Problem Frames

A Lecture

Michael Jackson
MIT 6898
06 March 2002

jacksonma@acm.org

Overview

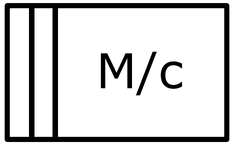
- Introduction
- Some tiny example problems
- Problem frames and concerns
- A larger problem and its decomposition
- Decomposition concerns
- Summary

Introduction — 1

- We will focus on problems
  - Describing, classifying, structuring problems
  - Recognising concerns and difficulties ...
  - ... but never exploring solutions (well, hardly ever)
- The approach is informal
  - Appropriate, because problems are in the world, and most of the world is informal
- Presentation is not rigorous
  - Much notation and terminology assumed or implied
- No particular development process is assumed
  - It’s sometimes good to think of the solution first
  - Problem analysis can be *ex post facto*
    - “If this is the solution, what was the problem?”

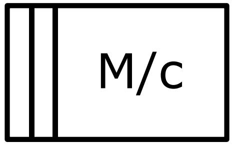
Introduction — 2

- Developing software is building a machine ...



Introduction — 2

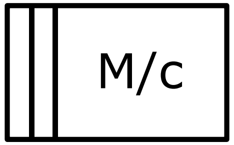
- Developing software is building a machine ...



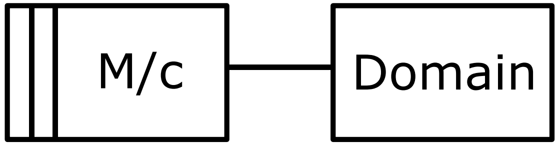
- One problem, one machine
- The machine is a general-purpose computer ... ... specialised by software
- The machine may be distributed
- One computer may support many machines
- Problem decomposition gives many subproblems ... ... and so many machines

Introduction — 2

- Developing software is building a machine ...

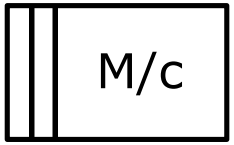


- ... to solve a problem in a given domain (a part of the world) ...

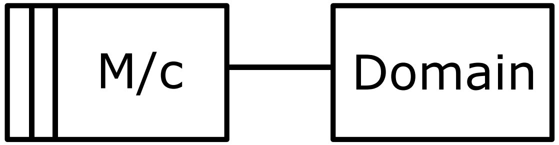


Introduction — 2

- Developing software is building a machine ...



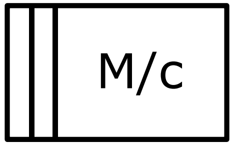
- ... to solve a problem in a given domain (a part of the world) ...



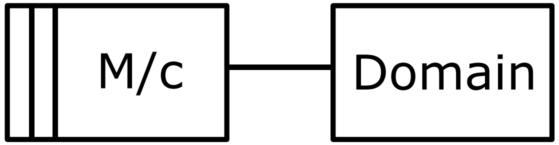
- The machine and the problem domain interact ...
  ... at an interface of shared phenomena (events, states, etc etc)
- Usually we need to structure the problem domain ...
  ... and to structure the problem into subproblems
- The machine in one subproblem may be a part of the problem domain in another subproblem

Introduction — 2

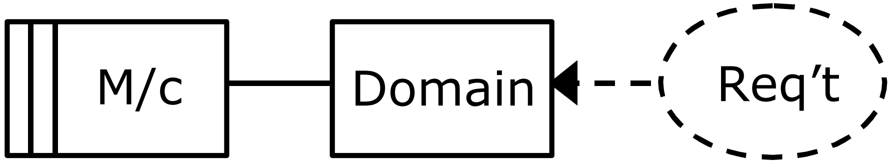
- Developing software is building a machine …



- … to solve a problem in a given domain (a part of the world) …

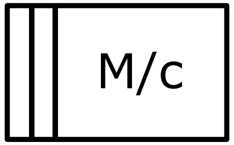


- … to meet a customer’s needs (the requirement)

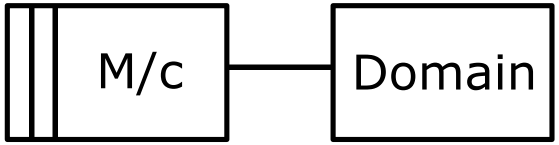


Introduction — 2

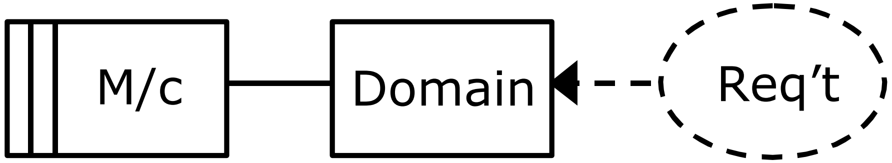
- Developing software is building a machine ...



- ... to solve a problem in a given domain (a part of the world) ...



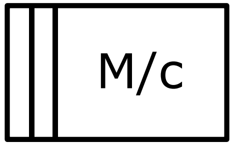
- ... to meet a customer’s needs (the requirement)



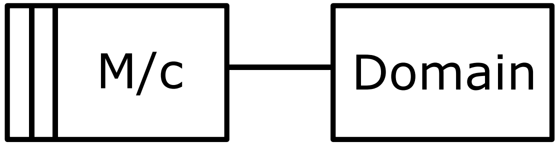
- The customer’s requirement is for some effect (or property or behaviour) in the problem domain
- ←--- means that the requirement adds a constraint to the domain’s intrinsic properties or behaviour

Introduction — 2

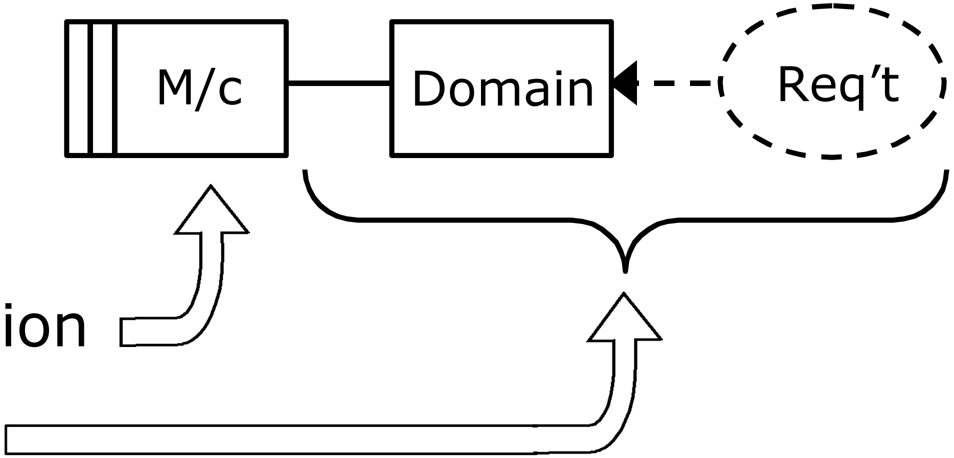
- Developing software is building a machine ...



- ... to solve a problem in a given domain (a part of the world) ...



- ... to meet a customer’s needs (the requirement)



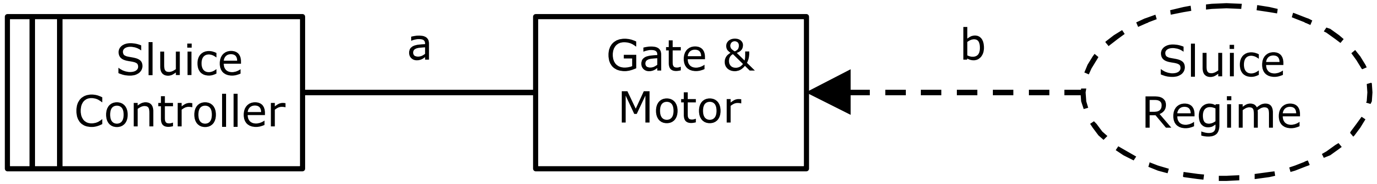
- The machine is the solution

- The problem is here:
  it is not inside the machine

Some Tiny Examples — 1

- A computer system is needed to control the sluice gate in a simple irrigation system. The gate must be held in the fully *Open* position for 10 minutes in every three hours and otherwise kept in the fully *Shut* position.
- The gate is opened and closed by vertical screws driven by a small motor controlled by *Clockwise*, *Anti-clockwise*, *On* and *Off* pulses. There are sensors at the *Top* and *Bottom* of the gate travel; at the top it is fully open, at the bottom it is fully shut. The connection to the computer consists of four pulse lines for motor control and two status lines for the gate sensors.

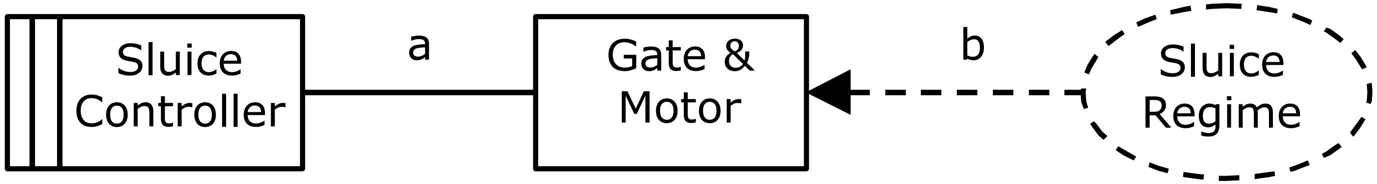
Sluice Gate Control



a: SC!{Clockw,Anti,On,Off}
GM!{Top,Bottom}

b: {Open,Shut}

Sluice Gate Control

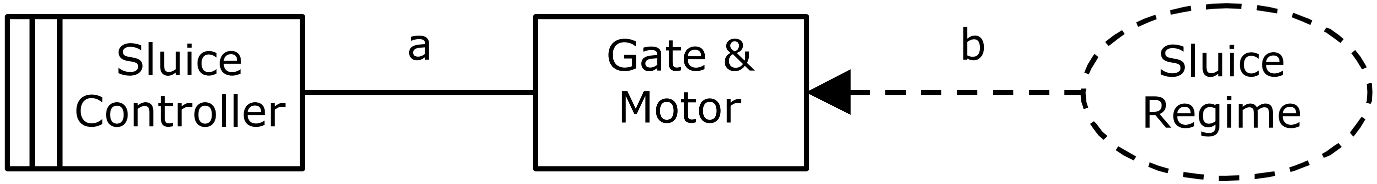


a: SC!{Clockw,Anti,On,Off}
GM!{Top,Bottom}

b: {Open,Shut}

- Interface annotations (a) show which domain controls the phenomena of each class
  - SC controls Clockw,Anti,On and Off events
  - GM controls Top and Bottom states

Sluice Gate Control



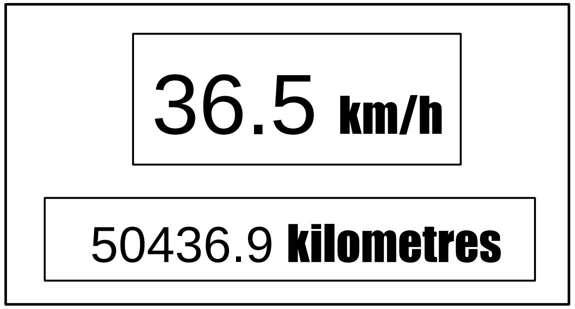
a: SC!{Clockw,Anti,On,Off}
GM!{Top,Bottom}

b: {Open,Shut}

- Reference annotations (b) show which domain phenomena are mentioned in the requirement
  - The customer cares only about whether the sluice gate is Open or Shut
- {Open, Shut} ≢ {Top, Bottom}

Some Tiny Examples — 2

- A microchip computer is required to control a digital electronic speedometer and odometer in a car:

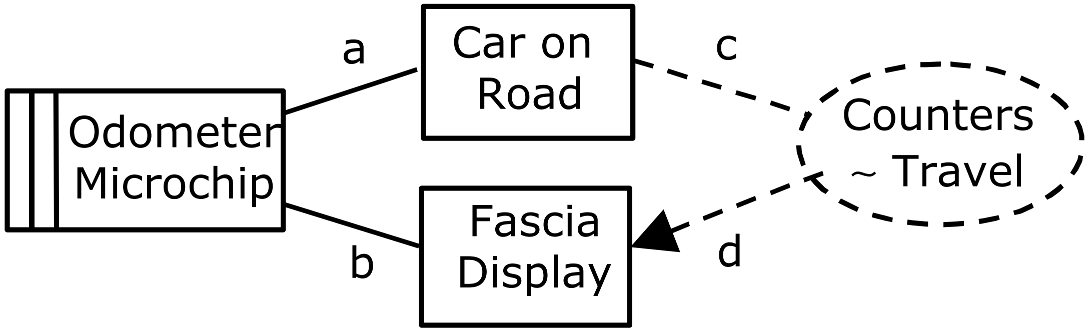


36.5 **km/h**

50436.9 **kilometres**

- One of the car’s rear wheels generates a pulse on each rotation. The computer can detect these pulses and must use them to set the current speed and total number of miles travelled in the two visible counters on the car fascia. The underlying registers of the counters are shared by the computer and the visible display.

Odometer Display



a: CR!{WheelPulse}

b: OM!{Inc/Dec Speed/Dist}

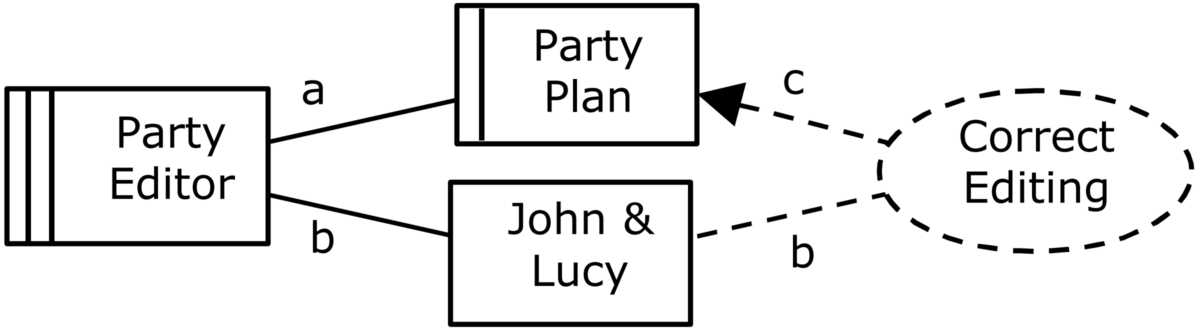
c: {Speed,CumDist}

d:{SpeedCount,DistCount}

Some Tiny Examples — 3

- Lucy and John give many parties, to which they invite many guests. They want a simple editor to maintain the information, which they call their Party Plan.
- The Party Plan is a list of parties, a list of guests, and a note of who’s invited to each party.
- The editor will accept command-line text input, in a (very old-fashioned) DOS or Unix style.
- We are not at all concerned with presenting or printing the information — just with creating and editing it.

Party Plan Editing



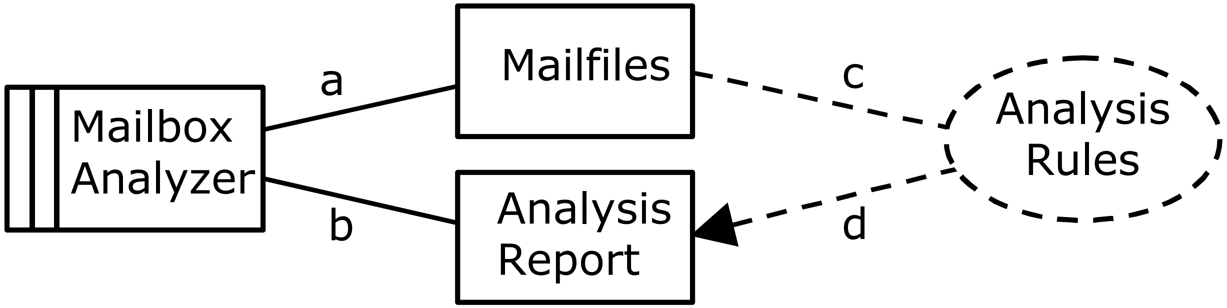
a: PE!{PlanOperations}
PP!{PlanStates}

b: JL!{Commands}
c: {PlanEffects}

Some Tiny Examples — 4

- A PC user wants to analyse the messages in the mail client’s mailbox.
- The required analysis will show statistics on volume, size and frequency of incoming messages, time taken to reply, and so on.
- The messages for each correspondent are concatenated in one file. The files are all in one directory.
- The analysis report has a specified format, with detail lines in order by correspondent name

Mailfiles Analysis



a: MF!{MsgDir,File,Line,Char}

b: MA!{ReportLine,Char}

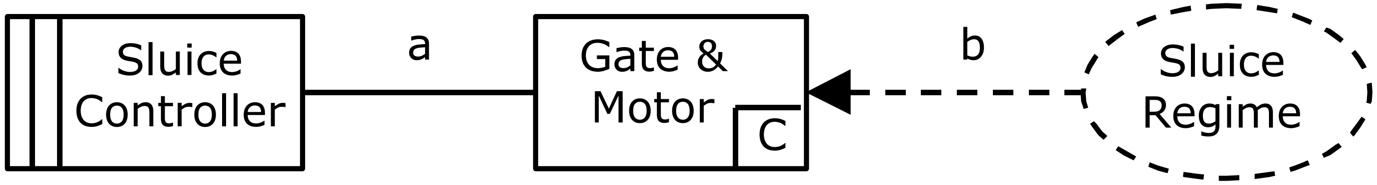
c:{Msg,From,To,Date,Length}

d:{LineData}

Problem Frames and Concerns — 1

- We have seen instances of four *problem frames*:
    - *Simple Behaviour* : Sluice Gate
    - *Simple Information* : Odometer Display
    - *Workpieces* : Party Plan Editing
    - *Transformation* : Mailfiles Analysis
- Each frame characterises a problem class
    - The problem is always to specify a suitable machine
- Frames differ in ...
    - ... types and configuration of domain parts
    - ... frame concern: what’s needed to solve the problem
    - ... descriptions necessary to address frame concern

Sluice Gate Control

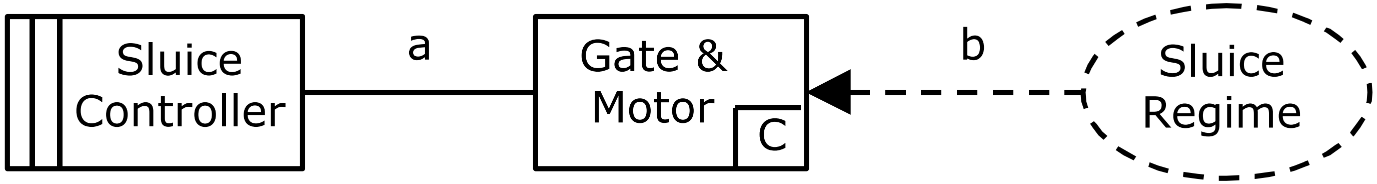


a: SC!{Clockw,Anti,On,Off}
GM!{Top,Bottom}

b: {Open,Shut}

- *Simple Behaviour Problem*
  - The *Controlled Domain* is a Causal domain
    - May be passive or weakly active or strongly active
  - The frame concern:
    - Find and exploit a control law to use phenomena (a) to satisfy the requirement in terms of phenomena (b)
    - Is there such a law?
    - Does *Control Machine* have enough timely information?

Problem Frames and Concerns — 2

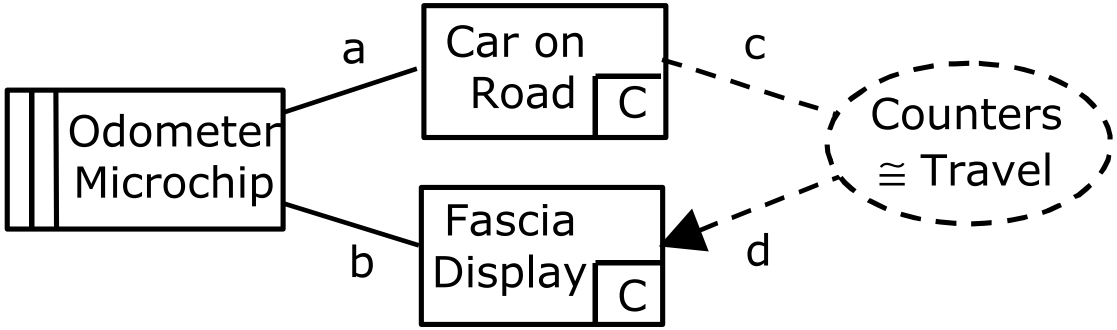


a: SC!{Clockw,Anti,On,Off}
GM!{Top,Bottom}

b: {Open,Shut}

- R: Requirements
  - What the customer *wants* to be true in terms of (b)
- D: Domain Properties
  - What we *know* to be true in the domain
- S: Specification
  - How we *want* the machine to behave at interface (a)
- General solution argument: $S,D \models R$
  - Similar arguments are needed for all frames

Odometer Display

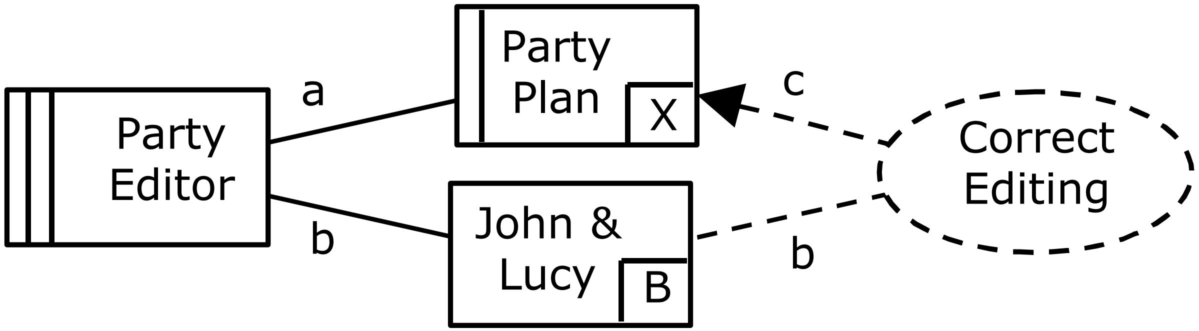


a: CR!{WheelPulse}
b: OM!{Inc/Dec Speed/Dist}

c: {Speed,CumDist}
d:{SpeedCount,DistCount}

- *Simple Information Problem*
  - The *Real World* and *Display Domain* are Causal domains
    - The *Real World* is autonomous
  - The frame concern:
    - Find and exploit domain properties by which the *Information Machine* can infer phenomena (c) from (a)

Party Plan Editing

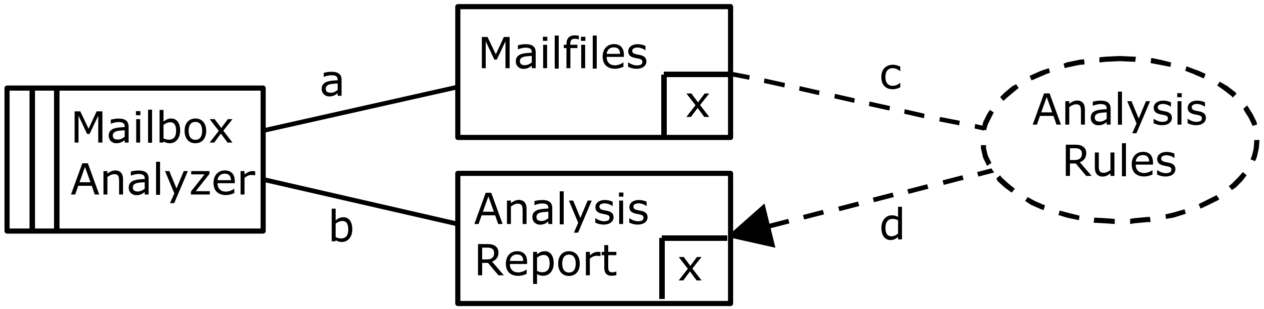


a: PE!{PlanOperations}
PP!{PlanStates}

b: JL!{Commands}
c: {PlanEffects}

- *Workpieces Problem*
  - The *Workpieces* is a Lexical domain
    - Lexical domains are always passive
  - The *Users* are a Biddable domain
  - The frame concern:
    - Generate operations (a) on Workpieces to give
    - correct effects (c) of commands at (b)

Mailfiles Analysis



a: MF!{MsgDir,File,Line,Char}
b: MA!{ReportLine,Char}

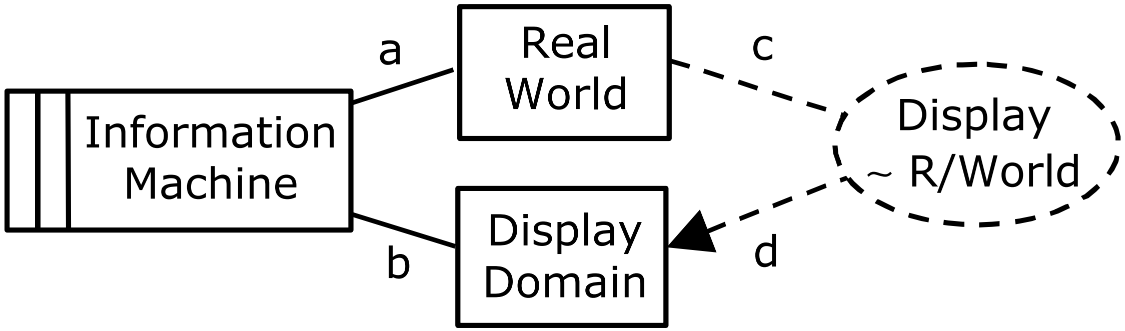
c:{Msg,From,To,Date,Length}
d:{LineData}

- *Transformation Problem*
  - The *Inputs* and *Outputs* are Lexical domains
  - The Requirement is an *Input/Output relation*
  - The frame concern:
    - Find interleaved traversals of *Inputs* and *Outputs* by *Transformation Machine* to achieve *I/O relation*

Problem Frames and Concerns — 3

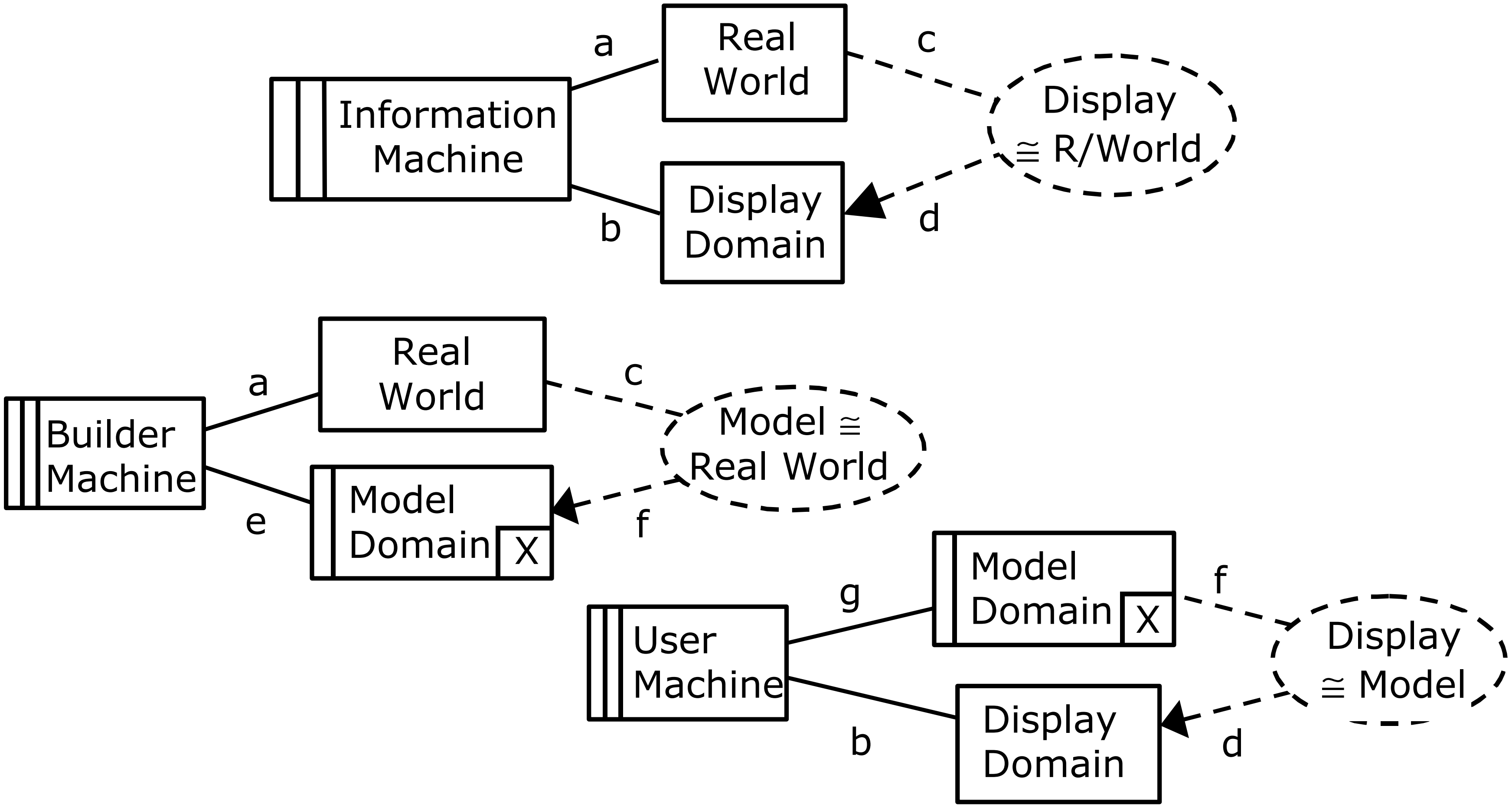
- Each problem frame has its characteristic concern
- Some concerns are common ...
  - ... to several frames, or ...
  - ... to domains of particular kinds
- Some examples:
  - *Untimely* phenomena in an information problem
  - *Reliability* of a causal domain in a behaviour problem
  - *Initial* states of machine and problem domain
  - Potential for *breakage* of a causal domain
  - *Identification* of individuals in multiple domain

Building and Using Model Domains — 1



- Real World phenomena may be *untimely* for display
  - Must be remembered, summarised, extrapolated etc
- Standard decomposition for such information problems
  - (1) Build and maintain a *model* of the *Real World*
  - (2) Use the model as a surrogate in producing display
- Examples of model domains:
  - Database
  - Object model within the machine

Building and Using Model Domains — 2



- We want:
  (Display ≅ Model ∧ Model ≅ World) → Display ≅ World

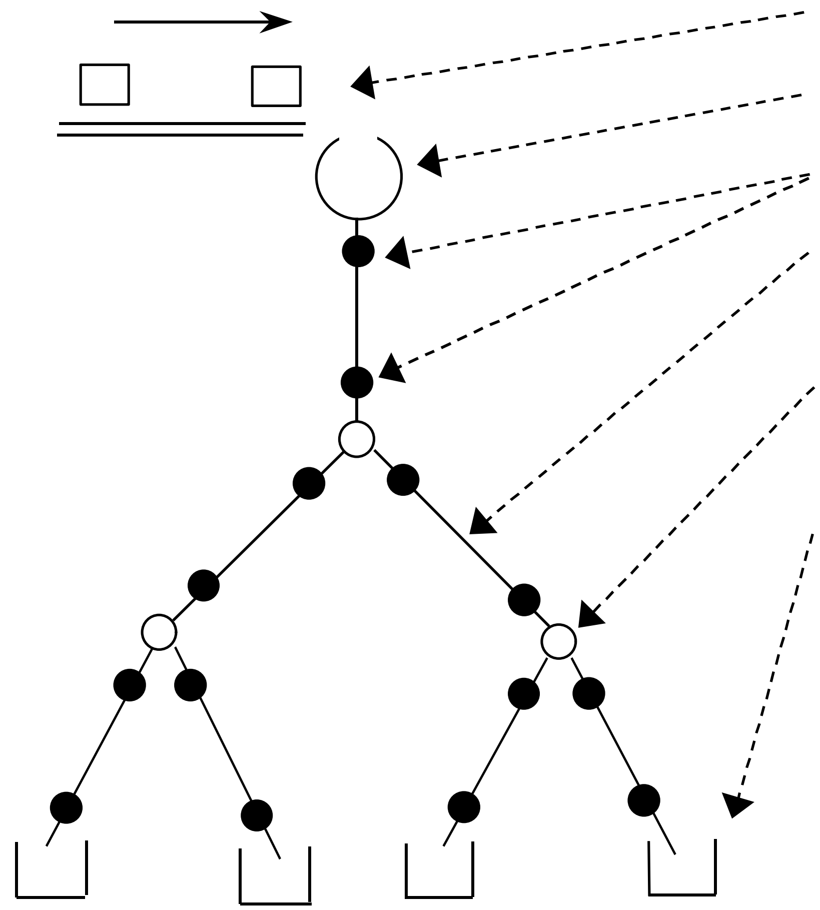
A Larger Problem — 1

- Package Router Control
  - Adapted from:
    William Swartout and Robert Balzer; On the Inevitable Intertwining of Specification and Implementation; Comm ACM 25,7
  - A package router is a large machine used by delivery companies (eg Fedex, Postal Service) to sort packages into bins according to bar-coded destination labels stuck to the packages

Package Router Control

- Packages move on a conveyor to a reading station where their bar-coded package-ids and destinations are read. They then slide by gravity down pipes fitted with sensors at top and bottom. The pipes are connected by two-position switches that the computer can flip (when no package is present in the switch).
- At the leaves are destination bins corresponding to the bar-coded destinations. The system must route packages to their destinations by setting the switches appropriately. Misrouted packages must be reported on the router display. The computer can start and stop the conveyor in response to operator commands.

A Larger Problem — 2



conveyor

reading station

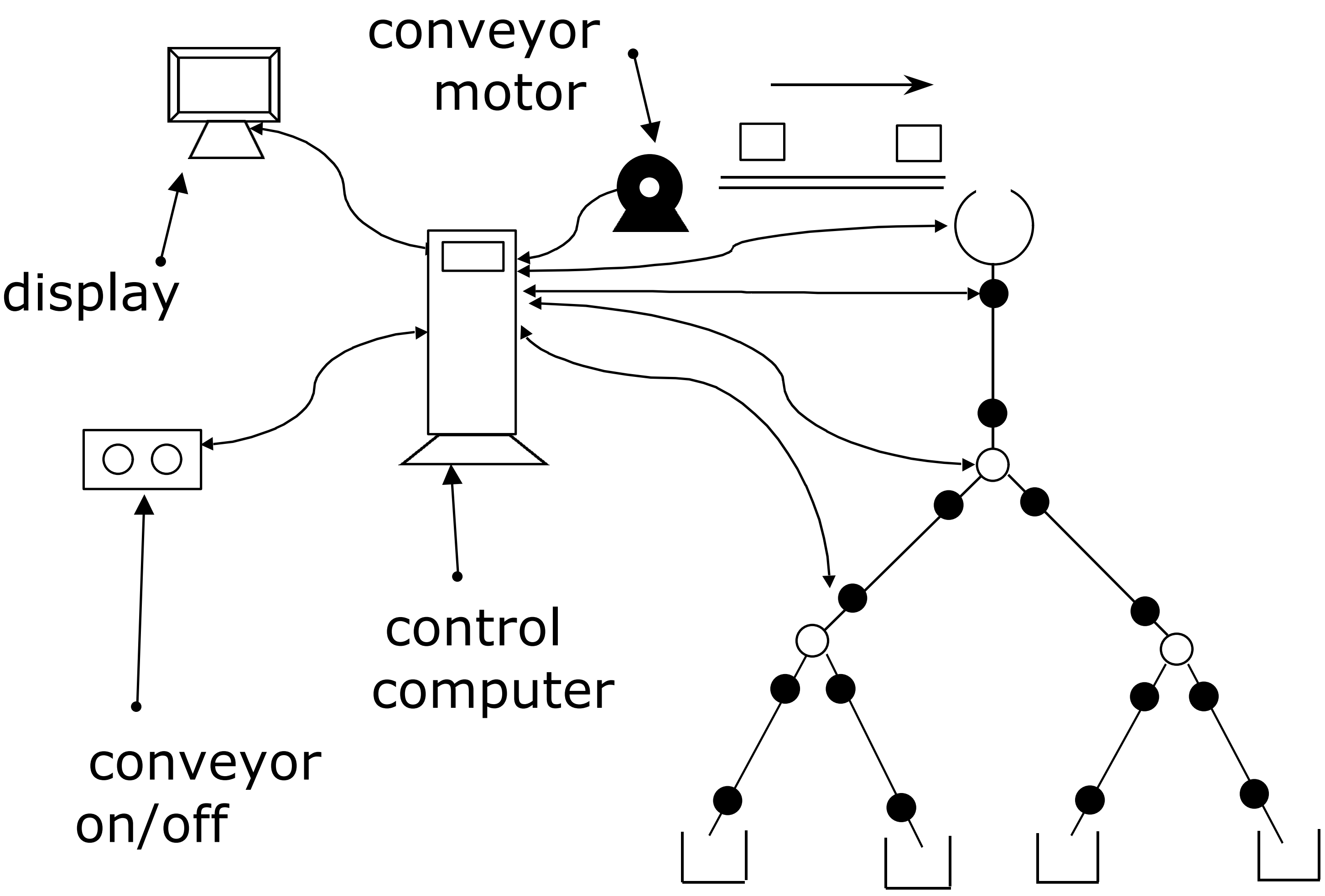
sensors

pipe (no
overtaking)

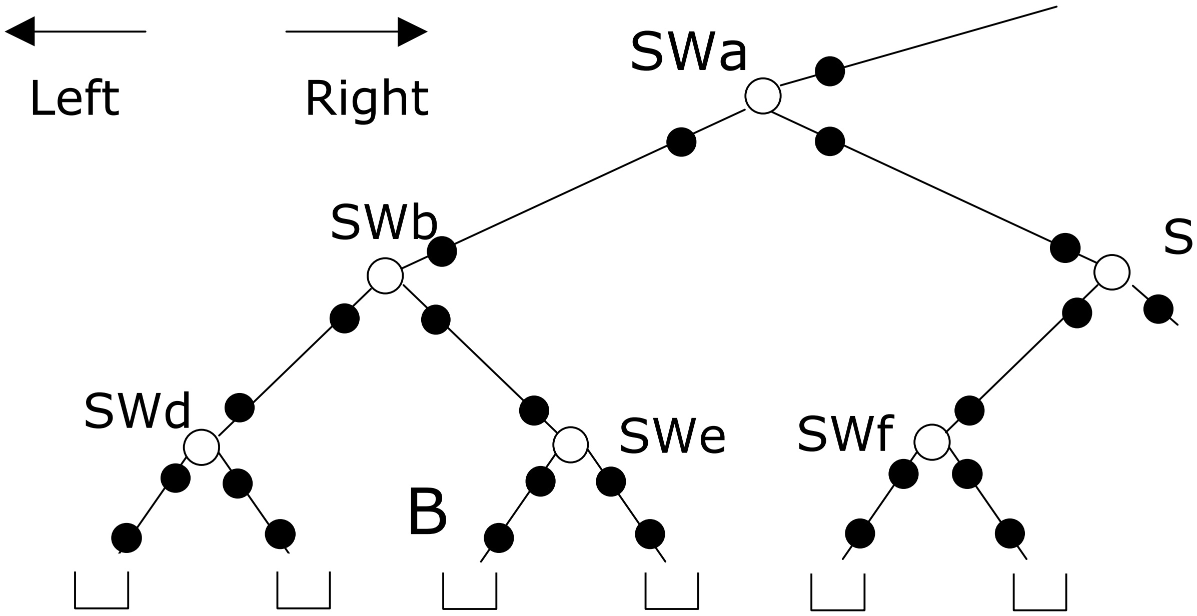
switch (no
overtaking)

bin

A Larger Problem — 3



Routing a Package

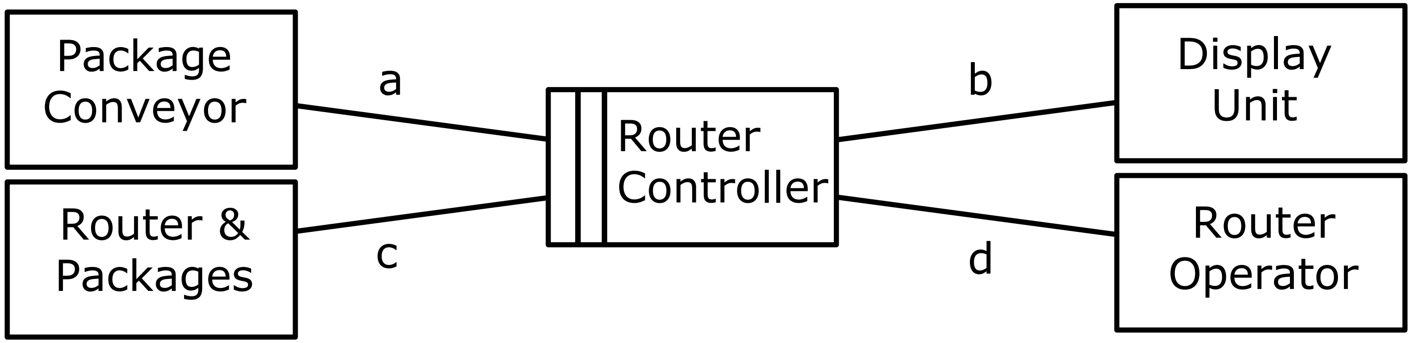


- Route at switch SWa to reach bin B:
  - Left at SWa, Right at SWb, Left at SWe
- Route at switch SWc to reach bin B:
  - No correct choice: package is already misrouted

Problem Decomposition — 1

- We want to decompose into simpler subproblems
  - A simple problem is one we recognise ...
    ... as matching a known frame
- Sometimes a frame concern itself leads to a further decomposition
- Decomposition is not, in general, hierarchical
- Decomposition leads to composition concerns
  - We defer the composition concerns ...
  - ... until the components are understood
- Decomposition starts with a context diagram
  - Machine and problem domain, no requirement
  - Each subproblem has its own requirement

Context Diagram



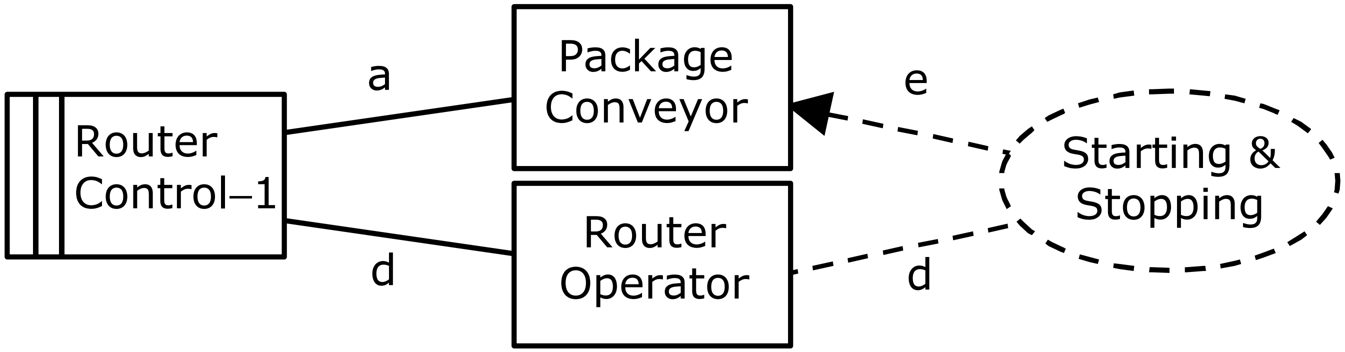
a: RC! {OnC, OffC}
b: RC! {ShowPkgId, ShowBin, ShowDestn}
c: RC! {LSw(i), RSw(i)}
RP! {SendLabel(p,l), LId(l,i), LDest(l,d), SwPos(i), SensOn(i)}
d: RO! {OnBut, OffBut}

- A context diagram shows:
  - The machine and the problem domain parts ...
  - ... but no requirement

Problem Decomposition — 2

- Three Basic Subproblems
  - Conveyor control
  - Routing packages
  - Reporting misrouted packages
- Simple parallel structure:
  - Requirements are conjoined

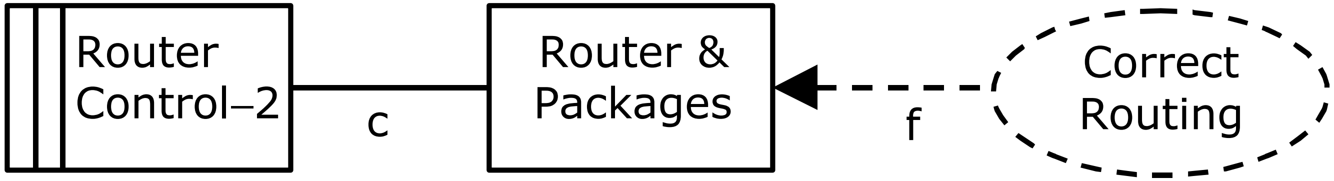
Conveyor Control: Commanded Behaviour



a: RC! {OnC, OffC}
d: RO! {OnBut, OffBut}
e: {Running, Stopped}

- Requirement: start and stop conveyor as commanded
- Requirement phenomena
  - Domain properties relate {Running,Stopped} to {OnC, OffC}

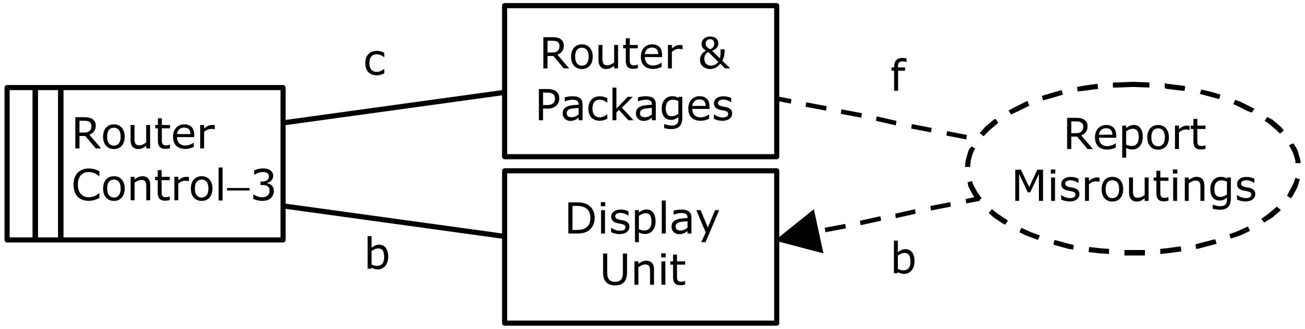
Routing Packages: Required Behaviour



c: RC! {LSw(i), RSw(i)}
RP! {SendLabel(p,l), LId(l,i), LDest(l,d), SwPos(i), SensOn(i)}
f: {PkgArr(p,b), Assoc(d,b), PDest(p,d)}

- Requirement: each package arrives at its destination bin
- Defined requirement phenomena:
  - PDest(p,d) ≜ label of package p shows destination d
- Frame concern
  - Can PkgArr be simply controlled using only the phenomena (c)?
  - Is information at (c) available when needed?

Reporting Misroutings: Information Display



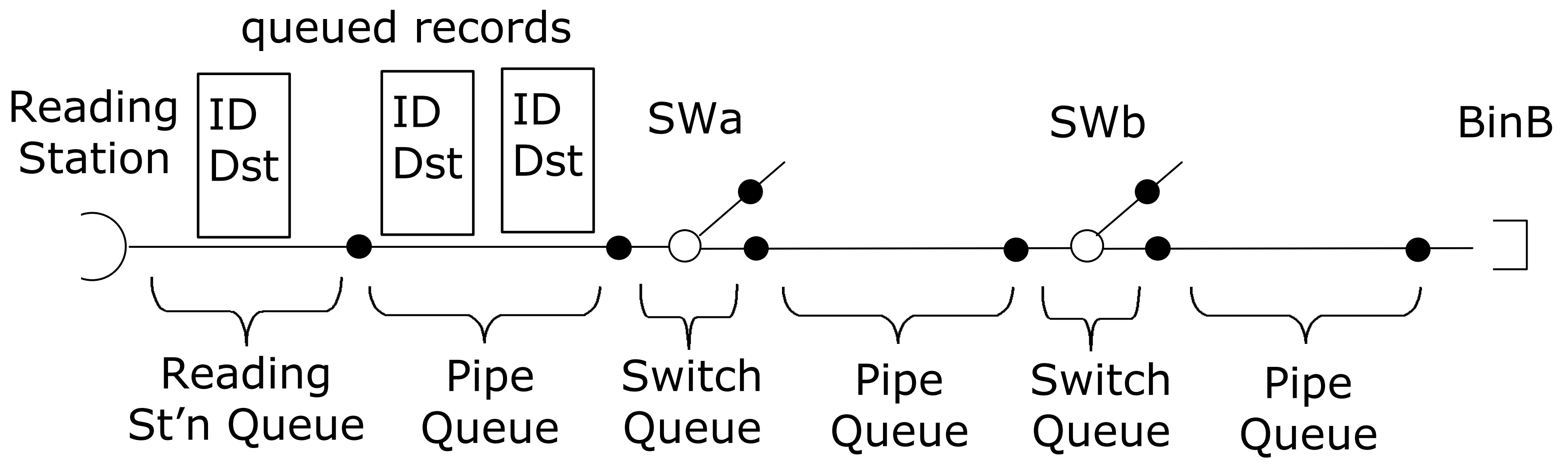
c: RP! {SendLabel(p,l), LId(l,i), LDest(l,d), SensOn(i)}
f: {PkgArr(p,b), Assoc(d,b), PDest(p,d)}
b: RC! {ShowPkgId, ShowBin, ShowDestn}

- Frame concern
  - Can phenomena (f) be inferred from phenomena (c)?
- The information that is needed:
  - Id and Dest of package on arrival at a bin
  - Association of destinations with bins

Model Domains & a Description

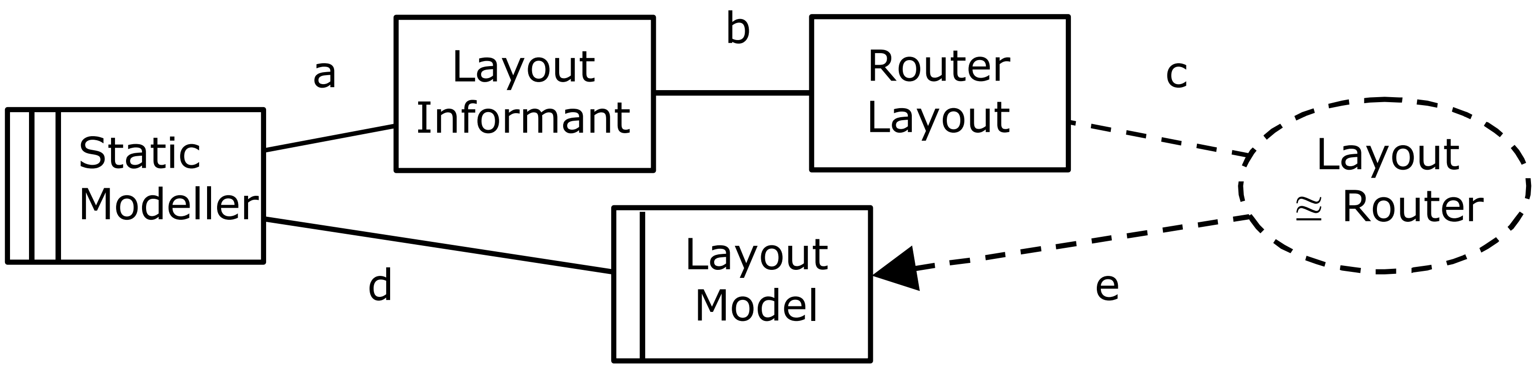
- Dynamic model
    - What is the Id and Destination of the package most recently arrived at sensor s?
- Static model
    - Where in router is sensor s? Switch w?
    - Which way from switch w to bin b?
- Associating destinations with bins
    - Which bin for destination d?
- ID model
    - Which sensor is attached to port p?

A Queues Model of Router Packages



- No overtaking, so packages form queues
- Each queue associated with its entry point {sensor}∪RS
  - Reading Station queue
  - Non-empty switch queue has only one active exit
- Queued elements are records (ID, Destination)

Building a Static Model



a: LI! {Data Entry Cmds}

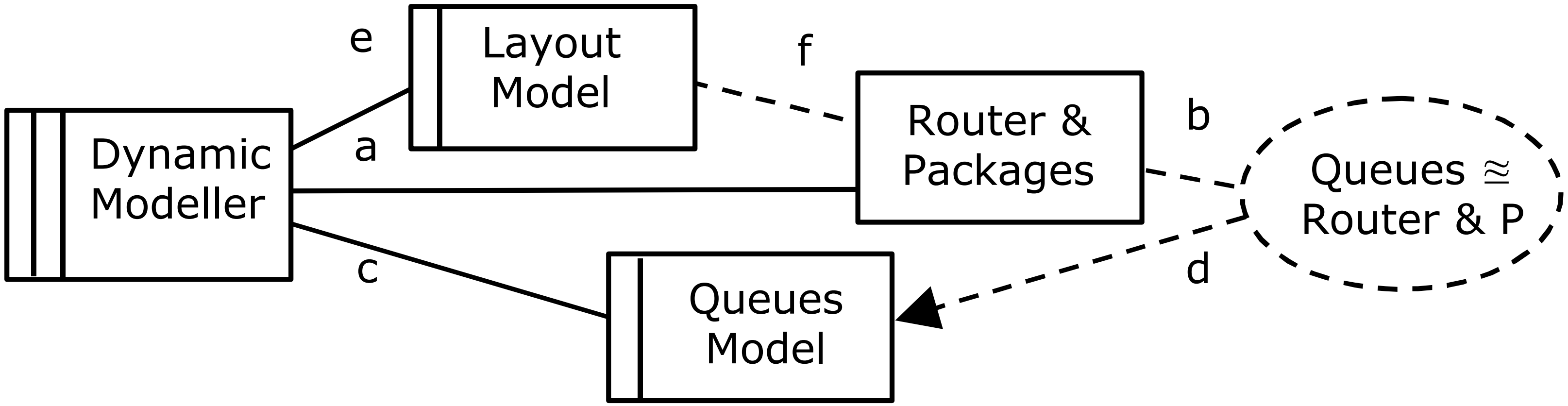
b: RL! {Visible Layout}

c: {Router Layout States}

d: SM! {Layout Model Operations}

e: {Layout Model Relations}

Building a Dynamic Model



a: RP! {SendLabel(p,l), LId(l,i), LDest(l,d), SensOn(i)}

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

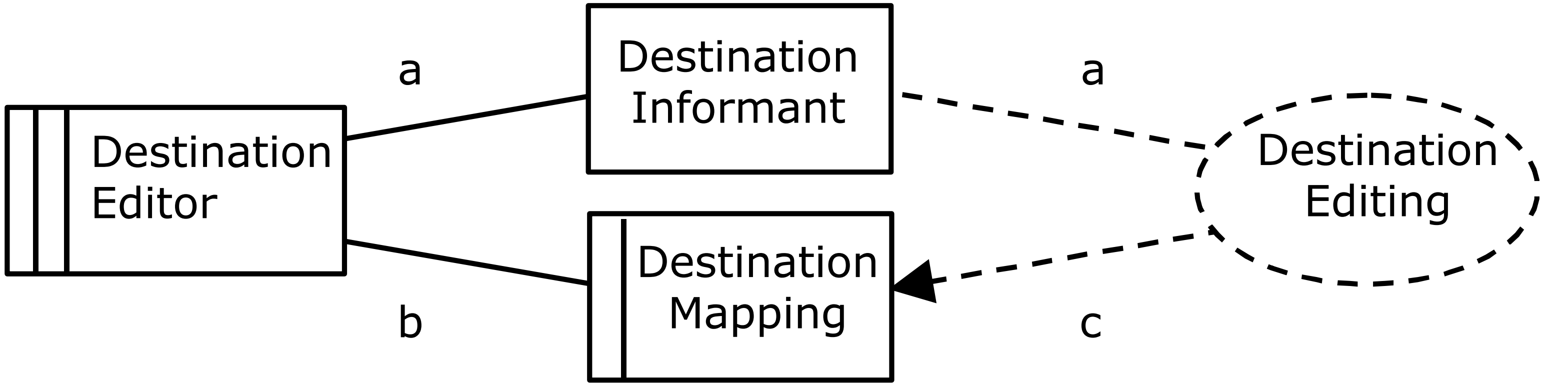
c: DM! {EnQ(r,q,s), DeQ(r,q,s), RecPkg(r,p,d)}

d: {LastArr(s,q,p,d), Empty(s,q)}

e: LM! {Layout Model Relations}

f: RP! {Router Layout States}

Associating Destinations with Bins



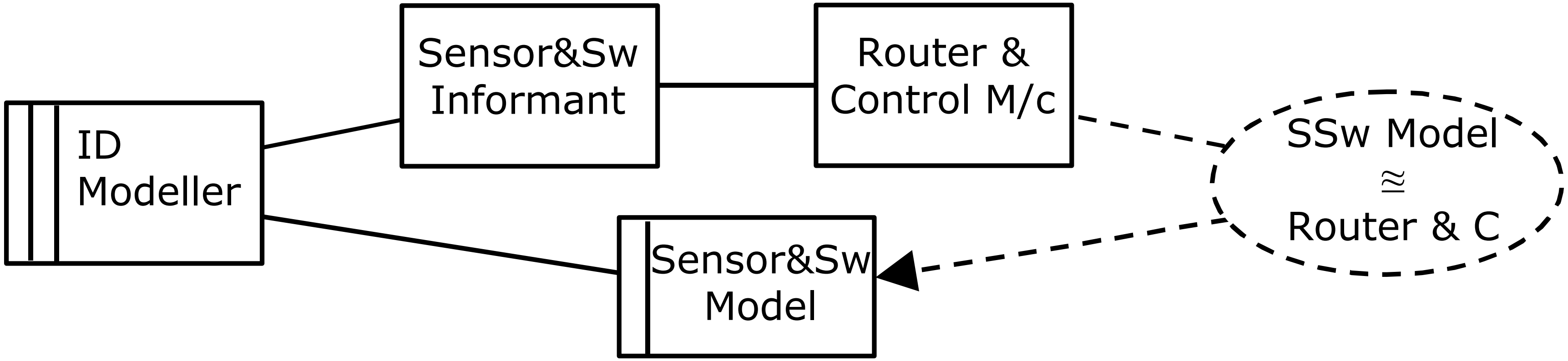
a: DI! {Edit Commands}

b: DE! {Mapping Operations}

c: {Destination Mapping elements & relations}

- Which bin associated with destination (string) d?
- Not a model-building problem: no Real World

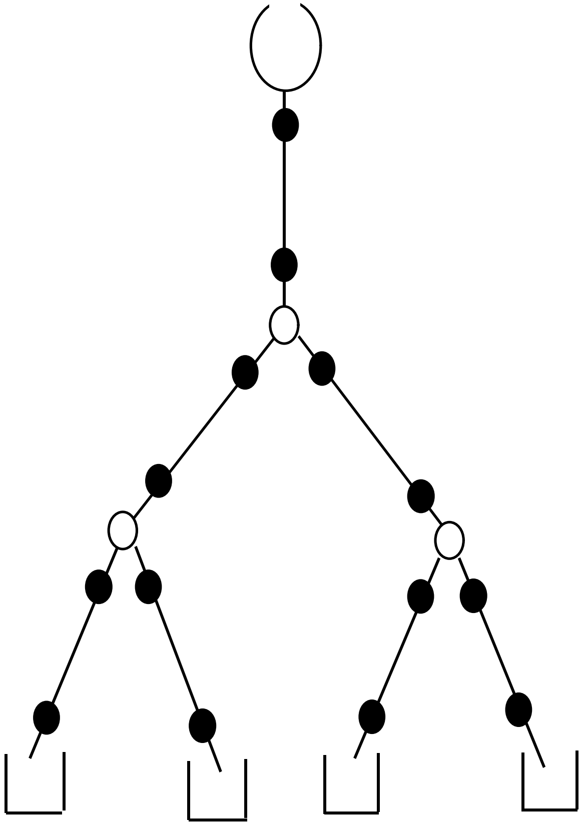
Making an ID Model for Sensors & Switches



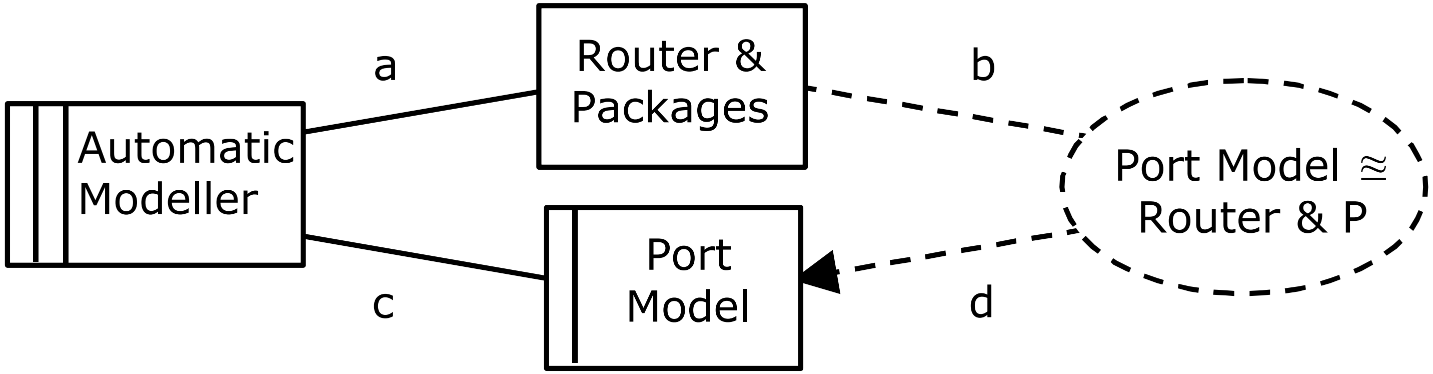
- Which port flips Swa? Which sensor at port p1?
- Sensor & Switch Model maps M/c port ↦ sensor &c
- Human informant
  - Can see Router & Control M/c
  - Enters data for mapping M/c port ↦ sensor &c

Automating ID & Static Modelling

- Build combined Id and Static Model automatically
  - Find portp ↦ switchw, switchw ↦ portq by LSw(p), RSw(p), SWPos(p)
  - Set all ports left, run 1 package:
    - <pi1, pi2, pi3, …> are sensors on left edge
  - Find any switch on left edge by running more packages, …
- What is the smallest number of packages to build a model of balanced tree with 2^n-1 switches?



Automating ID & Static Modelling



a: RP! {SendLabel(p,l), SensOn(i), SwPos(i)}
AM! {LSw(i), RSw(i)}

b: {RouterLayout}

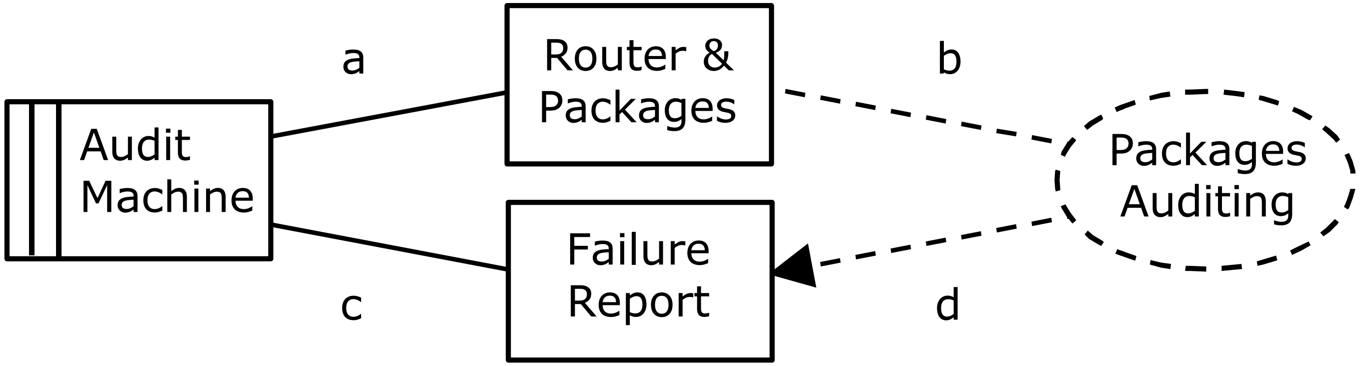
c: AM! {Port Model Operations}

d: {Port Model elements & relations}

Reliablity Concern

- What happens if a package is jammed?
  - The machine must stop the conveyor
- A standard *auditing* subproblem
  - Information problem

Auditing Package Misbehavour



a: RP! {SendLabel(p,l), SensOn(i), SwPos(i)}

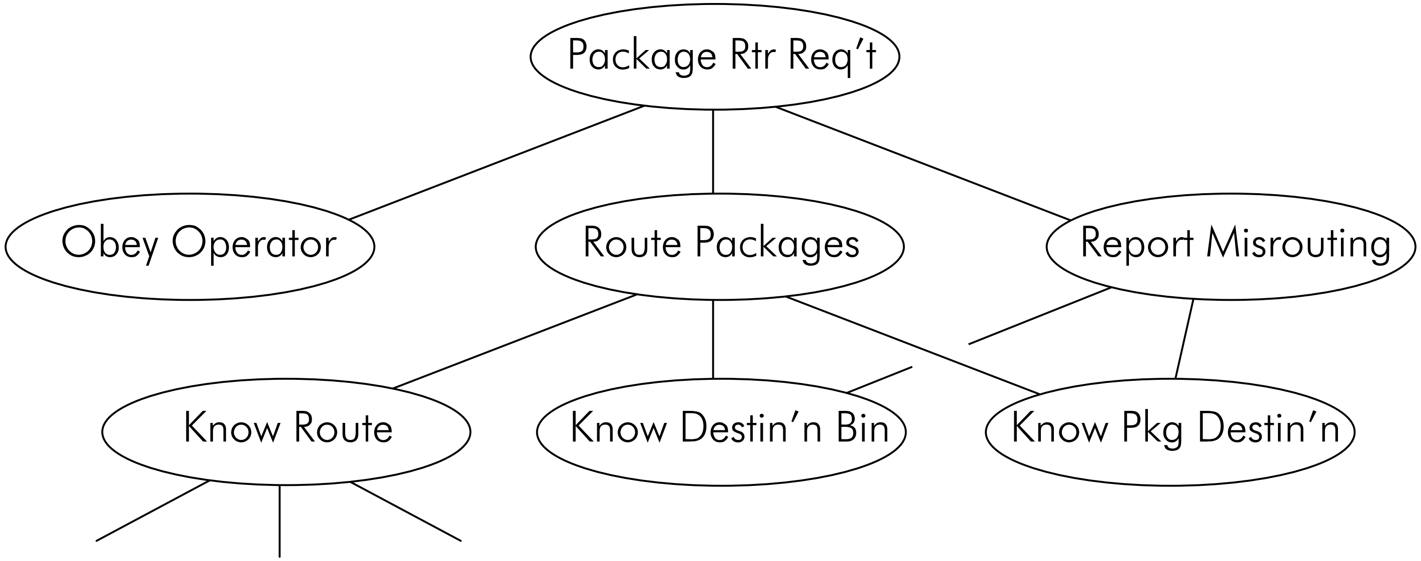
b: {Package Misbehaviour}

c: AM! {Failure Report Operations}

d: {Failure Report Information}

- Package misbehaviour: jam in switch, overtaking, …
- We must arrange that failure causes conveyor to stop
  - Some composition with Conveyor Control

Package Router: Overview



- Goal hierarchy (KAOS-like)
  - Conveyor, Route Packages, Report Misrouting
- Concerns (cf KAOS obstacles, conflicts)
  - Frame concerns, identities, reliability, …

A van Lamsweerde and E Letier; *Integrating Obstacles in Goal-Directed Requirements Engineering*; Proc ICSE-20, ACM-IEEE, April 1998

Decomposition Concerns

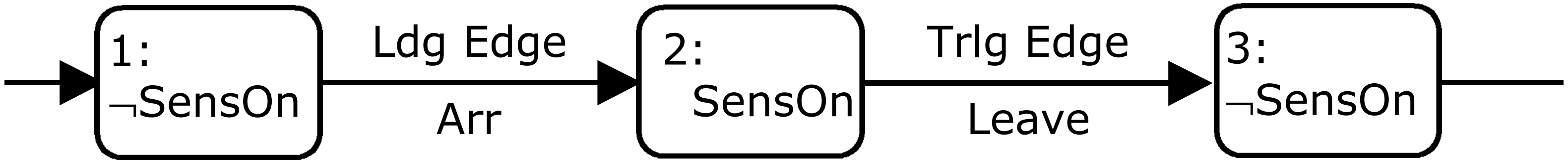
- Subproblem decomposition separates concerns of two categories
  - Concerns of the individual subproblems
  - Composition concerns of the subproblem set
- The trade-off
  - Making the individual subproblems simpler
  - Making the composition simpler
- ‘Traditional’ decomposition
  - Makes composition trivial (eg by procedure call)
  - Makes subproblems more complex
    - Composition concerns are distributed
  - Forces premature consideration of composition concerns

Interference and Synchronisation

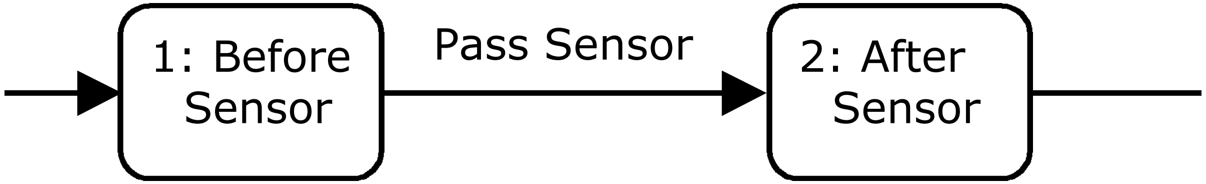
- Interference
  - Model is a shared variable for Build and Use machines
  - Use machine assumes lexical domain properties ...
  - ... so mutual exclusion is required in composition
- Interference examples
  - In Library: membership update vs. book-borrow
  - In Package Router: queue model update vs routing
- Synchronisation
  - An additional constraint beyond mutual exclusion
- Synchronisation examples
  - In Library: allow 2-week loan in last membership week?
  - In Package Router: when can Dest/Bin mapping change?

Commensurable 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?

Heuristics for Decomposition

- Recognising instances of problem frames
- Decomposition to handle concern
  - Introducing a model, an ID model, an audit problem
- More than one tempo
  - Library membership and borrowing
- More than two moods
  - ATC
    - planes and runway, minimum separation, correction procedure, emergency procedure
- Unexpected phenomena intruding into model
  - Of Operator, of User, ...
- ...

Summary

- Problem Frames characterise problem classes
  - Each frame has a machine and ≧ 1 problem domain
- The problem is not in the machine
  - Customer’s requirement is on problem domain
- Problem analysis is about the problem domain
  - This implies an emphasis on domain phenomena
- Problem decomposition is to recognised subproblems ...
  - ... for which we have known problem frames ...
  - ... and to recognised solutions of difficulties
- Problem decomposition defers composition concerns ...
  - ... until the subproblems are understood
- Problem frames can be used in any process setting
  - No particular development sequence is assumed