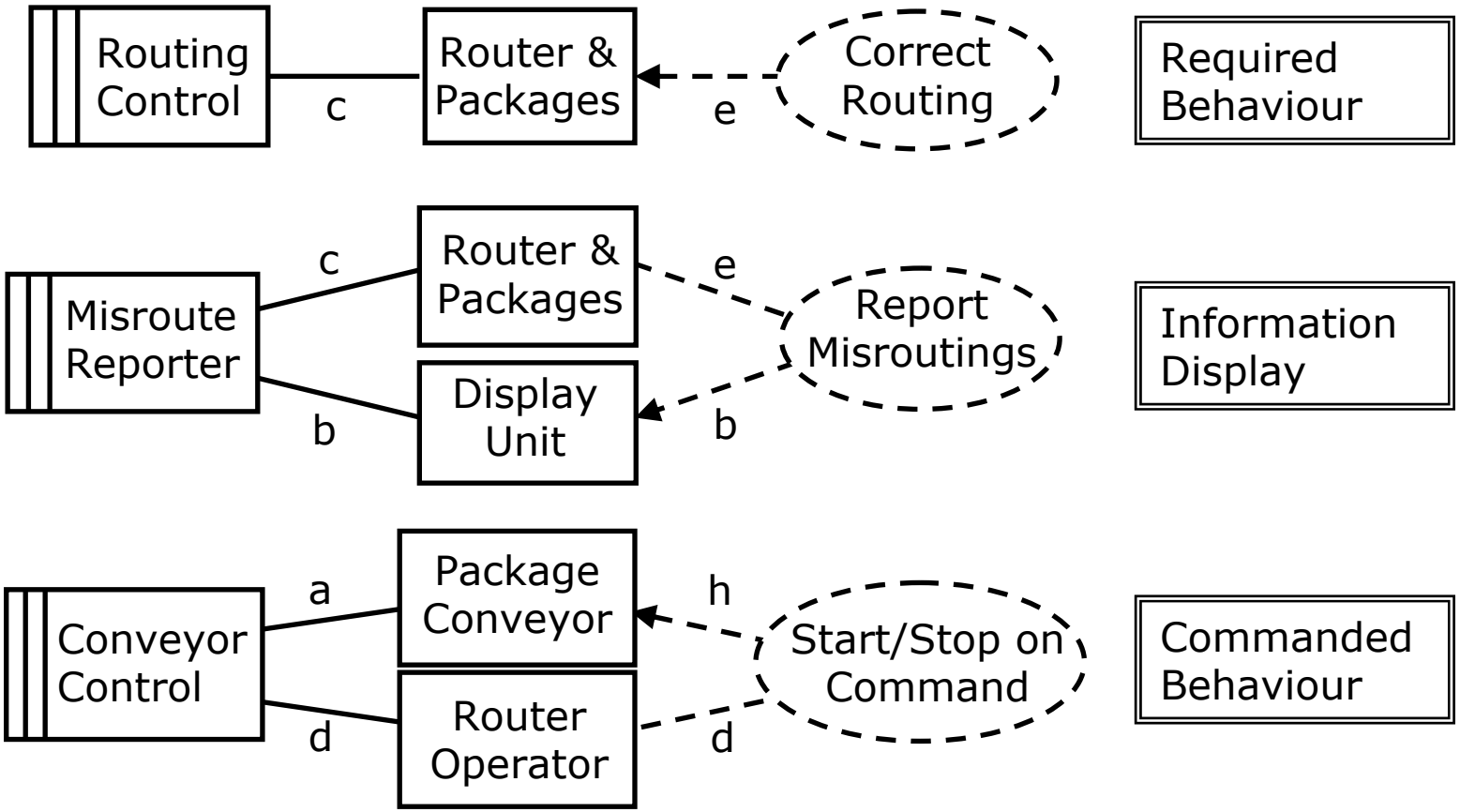
---

d: RO!OnBut ≈ event: the operator presses the On button
RO!OffBut ≈ event: the operator presses the Off button

Decomposition into subproblems

- A starting point in our structuring
  - Identify substantial subproblems of recognised classes
  - This won't be complete
    - We should expect to find more subproblems later
  - This may not be right
    - We may need to reconsider this structuring
- Three subproblems of recognised classes
  - Routing the packages
    - Problem class is required behaviour
  - Reporting misrouted packages
    - Problem class is information display
  - Obeying the operator's conveyor commands
    - Problem class is commanded behaviour

Initial decomposition: recognised subproblems



Package routing subproblem

c: RC! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest), SwPos[w], Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}

- Requirement: packages arrive at correct destination bins
  - PkgArr(p,b) ≈ package p reaches bin b
  - Assoc(d,b) ≈ bin b is associated with destination d
  - PDest(p,i,d) ≈ label of package p shows PkgId=i, Dest=d
- Problem World properties
  - Relate switch states (c) to switch settings (c)
  - Relate switch and sensor states (c) to bin arrivals (e)
  - Package behaviour (eg at sensors and switches, in pipes)
  - Association of bins with destinations

Package routing subproblem

c: RC! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest), SwPos[w], Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}

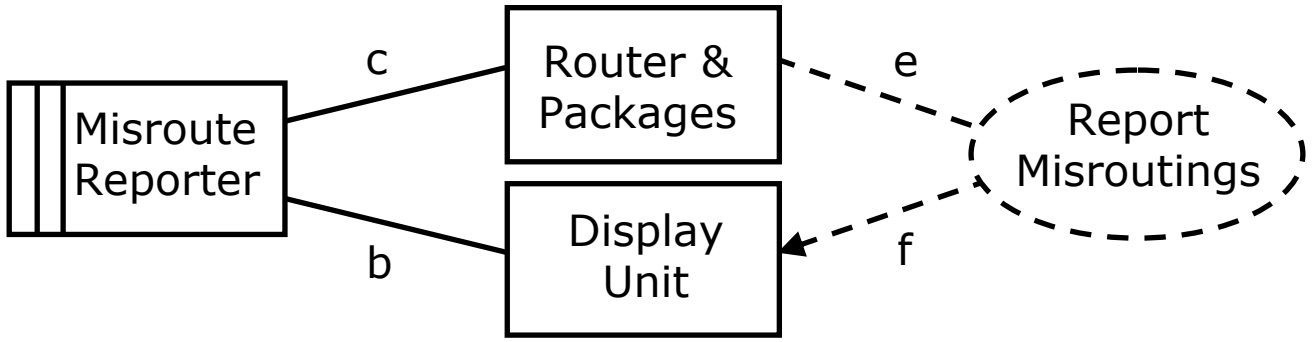
- Any questions about the requirement phenomena?
  - PkgArr(p,b) ≈ package p reaches bin b
  - Assoc(d,b) ≈ bin b is associated with destination d
  - PDest(p,i,d) ≈ label of package p shows PkgId=i, Dest=d

Package routing subproblem

c: RC! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest), SwPos[w], Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}

- Any questions about the problem world properties?
  - Relate switch states (c) to switch settings (c)
  - Relate switch and sensor states (c) to bin arrivals (e)
  - Package behaviour (eg at sensors and switches, in pipes)
  - Association of bins with destinations

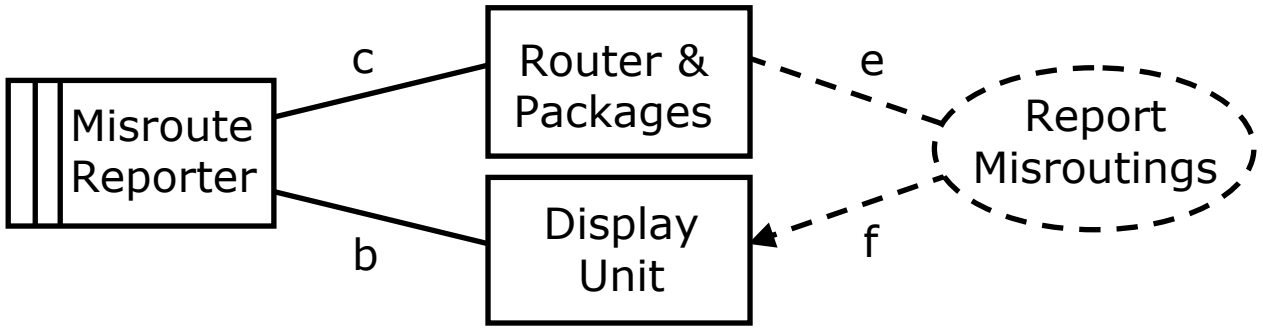
Misroute reporting subproblem



c: CRP! {LRd(PkgId,Dest), Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}
b: MR! {ShowMisrouting, (PkgId,Bin,Dest)}
f: {misrouting shown clearly on Display}

- Requirement: display details of misrouted packages
  - Bin b not associated with destination of arriving package p
- Problem World properties
  - Sensors, switches, arrivals
  - Format of messages input to Display (b,f)

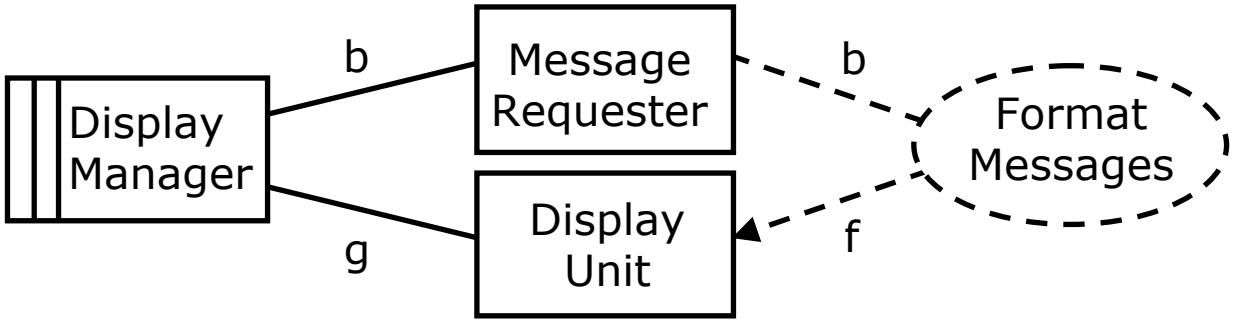
Misroute reporting subproblem



c: CRP! {LRd(PkgId,Dest), Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}
b: MR! {ShowMisrouting, (PkgId,Bin,Dest)}
f: {misrouting shown clearly on Display}

- Any questions about Display phenomena?
  - b: MR! {ShowMisrouting, (PkgId,Bin,Dest)}
    f: {"PkgId,Bin,Dest" shown on Display}

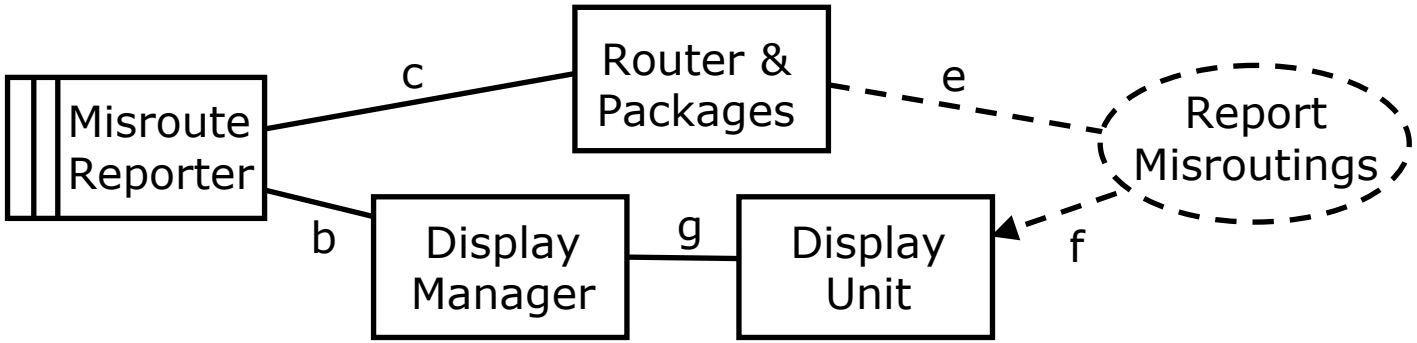
Display format subproblem



b: MR! {ShowMisrouting, (PkgId,Bin,Dest)}
g: DM! {Display Unit concrete operations}
f: {misrouting shown clearly on Display}

- Requirement: Manage display to format and position misrouting messages received from Message Requester
- Problem World properties
  - Display Unit behaviour relates concrete operations (g) to what is visible on the display screen (f)

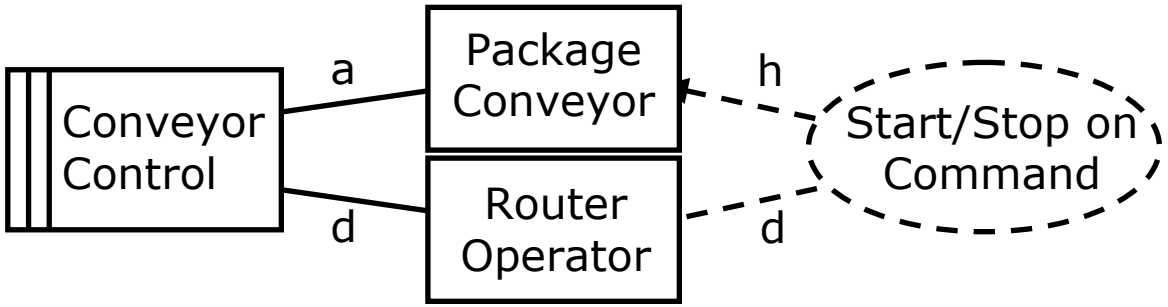
Misroute reporting with display management



c: CRP! {LRd(PkgId,Dest), Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}
b: MR! {ShowMisrouting, (PkgId,Bin,Dest)}
g: DM! {Display Unit concrete operations}
f: {misrouting shown clearly on Display}

- Display Manager problem domain
  - Is the machine from the subproblem “Display format”
  - Misroute Reporter plays role of Message Requester

Conveyor control subproblem

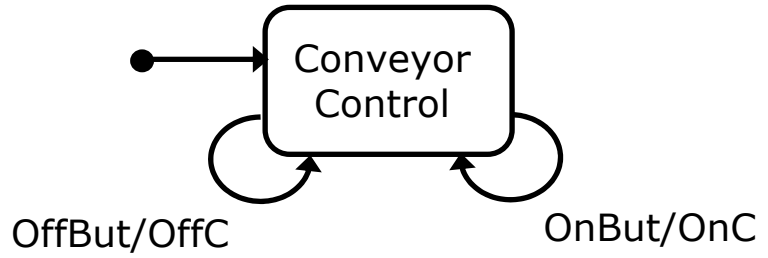


a: CC! {OnC, OffC}
d: RO! {OnBut, OffBut}
h: {Running, Stopped}

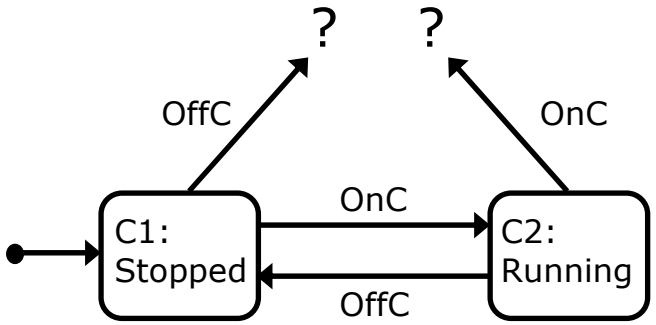
- Requirement: obey operator's conveyor commands
  - Following OnBut/OffBut conveyor is Running/Stopped
- Problem World properties
  - Relate {Running, Stopped} (h) to {OnC, OffC} (a)
  - The operator may cause {OnBut, OffBut} events at will

Breakage

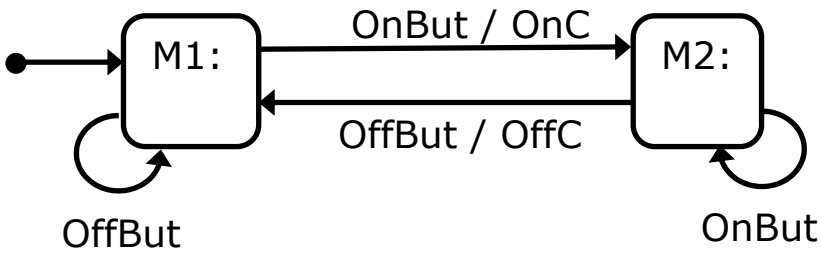
- The Conveyor Control specification must obviously be:



- But the conveyor may be fragile:



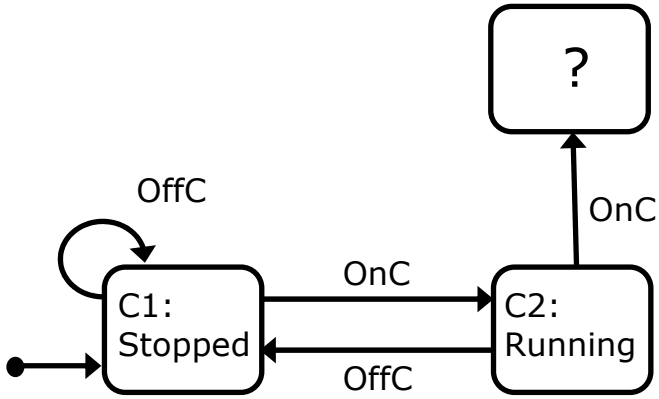
- In which case the Conveyor Control specification must be:



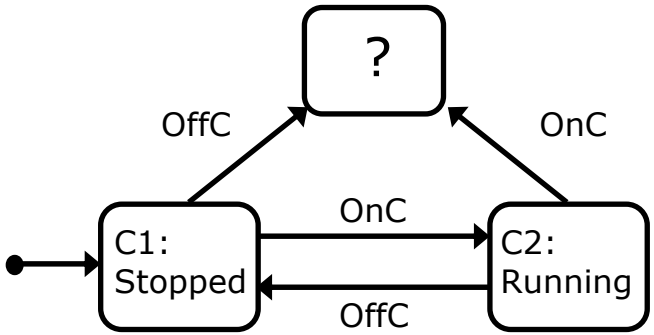
Initialisation

- Initialisation concern: harmonising the initial states of the machine and the problem world
  - Is the conveyor running or stopped initially?

- The conveyor may be like this ('?' denotes the unknown state, assumed broken)

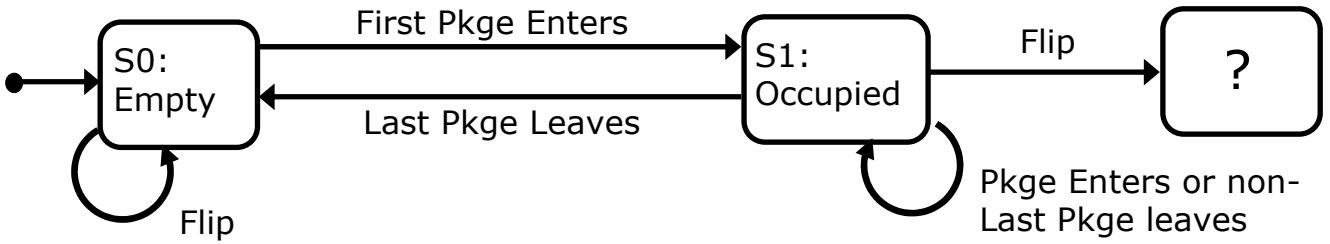


- What can be done if the conveyor is like this?



Breakage in another subproblem

- Breakage concern in the router mechanism
  - A router switch must not be flipped while there is a package in the switch



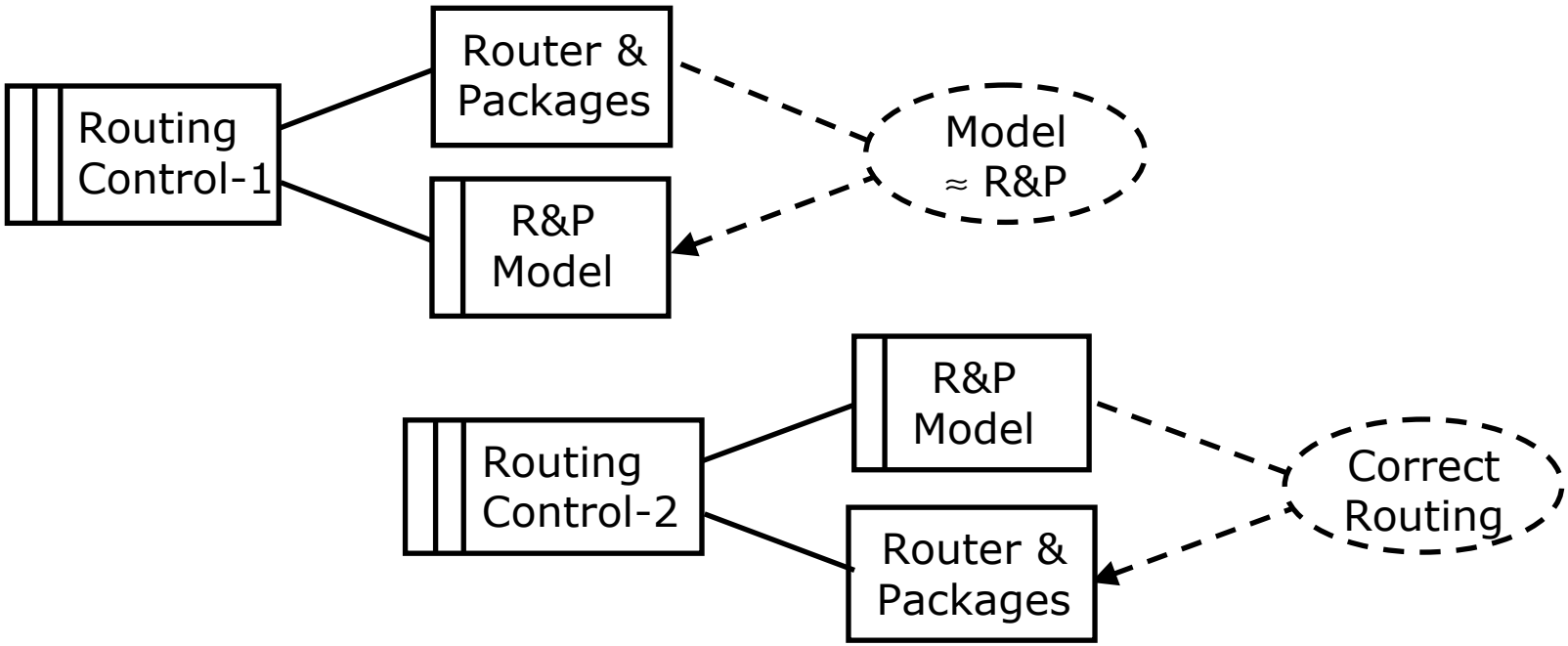
- How can breakage of the switch be avoided?

Package routing: information deficit concern

c: RC! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest), SwPos[w], Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}

- The problem raises an information deficit concern
  - Sens[s] flips on: to route the package, Routing Control machine must know the package destination
  - The state Sens[s] doesn't provide the information
- An information deficit concern requires a model domain
  - The model's purpose is to answer specific questions
    - "Which package has just arrived at sensor[s]?"
    - "What is its destination?"

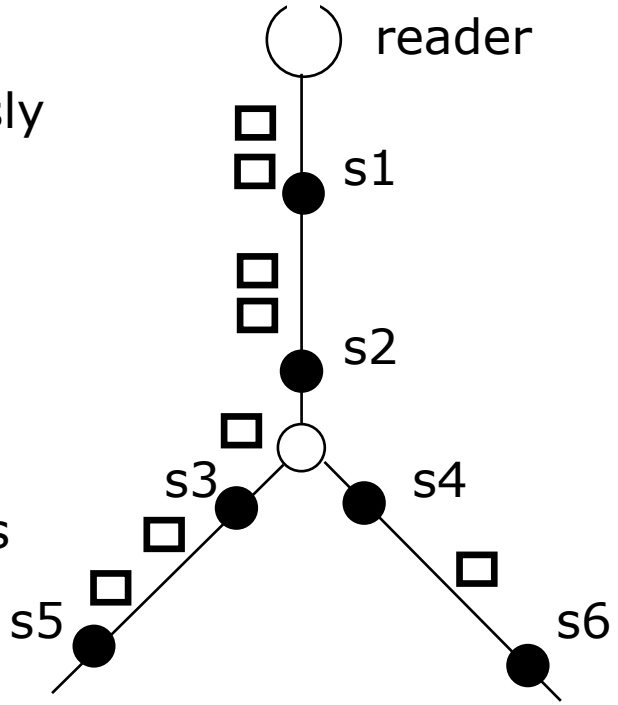
Building and using a model domain



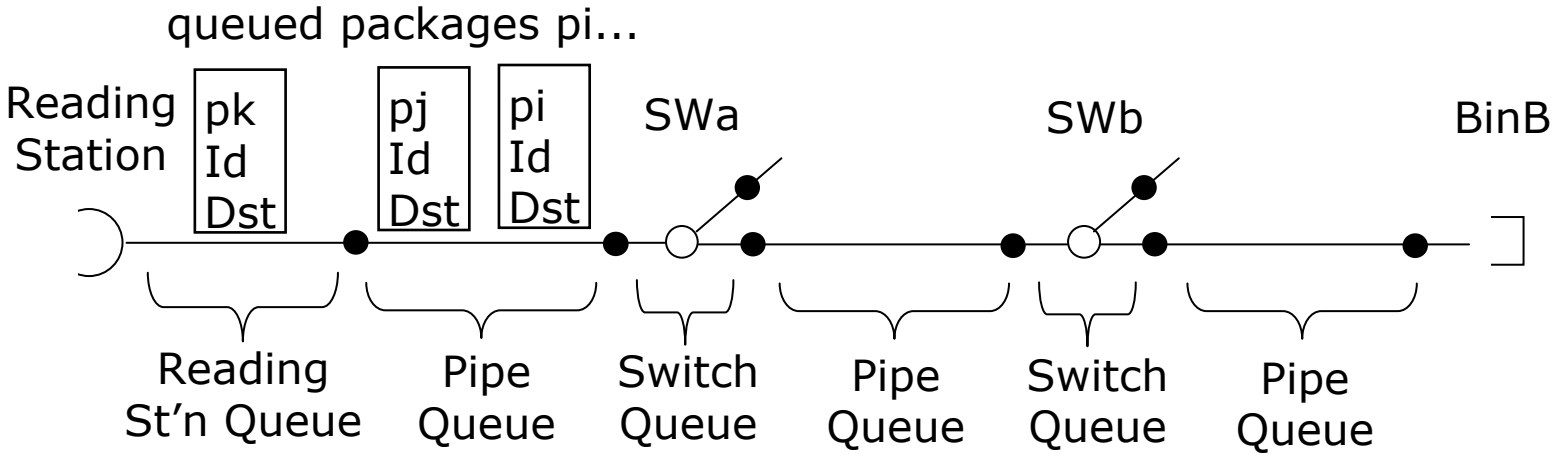
- Routing Control-1 builds the dynamic model domain
- Routing Control-2 uses the model domain to answer questions about the Router & Packages domain
  - To answer "What is its destination?" the Label Reading Station must be included in the model

Which package is now at sensor[s]?

- There is no overtaking in the pipes or switches
- When a package is read it joins the tail of a queue from reader to s1
- A now package at s1 was previously the head of the queue from reader to s1
- A package now at s3 or s4 was previously the head of the queue from s2 to s3 or s4
- The R&P Model domain is a dynamic model of these queues
  - Queued model elements are pairs (ID, Destination)
- The model domain provides the answers to the destination questions

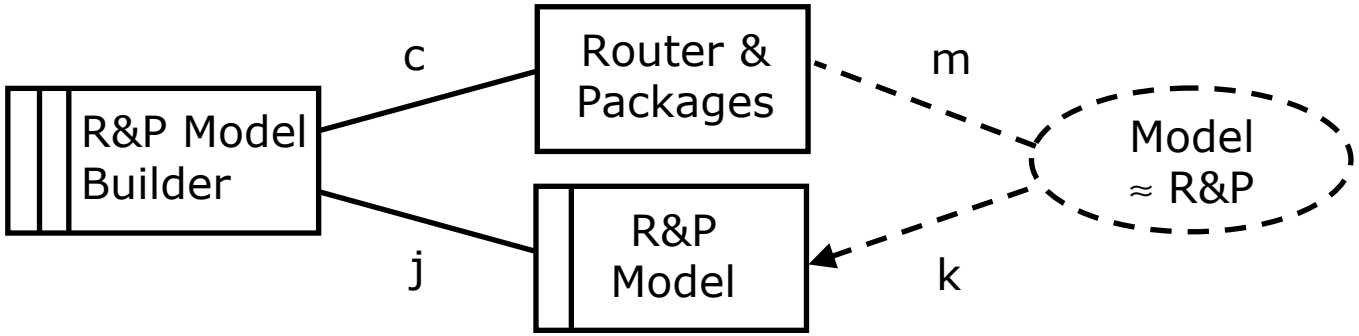


Which package is now at sensor[s]?



- No overtaking: packages queue in pipes and switches
- Each queue associated with its entry point $e \in \{\{\text{sensor}\} \cup RS\}$
  - Each package first enters the reading station queue
  - Package queues are in pipes and switches
    - For a switch, only one queue (exit) can be active at once
- The machine can maintain a dynamic model of these queues
  - Queued elements pi are pairs (Id, Destination)
  - (Id(p),Destination(p)) represents package p in the model

Building a model domain



c: RP! {LRd(PkgId,Dest),Sens[s],SwPos[w]}

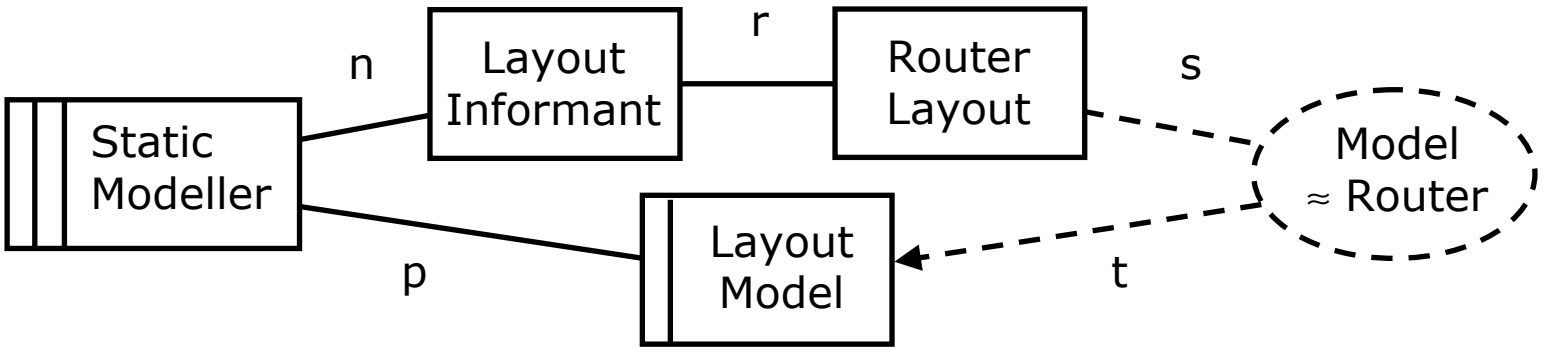
j: RMB! {NewPkg(p,i,d), EnQueue(p,q[e]), DeQueue(p,q[e])}

m: {PkgArr(p,s), PkgId(p,i), Dest(p,d)}

k: {PatS(s,p,i,d), NoneIn(q[e])}

- Information deficit concern:
  - When a package leaves queue[e] does it fall into a bin (if so, which one?) or join another queue (if so, which one)?
  - When a package arrives at a sensor, which queue has it just left?

Building a static model

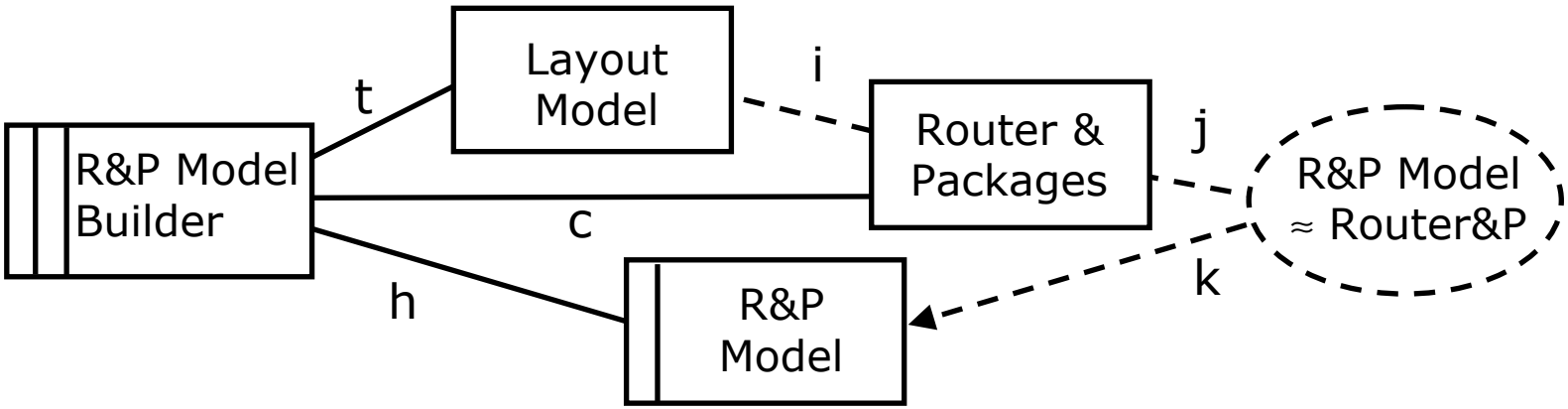


n: LI! {Data Entry Cmds}
p: SM! {Layout Model Operations}

r: RL! {Visible Layout}
s: {Router Layout States}
t: {Layout Model Relations}

- Can this model domain be built without a human informant?
- Is this model really static?
  - What if the router is extended by adding bins etc?
  - Can we assume that the layout doesn't change while package handling is in operation?

Using one model while building another



t: LM! {Layout Model Relations} i: RP! {Router Layout Relations}

j: {PkgArr(p,s), PId(p,i), PDest(p,i,d)}

h: RM! {NewPkg(p,i,d), EnQueue(p,q[e]), DeQueue(p,q[e])}

c: RP! {LRd(PkgId,Dest), Sens[s],SwPos[w]}

k: {LastArr(s,i,d), NoneIn(q[e])}

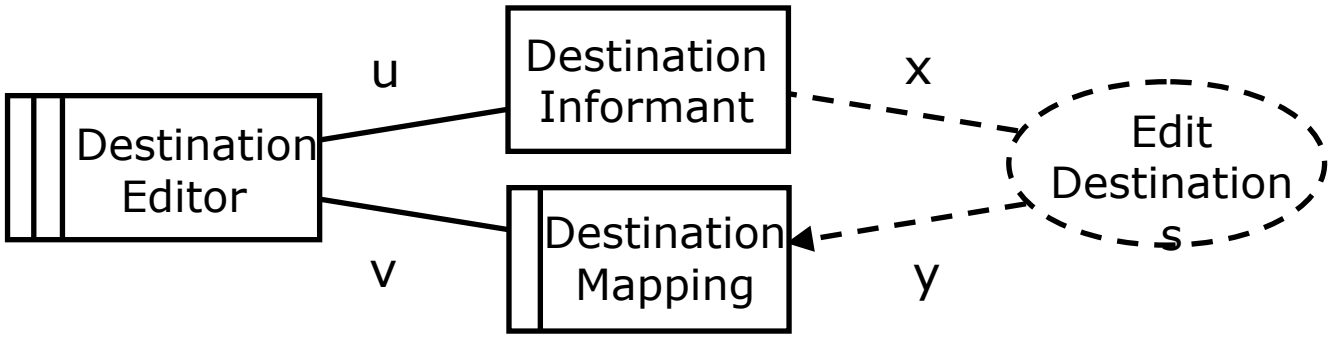
- R&P Model Builder uses the Layout Model
  - When a package arrives at a sensor, where did it come from and where is it going?

Package routing: another information deficit

c: RC! {SwL[w], SwR[w]}
RP! {LRd(PkgId,Dest), SwPos[w], Sens[s]}
e: {PkgArr(p,b), Assoc(d,b), PDest(p,i,d)}

- Another information deficit concern
  - Which bin b is destination d associated with?
- This information is not in the Router & Packages at all
  - The information requires a human informant
  - A model/description domain must be introduced

Associating destinations with bins



u: DI! {Edit Commands}

v: DE! {Mapping Operations}

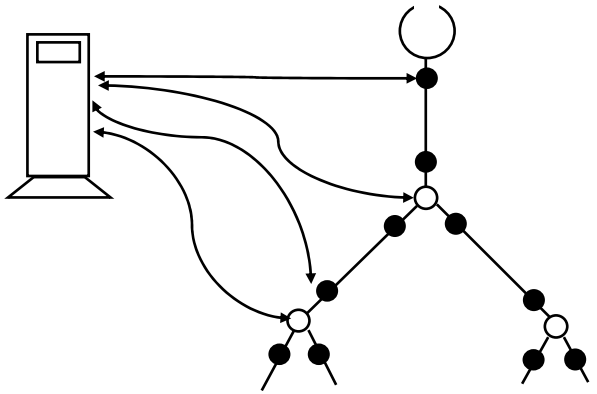
x: {Assoc(d,b)}

y: {MAssoc(d,b)}

- There is no escape from having a human informant
  - {Assoc(d,b)} phenomena are in human heads
  - We assume informant treats {Assoc(d,b)} correctly
- Destination Mapping is a model/description domain
  - Its phenomena {MAssoc(d,b)} are distinct from the phenomena {Assoc(d,b)}

An identities concern

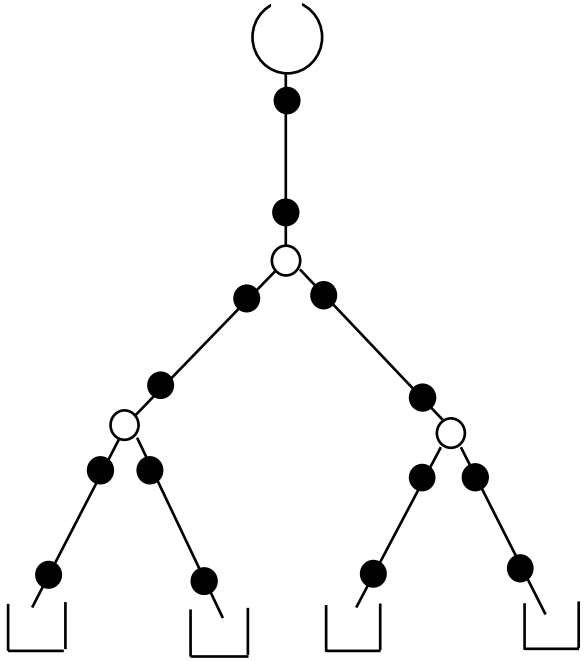
- Each sensor is attached to a machine port
- Each switch is attached to a machine output port for flipping its setting
- Each switch is attached to a machine input port for sensing its setting



- How can the mapping of devices to ports be established and maintained?
  - Standard technique: build and use a mapping domain
  - Use port names as device names
    - A switch now has two names
    - The Layout Model can then be built automatically

Automatic building of the layout model

- Build combined Id and Layout Model automatically without an informant
  - Find {port $\mapsto$ switch} mapping by SwL[pi], SwR[pi], SWPos[pi]
  - Set all switches L, run 1 package:
    - <pi1, pi2, pi3, …> are sensors on left edge
  - Find switches on left edge by running more packages
  - …

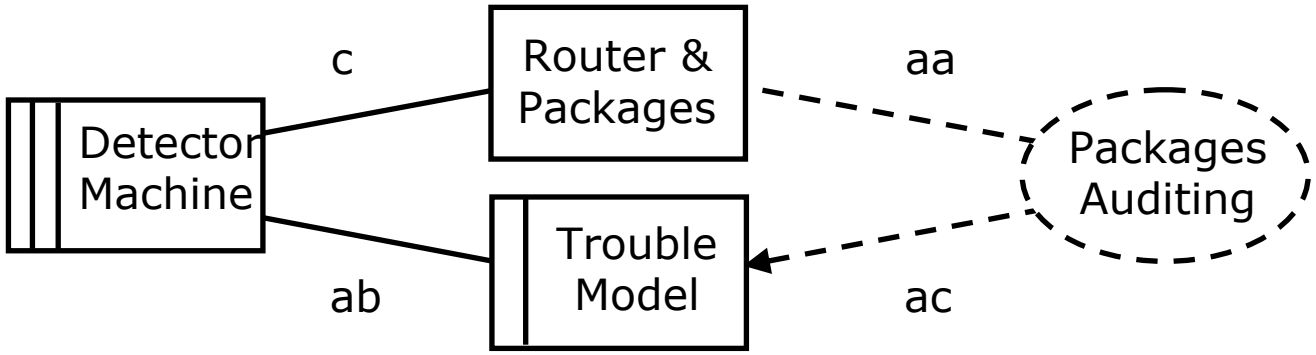


- What is the smallest number of packages that must be run to build a model of a balanced tree with 2^n-1 switches?

Domain reliability concern

- We have assumed certain domain properties:
  - Packages don’t get stuck
  - Packages don’t overtake each other
  - Packages are separated at sensors
- In the real package router these properties may fail
- We need to address this concern
  - Not by elaborating existing subproblems …
    … but by introducing a new subproblems
    - Audit subproblem to detect malfunction
    - Behaviour suproblem when malfunction detected
      - Turn conveyor off and keep it off
      - Complete routing of remaining packages so far as possible
      - Minimise package jamming

Detecting domain malfunction



c: RP! {LRd(PkgId,Dest), Sens[s], SwPos[w]}

aa: {Package Misbehaviour}

ab: AM! {Trouble Model Output Operations}

ac: {Trouble Model Information}

- Malfunctions: jam in switch, overtaking, …
  - detector Machine checks for misbehaviour
  - This is an information display problem
    - How much diagnosis of trouble?
    - How many separate problems? Why?

Interference

- A model domain is a shared variable
  - For the Build machine it is read/write
  - For the Use machine it is read only
  - Use machine assumes certain lexical domain properties ...
    - ... so composition requires mutual exclusion
    - Mutual exclusion can be enforced either in the model domain (eg by locks) or in the sharing machines
- Interference examples
  - Queues model update vs routing
  - Queues model update vs detecting malfunction

Interleaving

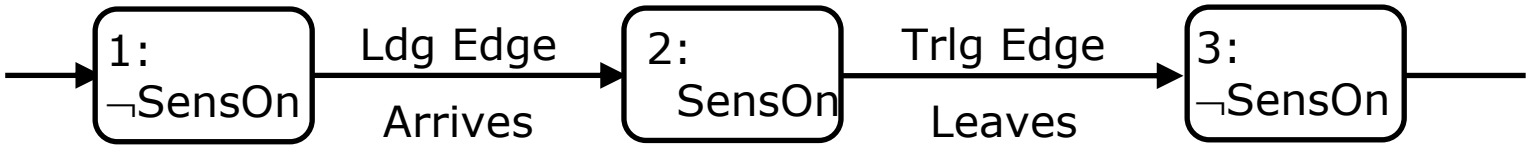
- A concern distinct from mutual exclusion
  - Usually raises a non-technical requirements question, for the customer
- Interleaving examples
  - Can the Destination Mapping change while routing is in progress?
  - Can the Router Layout model domain change while routing is in progress?
  - Does normal routing continue after a jam has been detected until all possible packages are in bins?

Conflict and precedence

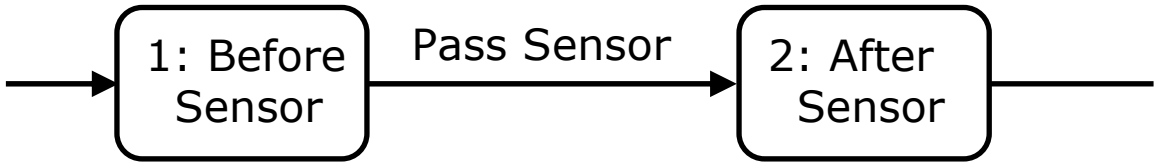
- Two subproblems cause contradiction in the domain
  - Both subproblem requirements can't be satisfied
  - Sometimes (but not always) resolved by priority
- Conflict example
  - Audit Machine detects a jam, conveyor is stopped; operator presses On Button
- How is the conflict to be resolved?
  - Operator has precedence?
  - Conveyor Controller has precedence?
  - 'Off' has precedence over 'On'?
- In a safety-critical system how can we guarantee safety has precedence over erroneous components?

Incommensurable description

- Different subproblems need different domain projections and different abstractions of phenomena
  - For auditing package jams: a finer abstraction



  - For reporting misrouting: a coarser abstraction



  - For dynamic model and package routing?

Switching

- A typical switching concern
  - Switching from normal operation to fault handling
    - When can the normal operation machine relinquish control of the domain?
    - When can the fault-handling machine take over control of the domain?
- Relinquishing control
  - The domain must be left in a ‘safe’ state
    - A juggler can’t stop while a ball is in the air
- Taking over control
  - The initialisation concern must be addressed for the machine taking over control
  - A termination concern must be addressed for the machine relinquishing control

10 minutes

Short break

15 minutes

4. Practicalities

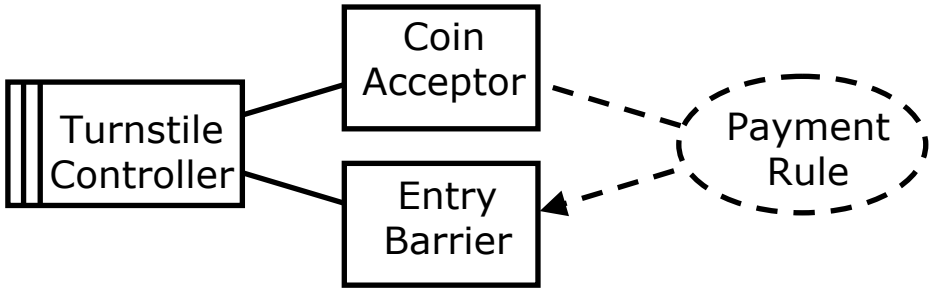
- Applying problem frames
- Some general principles
- Final discussion

Applying problem frames

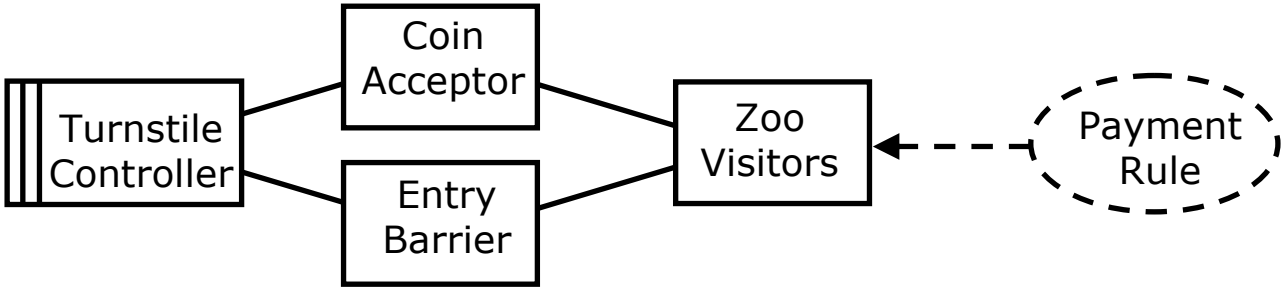
- A heavyweight approach
  - Full decomposition into problem frames
  - Fully explicit composition of subproblems
  - Software implementation by refining and transforming
    - ‘Refactoring’ subproblem machines
- A lightweight ubiquitous approach
  - Apply basic PF techniques as a regular practice
  - Use PF principles as a crosscheck on development
- A lightweight occasional approach
  - Try PF ideas and techniques on finding a difficulty
  - Try PF ideas and techniques on finding an opportunity
- A rejectionist approach
  - Ignore PF ideas and principles …
  - … they don’t address any of my difficulties

PF principle: depth in the world — 1

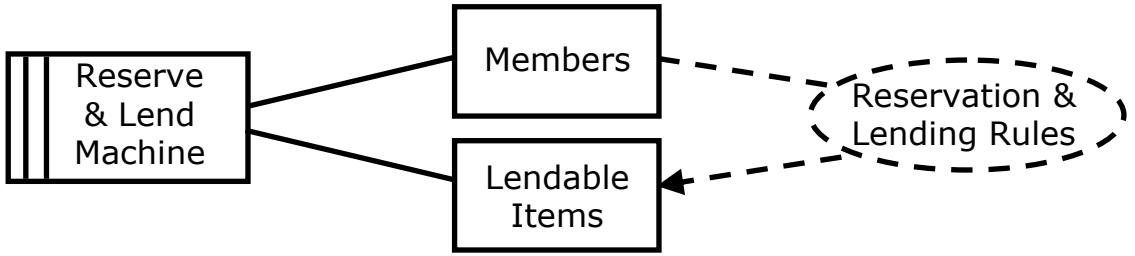
- The problem world gives meaning to the requirement
  - Is this right: (#Coins — #Pushes) ≤ 1 ?



  - Or is this right: #Coins ≥ #Pushes ?
- The answer depends on behaviour of zoo visitors

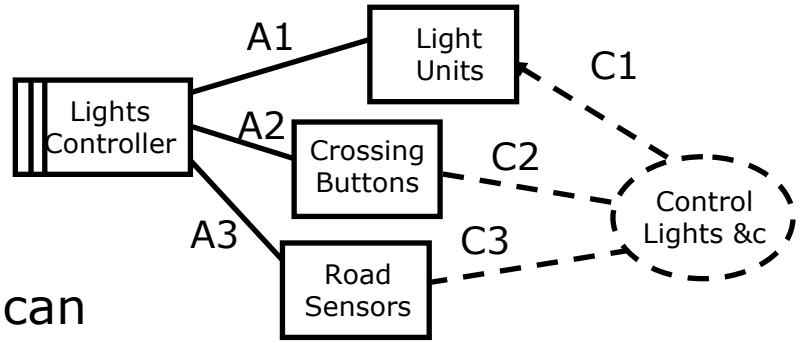


PF principle: depth in the world — 2



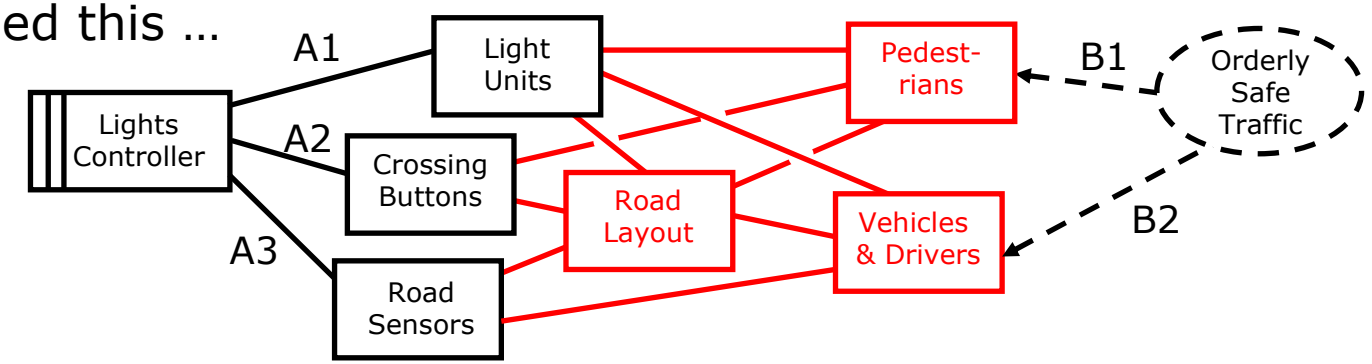
- Important problem domain behaviour may occur …
  - … out of sight of the machine …
  - … unreported to the machine
- Any possible event may be relevant
  - eg: member dies
  - eg: member is bankrupt
  - eg: book is withdrawn from library
  - eg: book's barcode label is lost
  - eg: author changes name
  - …

PF principle: depth in the world — 3



- With only these domains, how can we understand the problem world?
  - eg: what do the sensor states mean?
  - eg: what light sequences are required?

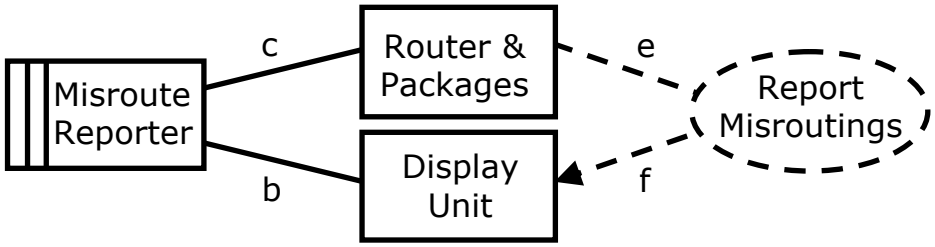
- We need this …



  - … and careful study of the deeper domains

PF principle: subproblem decomposition

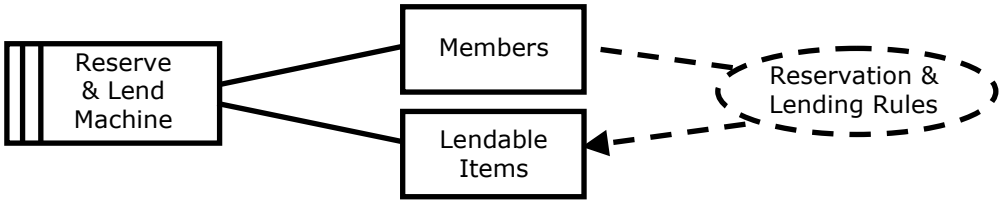
- Why is subproblem decomposition helpful?



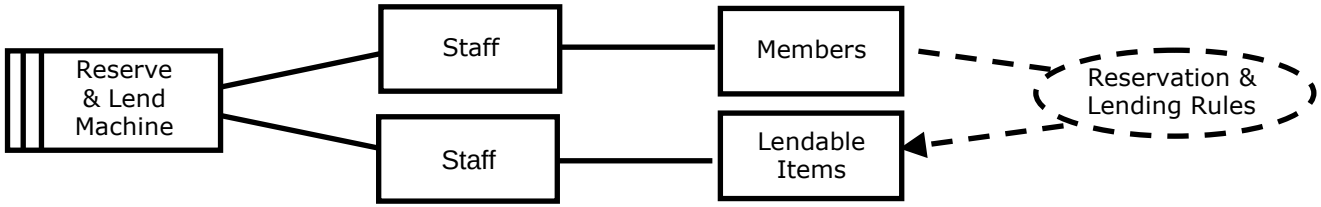
- Allows a helpful separation: low coupling, high cohesion
  - Subproblem function is (almost) an isolated matter
- Subproblems are maximally simple
  - Heuristics for single tempo, single role etc
- Ensures an appropriate granularity
  - No a priori assumptions about subproblem span
- Allows capturing experience in problem classes
  - Identifying important concerns and normal design
- $\mathcal{M}, \mathcal{W} \models \mathcal{R}$ exposes problem domain property assumptions
  - What is the assumed behaviour of Router & Packages?

PF principle: subproblem simplification

- Aren't PF subproblems very simplified (for example, we seemed to ignore the role of library staff in Reservations & Loans)?
- Yes: they are very simplified, and should be viewed as abstractions of the more complex reality — eg …



… is, in reality, more like this 'full' version:



- The simplification assumes 'correct' staff behaviour, in which the staff act 'transparently'
  - The 'full' version can be considered as necessary

PF principle: know what is being described

- Descriptions are associated with problem parts
  - Machine, problem world, requirement
- We must choose a clear view of the phenomena
  - eg: what does 'X is mother of Y' mean?
  - Appropriate view depends on the problem
    - eg: 'mother' in labour ward, 'mother' in school, 'mother' in divorce court, …
- In difficult cases write designations for phenomena
  - eg: mother(X,Y) ≈
    "X is the biological mother of Y in the sense that Y is born from a fertilised egg of X"

Short PF bibliography — 1

Steven J Bleistein, Karl Cox and June Verner; *Problem Frames Approach for e-Business Systems*; in Proceedings of the 1st International Workshop on Advances and Applications of Problem Frames, 2004.

Ian K Bray and Karl Cox; *The Simulator; Another, Elementary Problem Frame?* in Proceedings of the 9th International Workshop on Requirements Engineering: Foundation For Software Quality (REFSQ03), 2003.

Karl Cox, Jon G Hall and Lucia Rapanotti; *A Roadmap of Problem Frames Research*; Information and Software Technology Volume 47 Number 14, pages 891-902, 2005.

K Cox, J G Hall and L Rapanotti eds, Journal of Information and Software Technology: Special issue on Problem Frames, Elsevier, Volume 47 Number 14, November 2005.

Jon G Hall, Michael Jackson, Robin Laney, Bashar Nuseibeh and Lucia Rapanotti; *Relating Software Requirements and Architectures Using Problem Frames*; in Proceedings of the 2002 International Conference on Requirements Engineering (RE'02), Essen, 2002.

Jon G. Hall, Lucia Rapanotti and Michael Jackson; *Problem frame semantics for software development*; Software and Systems Modeling, Springer Volume 4 Number 2, pages 189-198, May 2005.

Jon G Hall, Lucia Rapanotti, Michael Jackson; *Problem-Oriented Software Engineering;* Open University Technical Report 2006/10, 12 October 2006.

Denis Hatebur and Maritta Heisel; *Problem Frames and Architectures for Security Problems*; in Computer Safety, Reliability and Security, pages 390-404, LNCS 3688, Springer Verlag, 2005.

Michael Jackson; *Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices*; Addison-Wesley, 1995.

Short PF bibliography — 2

Michael Jackson; *The Real World*; in *Millennial Perspectives in Computer Science*; Jim Davies, Bill Roscoe, Jim Woodcock eds; Proceedings of the 1999 Oxford-Microsoft symposium in honour of Sir Antony Hoare, pages 157-173; Palgrave, Basingstoke, England, 2000.

Michael Jackson; *Problem Analysis and Structure*; in *Engineering Theories of Software Construction*, Tony Hoare, Manfred Broy and Ralf Steinbruggen eds; Proceedings of NATO Summer School, Marktoberdorf; IOS Press, Amsterdam, Netherlands, August 2000.

Michael Jackson; *Problem Frames: Analysing and Structuring Software Development Problems;* Addison-Wesley, 2000.Michael Jackson; *Why Program Writing Is Difficult and Will Remain So*; Information Processing Letters Volume 88; Proceedings of "Structured Programming: The Hard Core of Software Engineering", a symposium celebrating the 65th birthday of Wladyslaw M Turski, Warsaw 6 April 2003, pp13-25.

Michael Jackson; *Some Structural Relationships Among Models In the Development of Software-Intensive Systems;* Proceedings of Dagstuhl Workshop 06351 on Methods for Modelling Software Systems, August 27 to September 1st 2006, 2007.

Robin Laney, Leonor Barroca, Michael Jackson and Bashar Nuseibeh; *Composing Requirements Using Problem Frames*; in Proceedings of the 12th International Conference on Requirements Engineering RE'04, IEEE CS Press, 2004, pages 113-122.

Robin Laney, Thein T Tun, Michael Jackson and Bashar Nuseibeh; *Composing Features by Managing Inconsistent Requirements*; in Proceedings of the International Conference on Feature Interactions in Telecommunications and Software Systems, Grenoble, September 2007.

Short PF bibliography — 3

Jeremy T Lanman; *A Software Requirements Engineering Methodology Comparative Analysis: Structured Analysis, Object-Oriented Analysis, and Problem Frames*; Department of Computing and Mathematics, Embry-Riddle Aeronautical University, 8 April 2002.

Derek Mannering, Jon G Hall and Lucia Rapanotti; *Towards Normal Design for Safety-Critical Systems*; in Fundamental Approaches to Software Engineering, LNCS 4422, Springer Verlag 2007.

Thomas Rottke, Denis Hatebur, Maritta Heisel, Monika Heiner; *A Problem-Oriented Approach to Common Criteria Certification*; Proceedings of SAFECOMP 2002, Anderson et al eds, LNCS 2434, pages 334-346, Springer Verlag 2002.

M Salifu, B Nuseibeh and Yijun Yu; *Specifying Monitoring and Switching Problems in Context;* Proceedings of the 15th International Requirements Engineering Conference (RE07), Delhi, India, 2007.

Robert Seater and Daniel Jackson; *Requirement Progression in Problem Frames Applied to a Proton Therapy System*; Proceedings of the 14th International Requirements Engineering Conference (RE06), Minneapolis USA, 2006.

Yun Lin; *Applying problem frames to modeling 'abstraction' concepts*; in Proceedings of the 1st International Workshop on Advances and Applications of Problem Frames, 2004.

Pamela Zave and Michael Jackson; *Four Dark Corners of Requirements Engineering*; ACM Transactions on Software Methodology Volume 6 Number 1, pages 1-30, July 1997.

Thank you