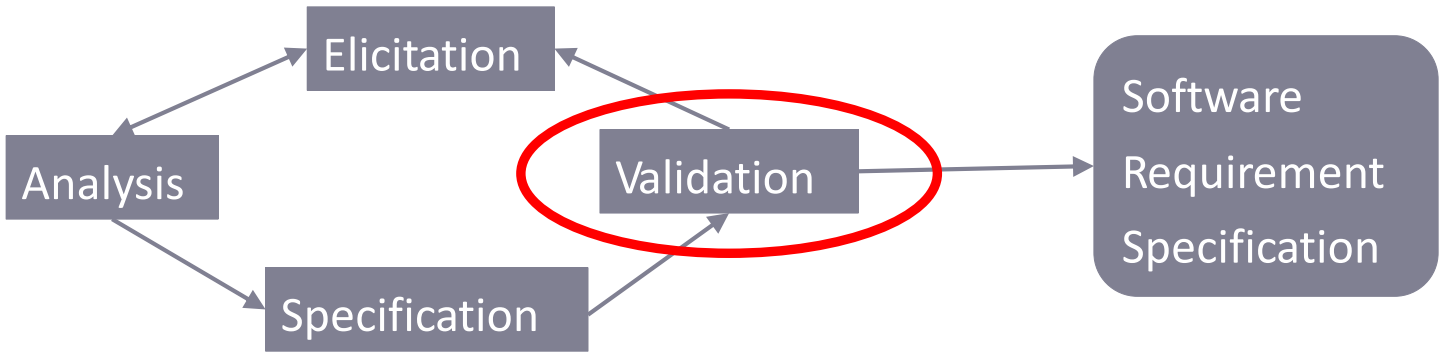
Software Engineering
Lecture 6: Requirements Validation

Stefan Hallerstede, Carl Schultz, Peter Gorm Larsen

Århus University Department of Engineering

3 October 2018

PROCESS FOR CAPTURING REQUIREMENTS



- **Elicitation**: Collecting the user's requirements
- **Analysis**: Understanding and modelling of desired behaviour
- **Specification**: Documenting the behaviour of the proposed software system
- **Validation**: Checking that the specification matches the user's requirements

1

Contents

Contents

Review and Inspection

- **Reviews and inspections**
  Analyse artefacts such as requirements documents, design diagrams and program source code
- **Automated software assessment**
  Run software that computes metrics of such artefacts

(Informal) Reviews

Why make informal reviews?

- *Everyone* makes mistakes
- Create open atmosphere (increase productivity)
- Programming is a *social activity* (or should be)
- Find errors in program *early* (before it is run the first time)
- Find *quality issues*
- Improve *programming skills* of all involved
- Anything can be reviewed
  (. . . , use cases, documentation, . . .)

How To Hold A Review Meeting?

- **Purpose:** to *evaluate* a software product to
  - determine its *suitability* for its intended use
  - identify *discrepancies* from specifications and standards
- Participants read documents in advance
  - then bring their *comments* to a meeting for discussion
- A review
  - may provide *recommendations* and suggest alternatives
  - may be held at any time during a project
  - need *not reach conclusions* on all points

How To Hold A Review Meeting?

- **Purpose:** to *evaluate* a software product to
  - determine its *suitability* for its intended use
  - identify *discrepancies* from specifications and standards
- Participants read documents in advance
  - then bring their *comments* to a meeting for discussion
- A review
  - may provide *recommendations* and suggest alternatives
  - may be held at any time during a project
  - need *not reach conclusions* on all points

- What should *not* happen in a review?
  - *Improvements* to the program
  - *Blaming* programmers
  - *Finger pointing*

(Formal) Inspections

- Idea behind inspection: Michael Fagan (IBM, 1976)
- **Purpose:** detect and identify software product anomalies by *systematic peer evaluation*
- The inspection leader is *not the author*
  - is a *trained* moderator
  - organizes the *selection of inspectors*
  - distributes the documents
  - *leads* the meeting
  - ensures all *follow up actions* are taken

How To Inspect?

- Set an *agenda* and maintain it
- *Limit* debate and rebuttal
- Do *not* attempt to solve every problem noted
- Take *written notes*
- Insist on *advance preparation*
- Conduct meaningful *training* for all participants
- Inspect your *earlier inspections*

Contents

Modelling With Problem Frames[1]

Problem Frames:

- Focus on the problem solving aspect of requirements modelling
- Do not commit early to solutions: do not implement before you understand the problem
- Abstract from state and behaviour

[1] M. Jackson (2001) *Problem Frames – Analysing and structuring software development problems*. Addison-Wesley

Contents

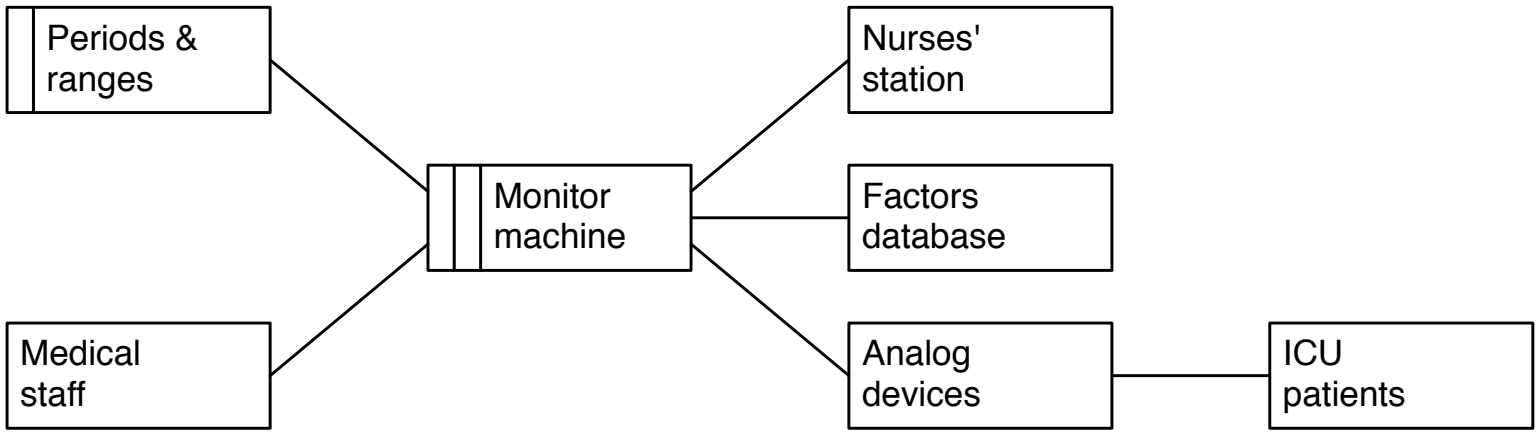
Patient Monitoring Problem

A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

Patient Monitoring Problem

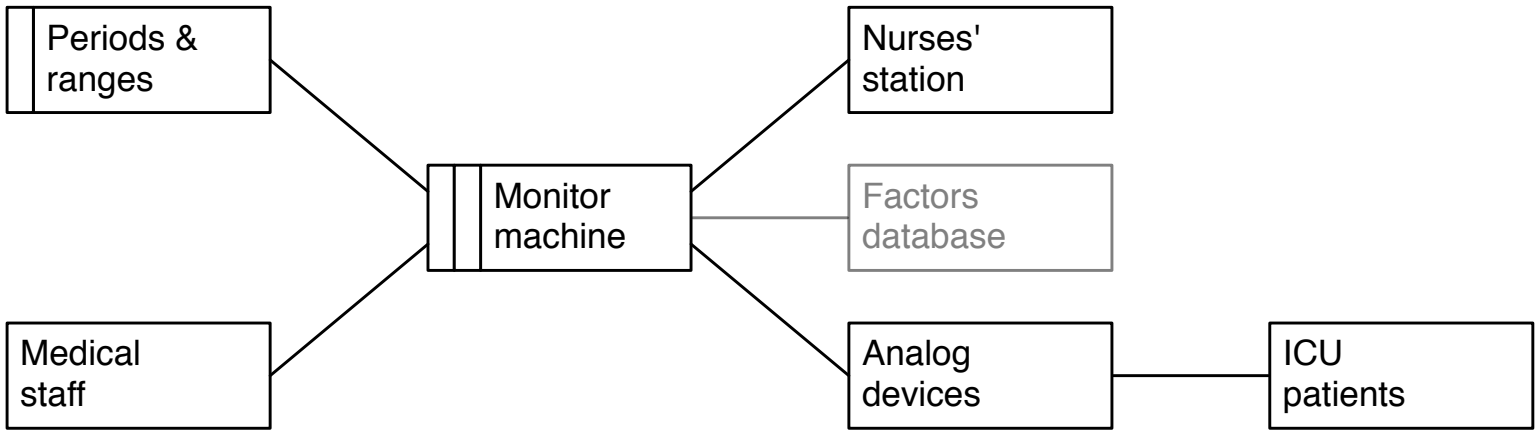
A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

Patient Monitoring Problem



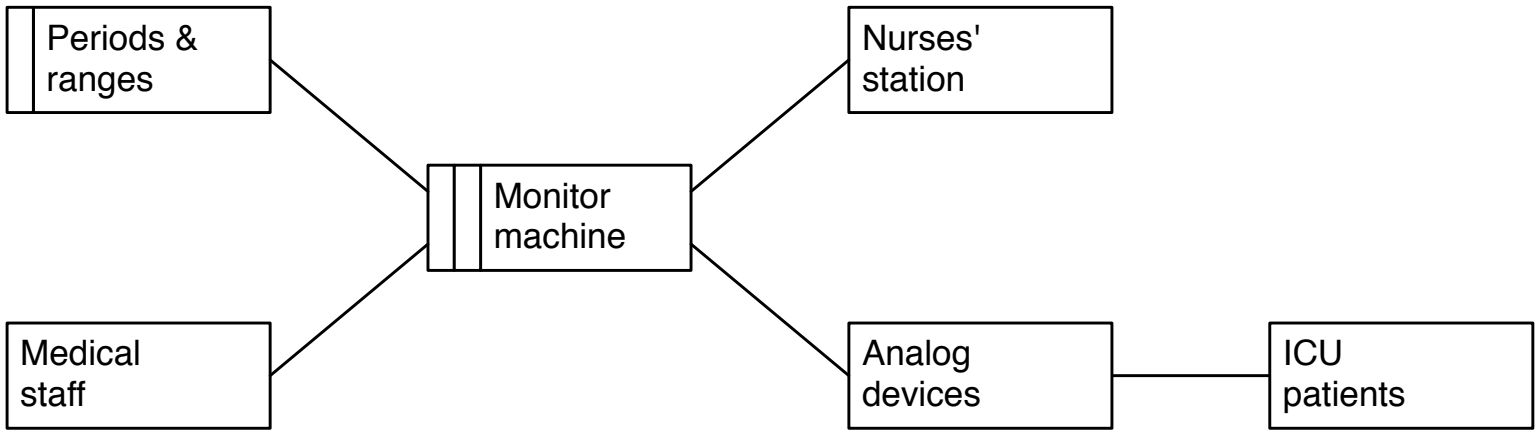
A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

Patient Monitoring Problem



A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

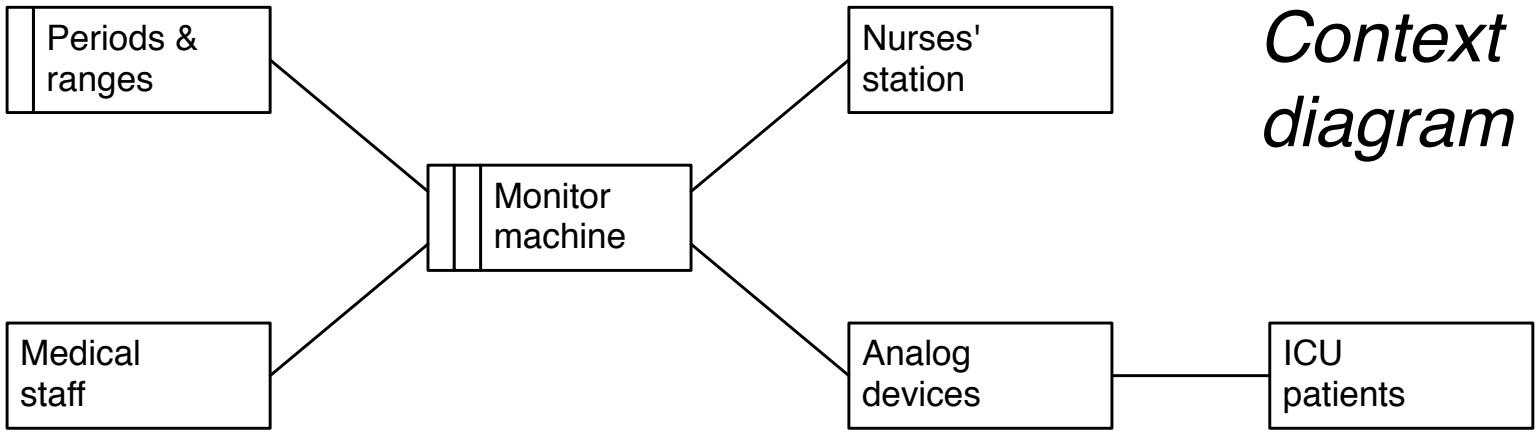
Patient Monitoring Problem



A patient monitoring program is required for the ICU in a hospital. Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure and skin resistance. The program reads these factors on a periodic basis (specified for each patient) and stores the factors in a database. For each patient, safe ranges for each factor are also specified by medical staff. If a factor falls outside a patient's safe range, or if an analog device fails, the nurses' station is notified.

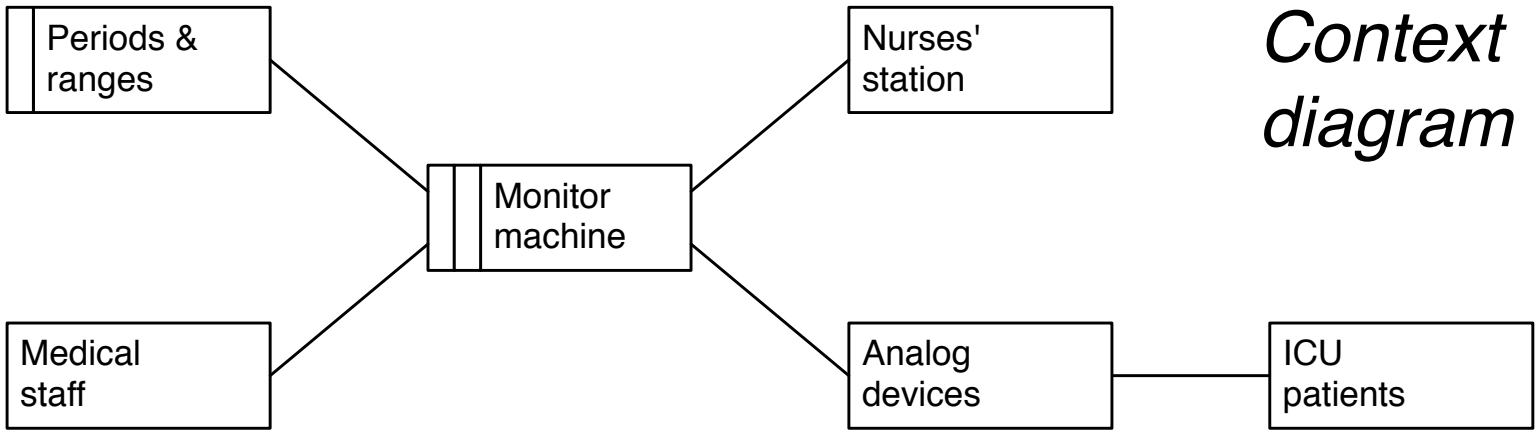
Contents

Patient Monitoring Problem



Context diagram

Patient Monitoring Problem

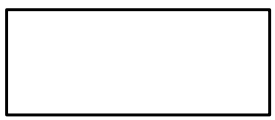


Context diagram

Problem domains:

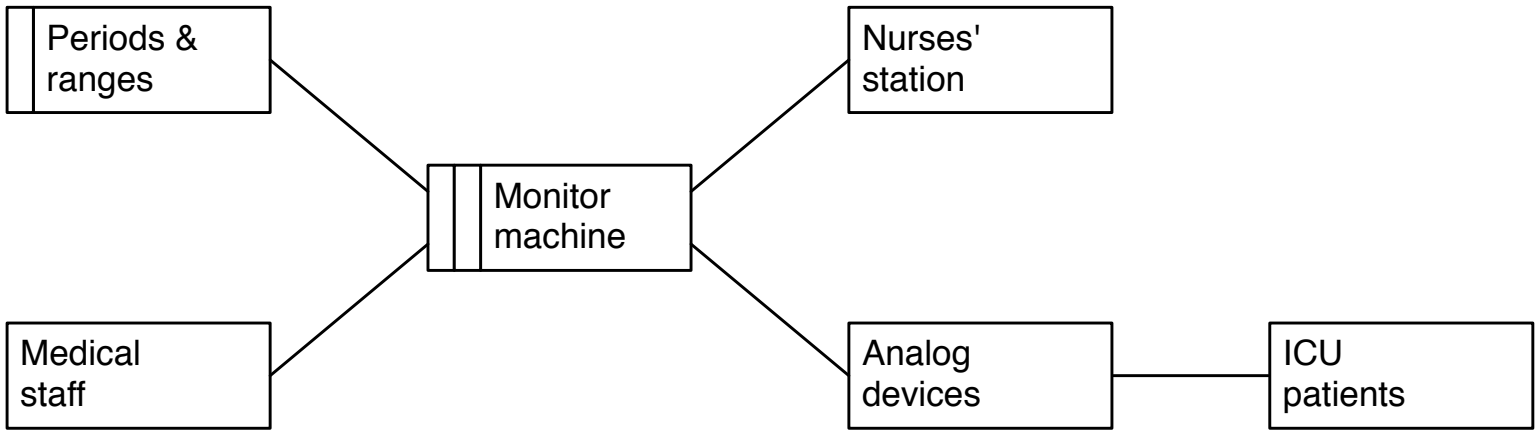
the *machine domain* is the computer for which the software is to be developed

a *designed domain* is the physical representation of some information, e.g., on a hard disk or screen



a *given domain* is a problem domain whose properties are given. The domain cannot be designed.

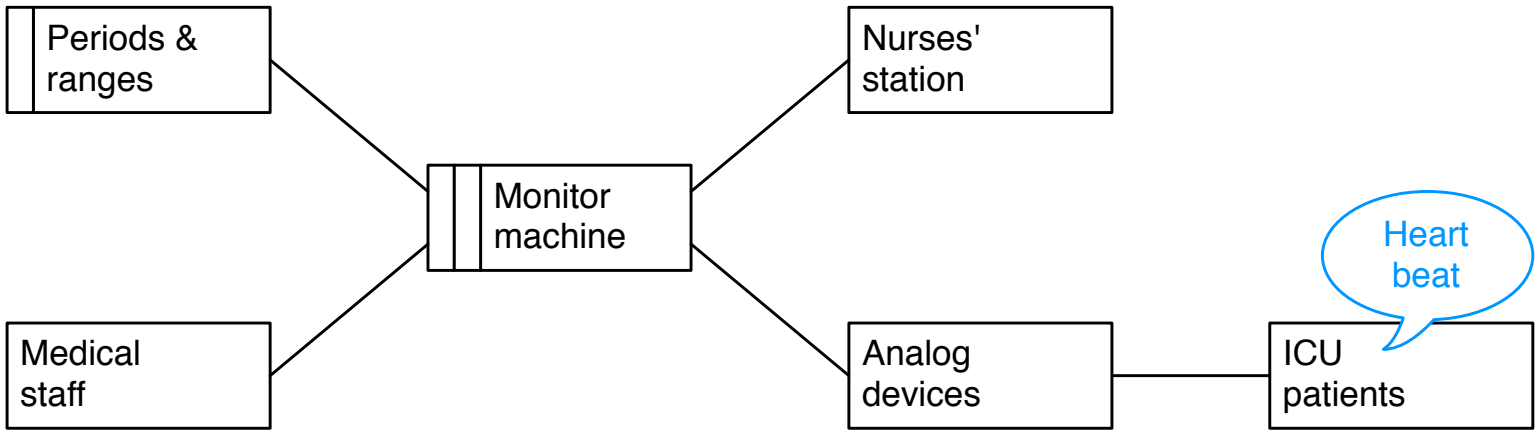
Patient Monitoring Problem



We can ask questions about the model. For instance,

Do we really need the ICU patients domain?

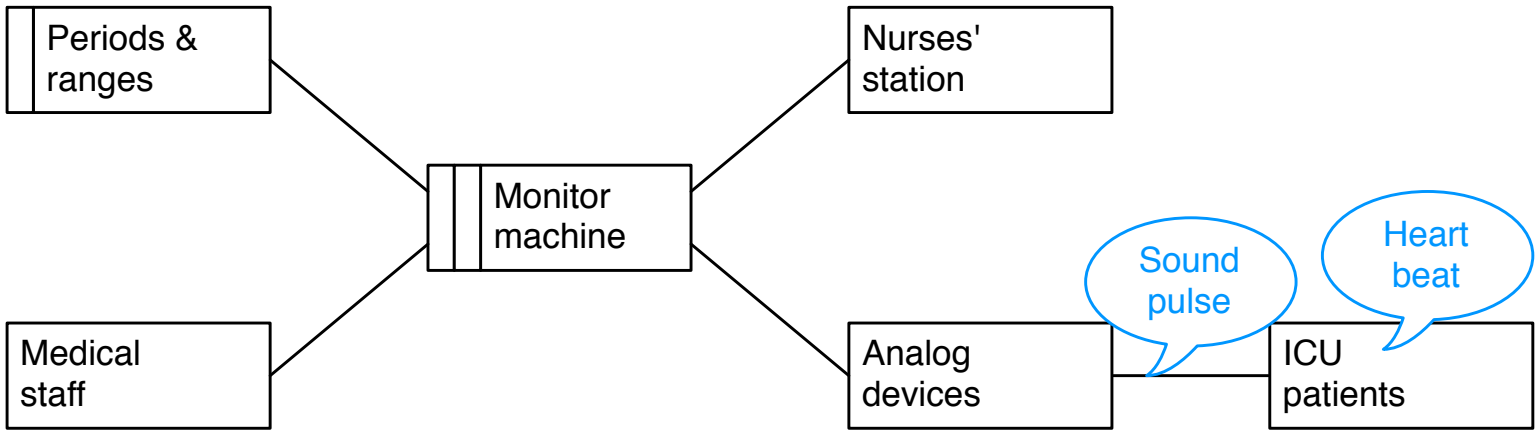
Patient Monitoring Problem



We can ask questions about the model. For instance,

Do we really need the ICU patients domain?

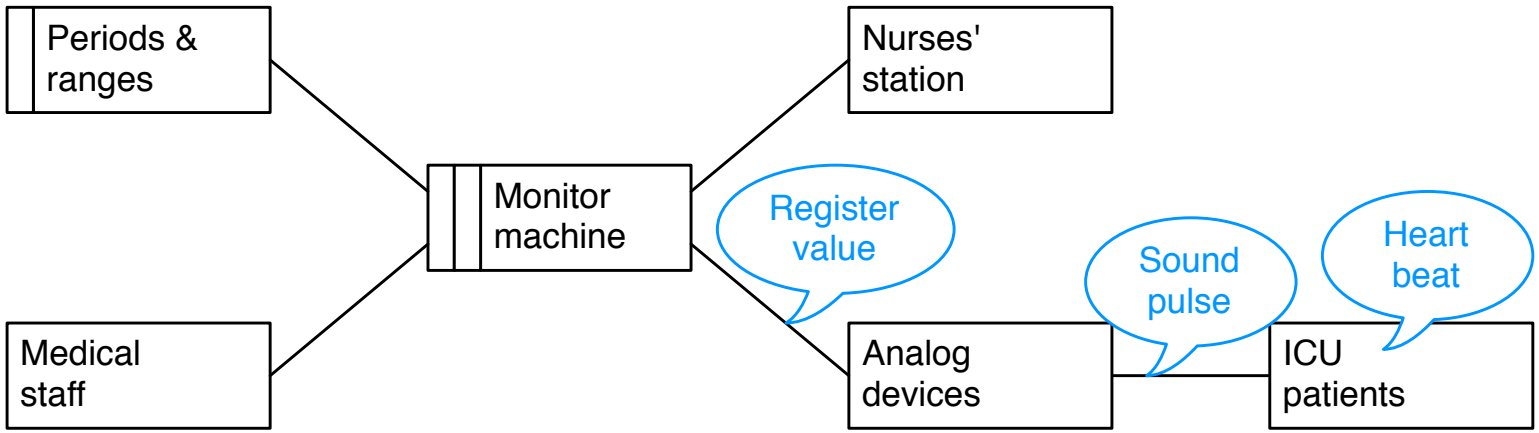
Patient Monitoring Problem



We can ask questions about the model. For instance,

Do we really need the ICU patients domain?

Patient Monitoring Problem



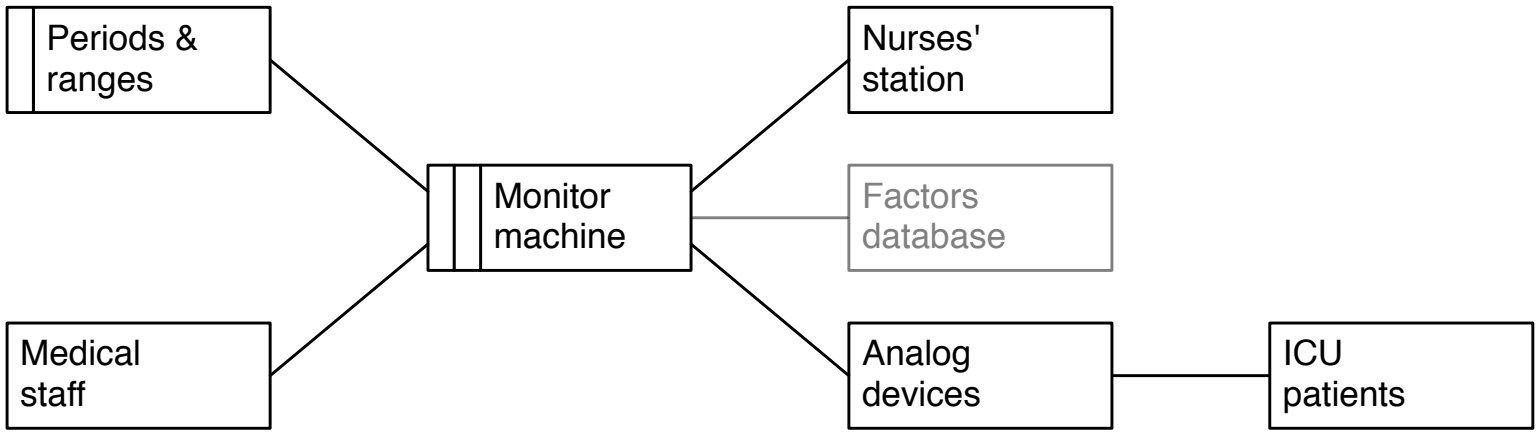
We can ask questions about the model. For instance,

Do we really need the ICU patients domain?

We can answer the question.

Yes, the patients are at the heart of the problem.

Patient Monitoring Problem

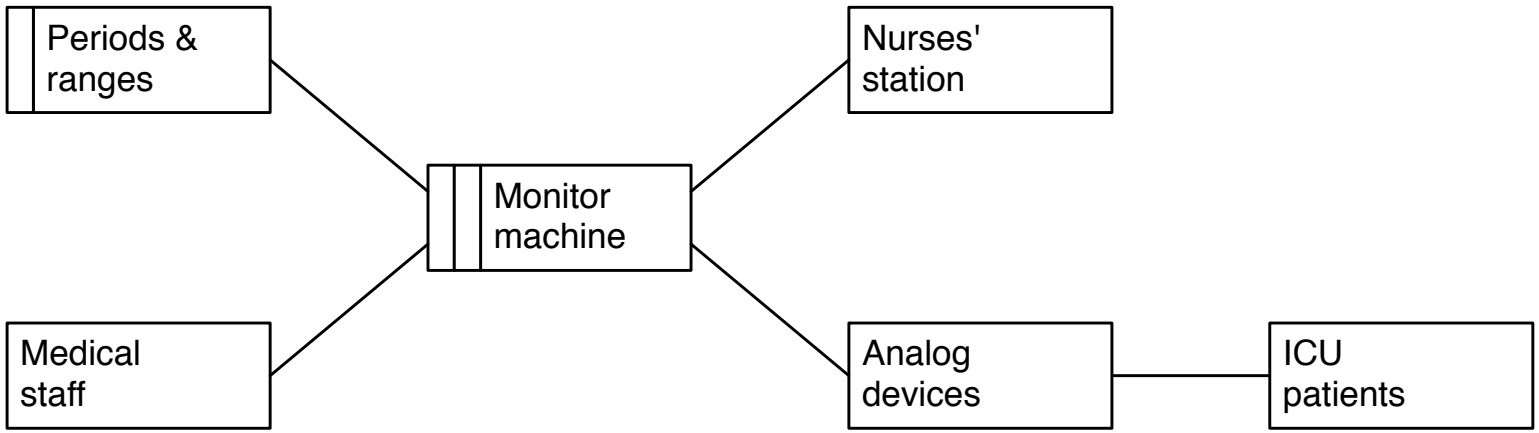


We can ask questions about the model. For instance,

Do we really need the ICU patients domain?

Earlier we asked a similar question about the "*factors database*".

Patient Monitoring Problem



We can ask questions about the model. For instance,

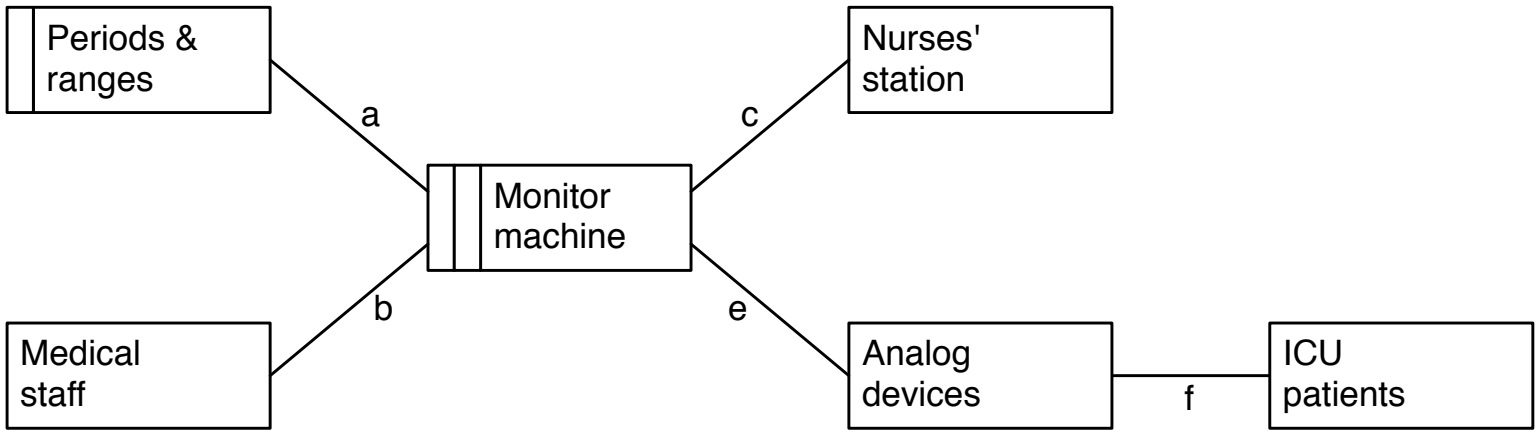
Do we really need the ICU patients domain?

Earlier we asked a similar question about the "*factors database*".

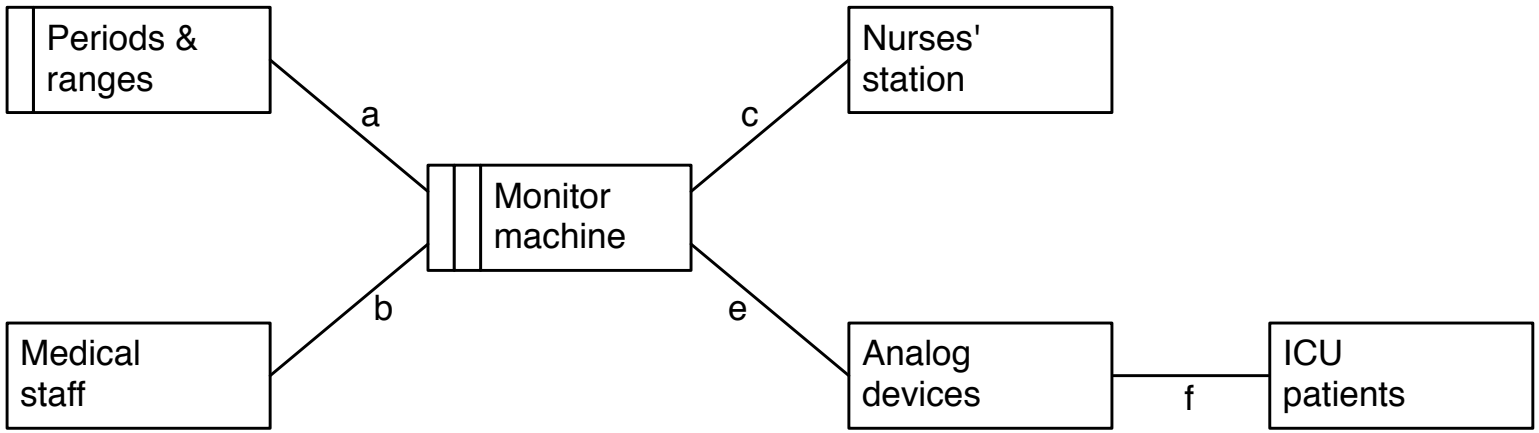
The answer was: **no**.

Contents

Patient Monitoring Problem: Domain Interfaces



Patient Monitoring Problem: Domain Interfaces

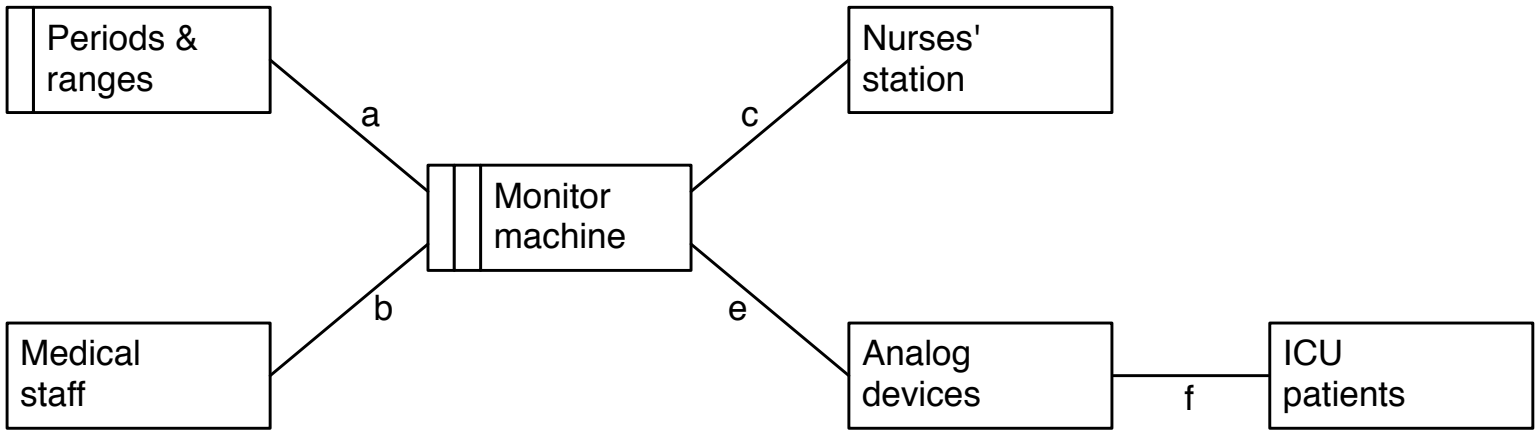


a:	Period, Range, PatientName, Factor	c:	Notify
b:	EnterPeriod, EnterRange, EnterPatientName, EnterFactor	e:	RegisterValue
		f:	FactorEvidence

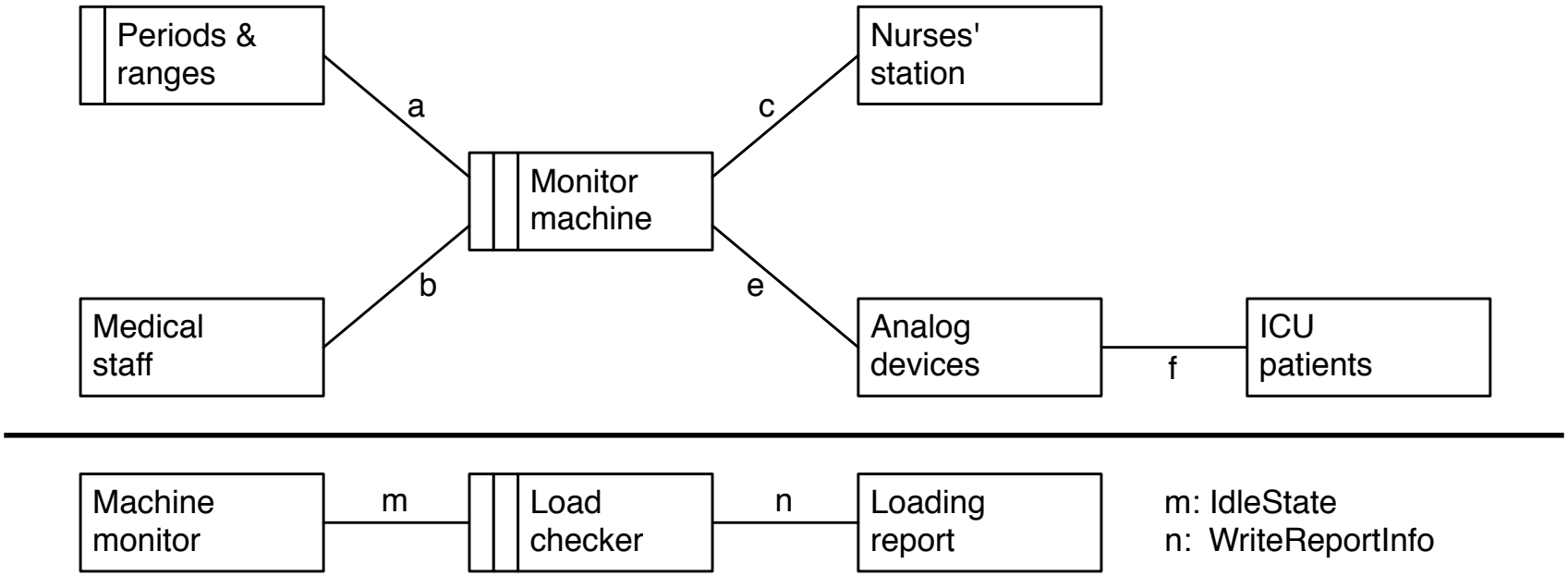
Each interface is a set of *shared phenomena*.

A phenomenon is an *event*, *state* or *value*.

Patient Monitoring Problem: Machine Domains



Patient Monitoring Problem: Machine Domains

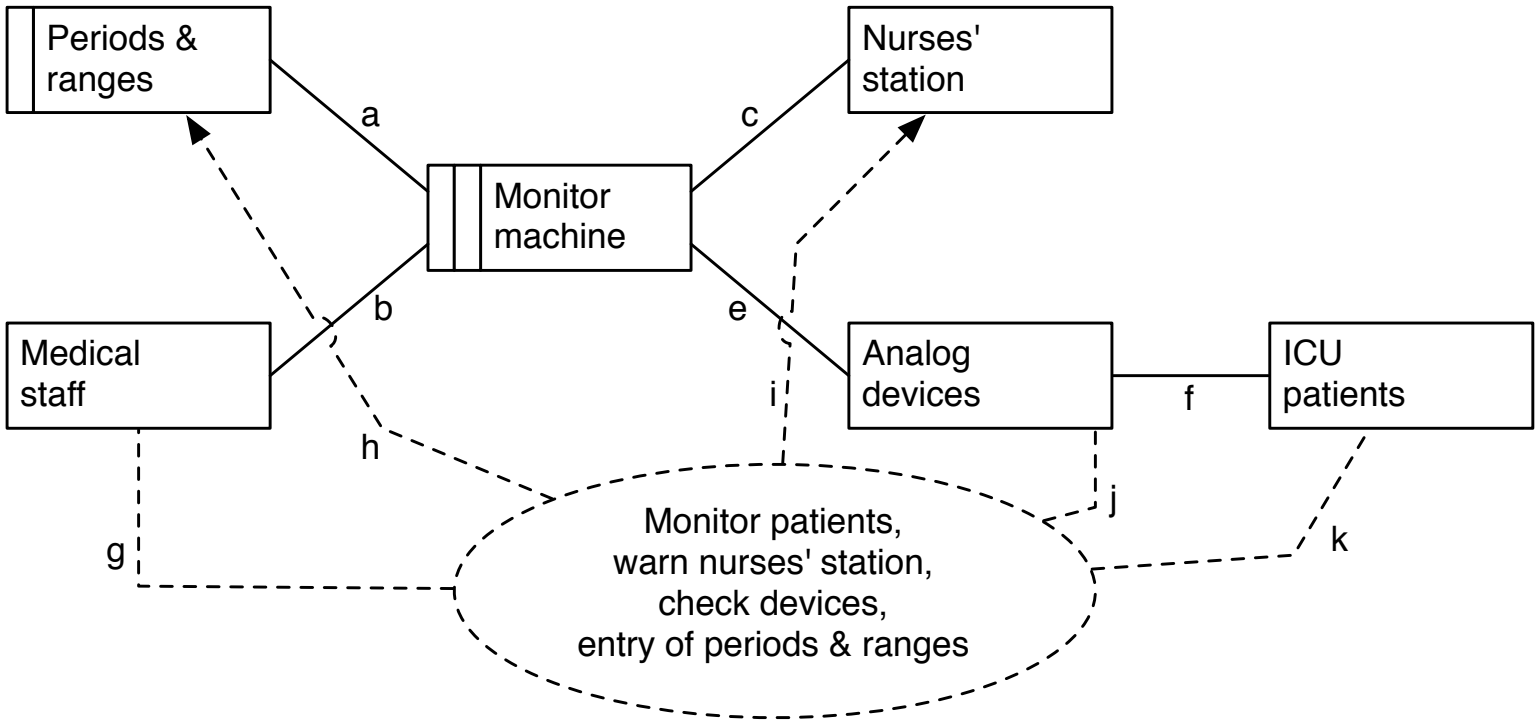


Computers can also be given domains.
A machine domain of one problem can be a given domain of another.

E.g. a problem of analysing whether the monitor machine of the patient monitoring problem is overloaded.

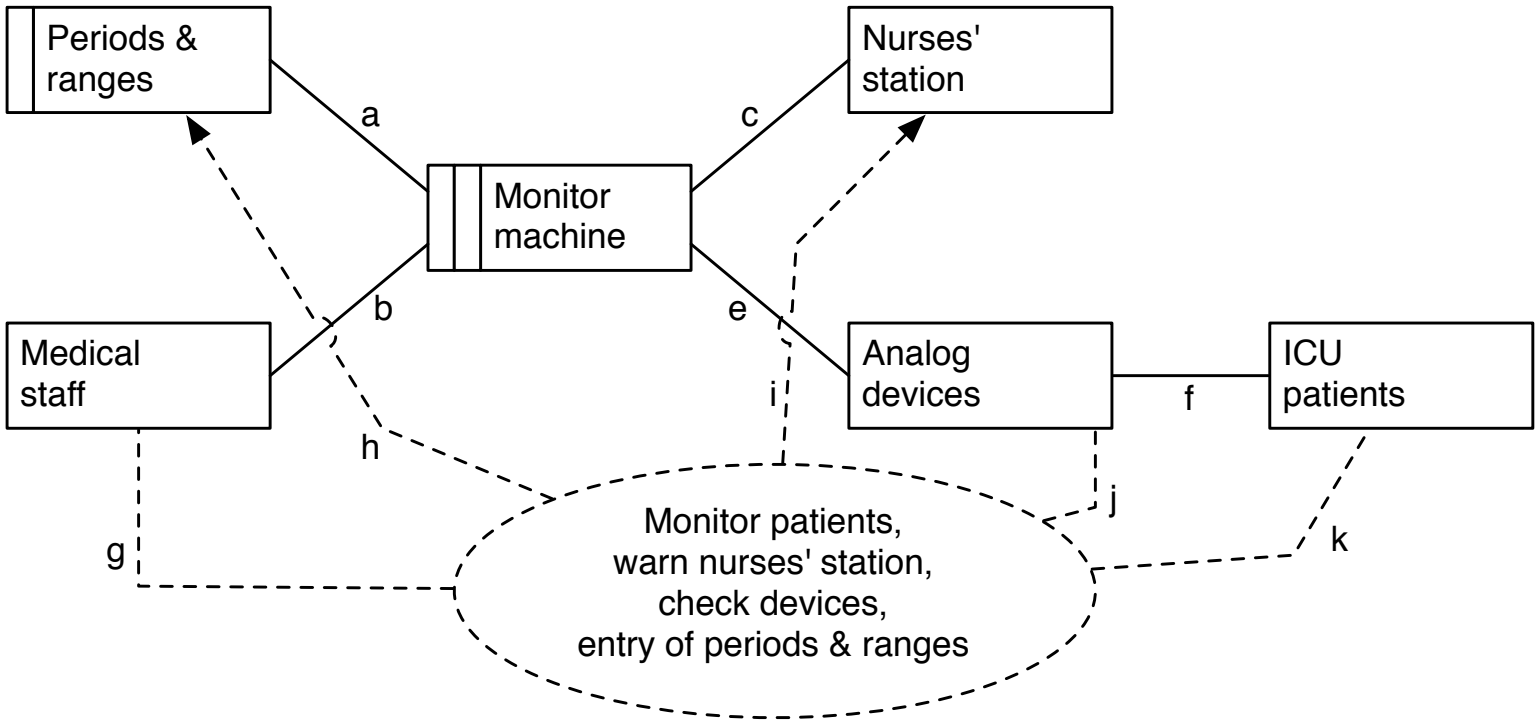
Contents

Patient Monitoring Problem: Problem Diagrams



A *problem diagram* shows how requirements are related to problem domains.

Patient Monitoring Problem: Problem Diagrams



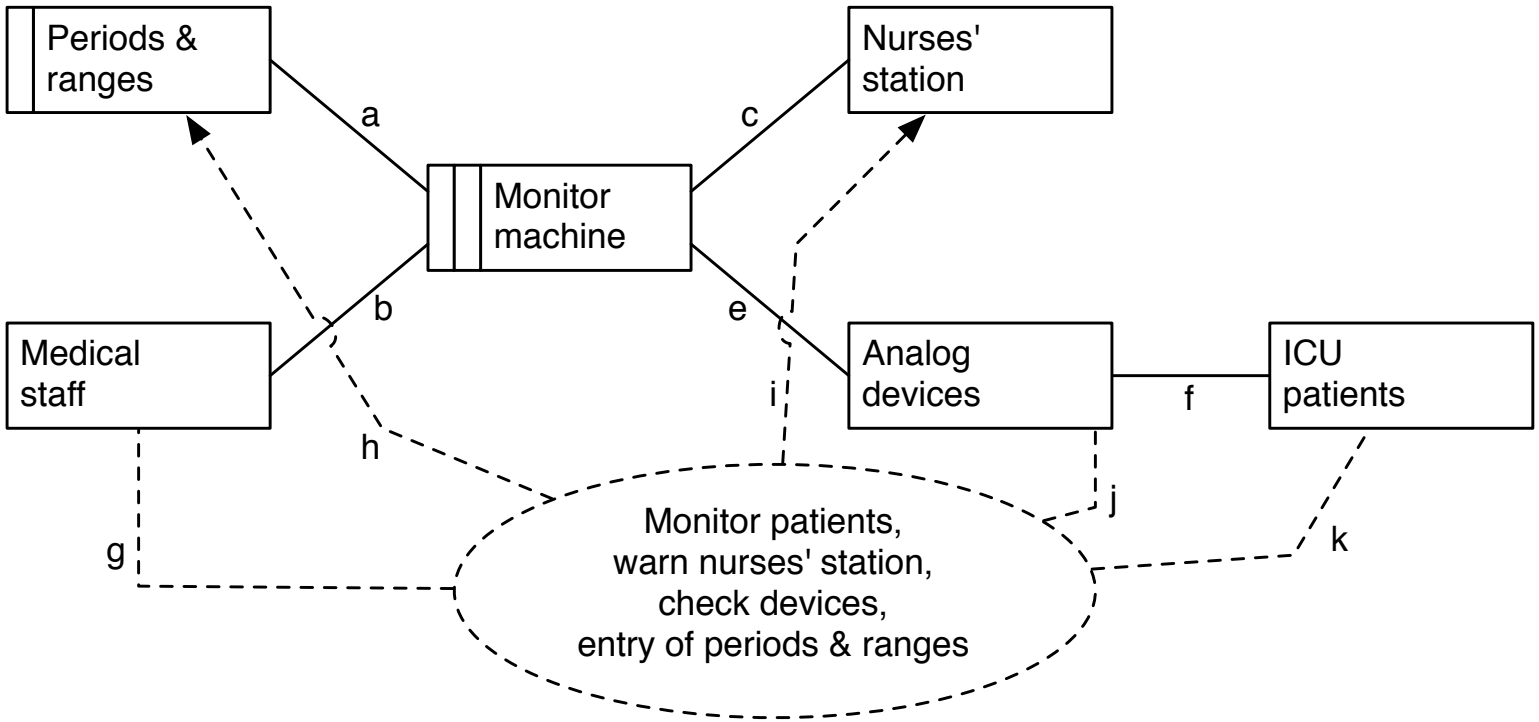
A *problem diagram* shows how requirements are related to problem domains.

a *requirement* to be respected

a *requirement reference*: the requirement refers to phenomena of the connected domain

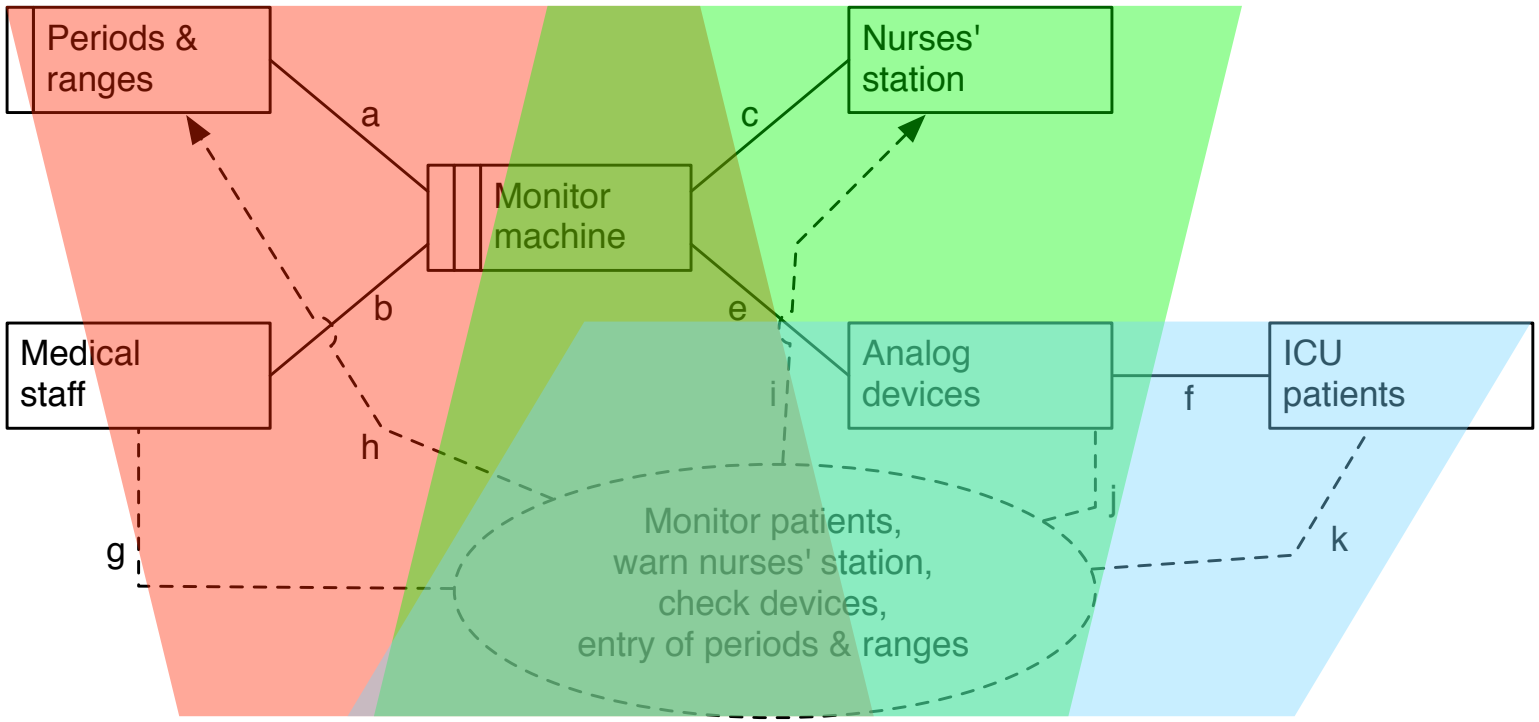
a *constraining requirement reference* the requirement constrains phenomena of the connected domain

Patient Monitoring Problem



This problem diagram is too complex:
the requirement mixes many aspects of the system.

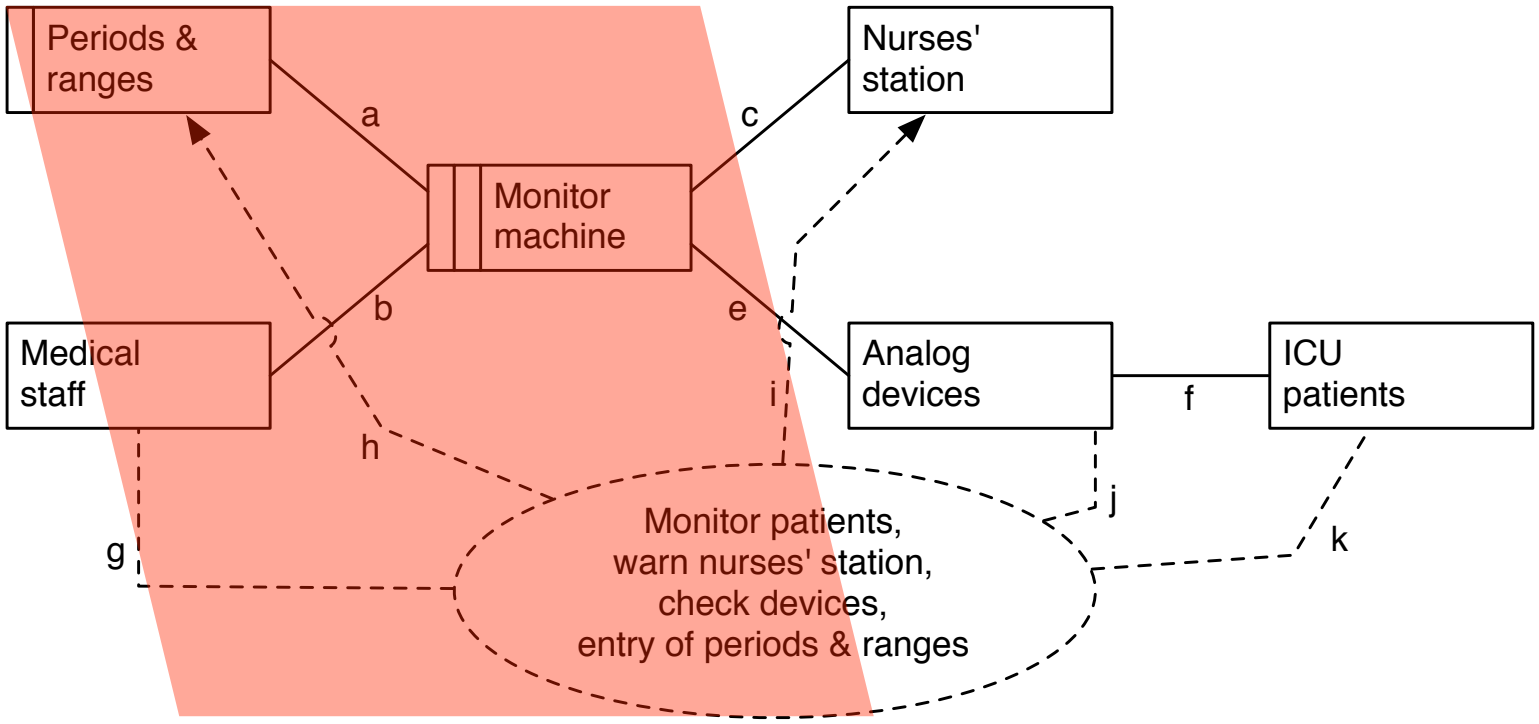
Patient Monitoring Problem



This problem diagram is too complex:
the requirement mixes many aspects of the system.

We should work on (smaller) *projections* of the diagram.

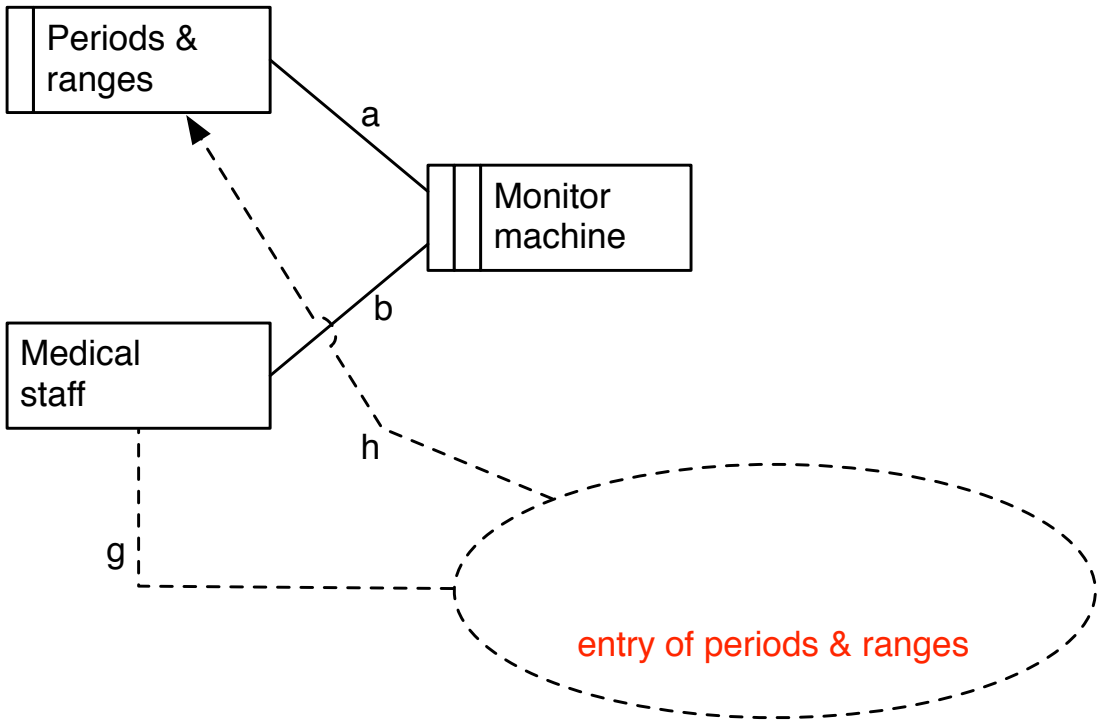
Patient Monitoring Problem



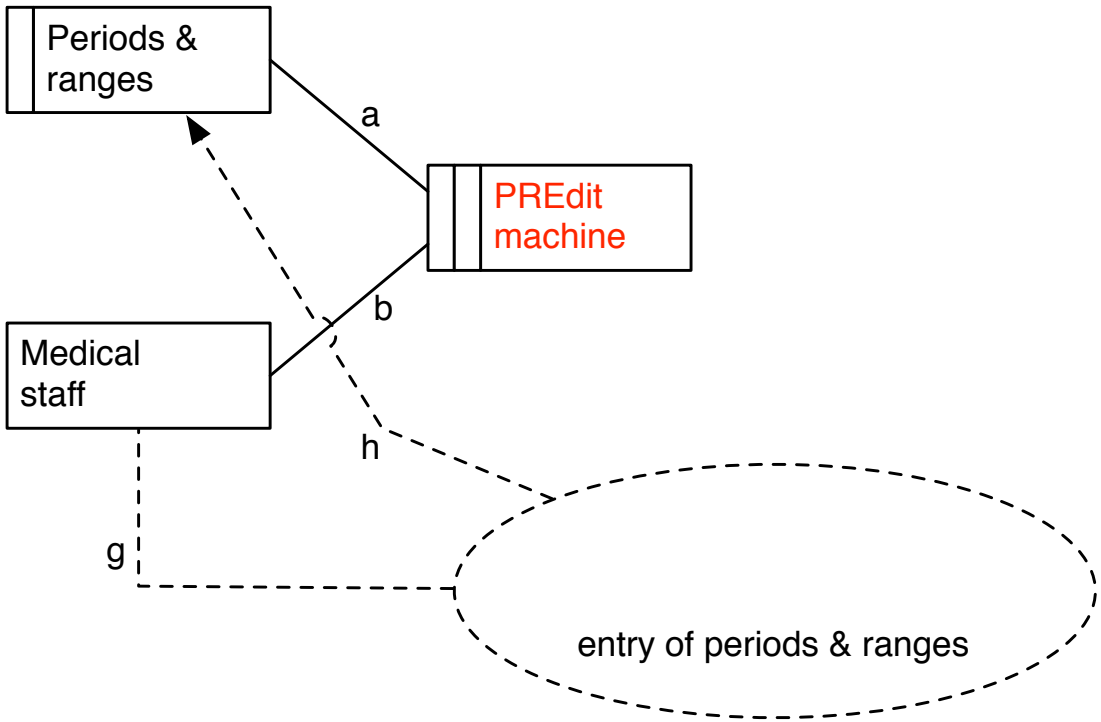
This problem diagram is too complex:
the requirement mixes many aspects of the system.

We should work on (smaller) *projections* of the diagram.

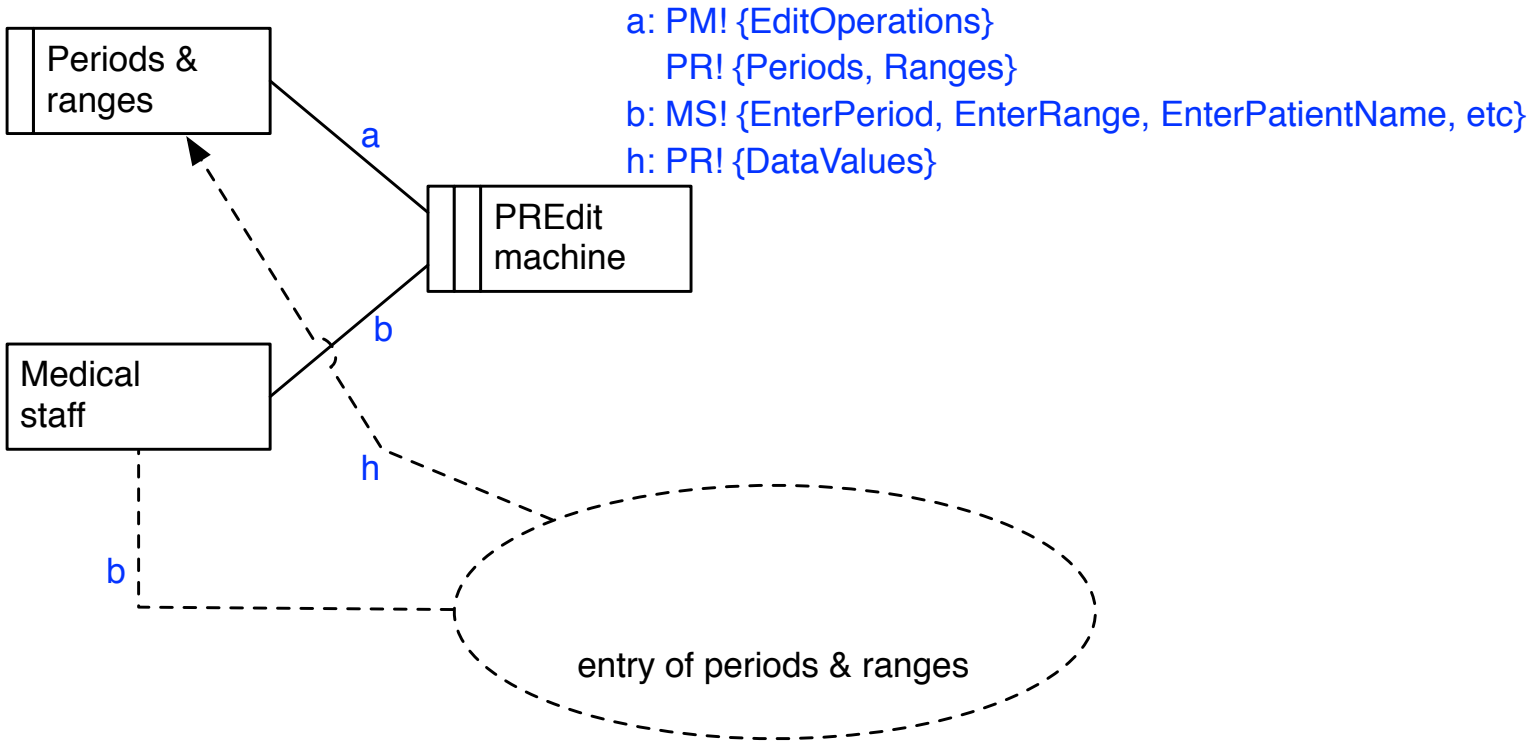
Patient Monitoring Problem: Subproblem Diagram



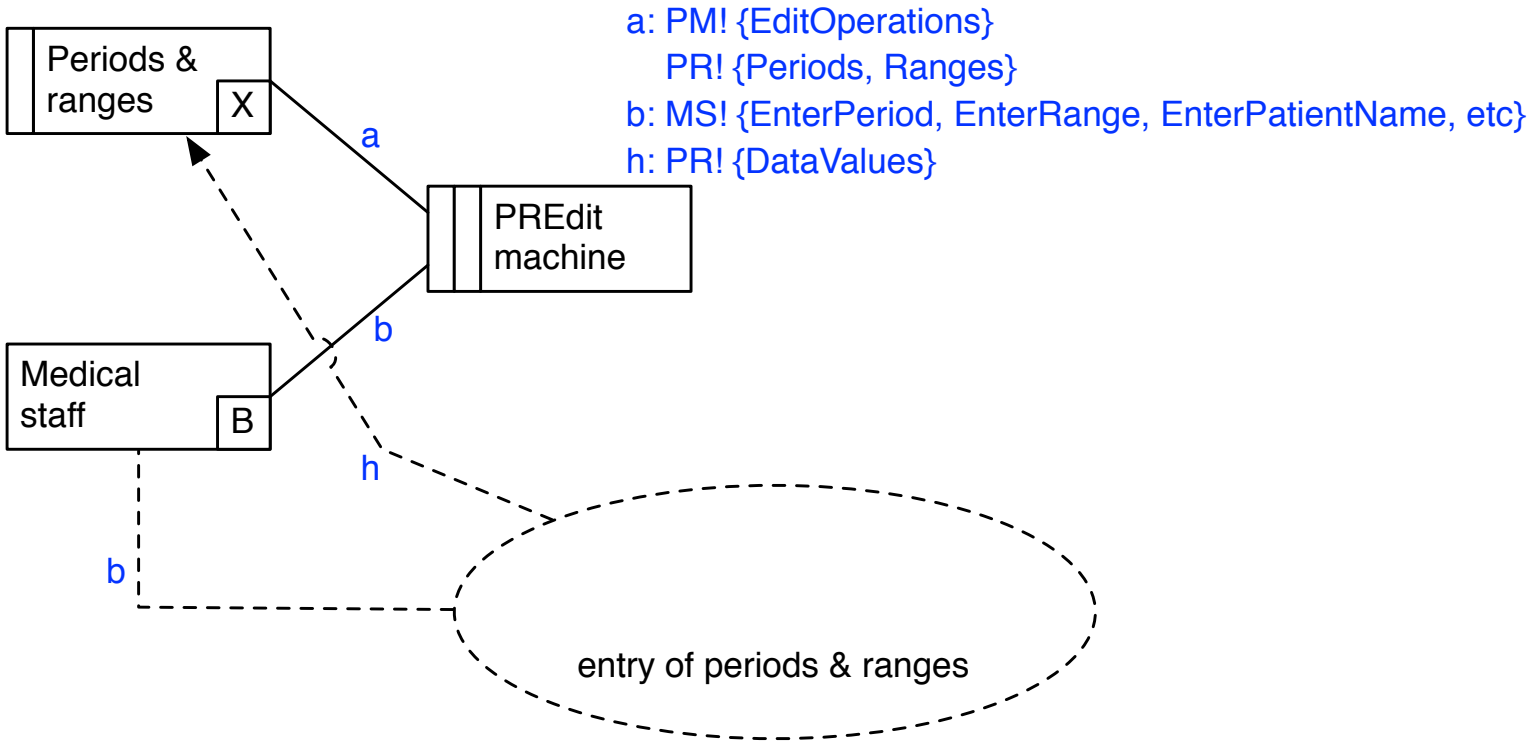
Patient Monitoring Problem: Subproblem Diagram



Patient Monitoring Problem: Subproblem Diagram



Patient Monitoring Problem: Subproblem Diagram



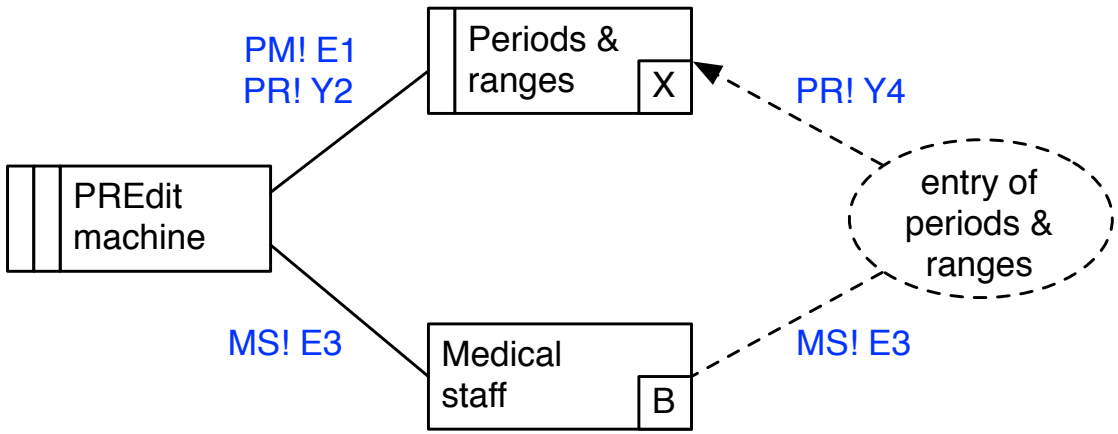
X — a *lexical domain*: physical representation of data

C — a *causal domain*: predicable relationship between phenomena, e.g., a motor

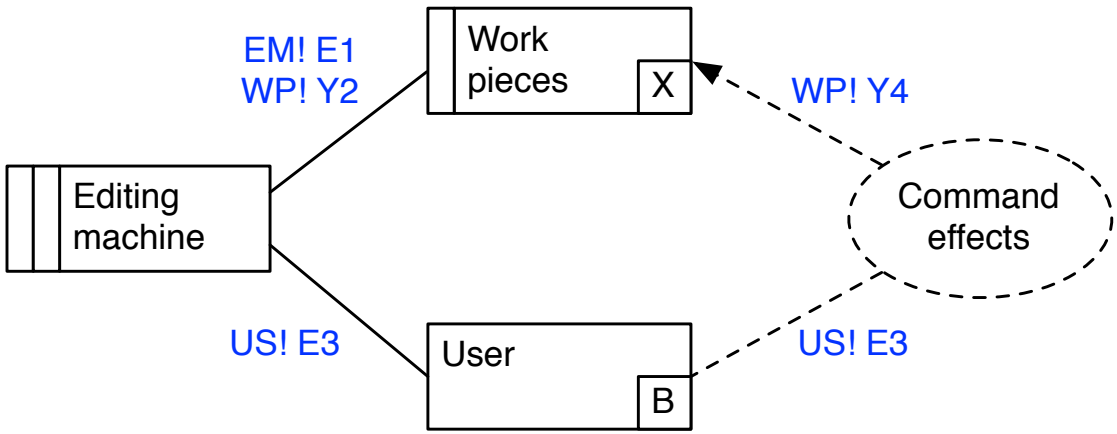
B — a *biddable domain*: usually consists of people, no predictable internal causality

Contents

Patient Monitoring Problem

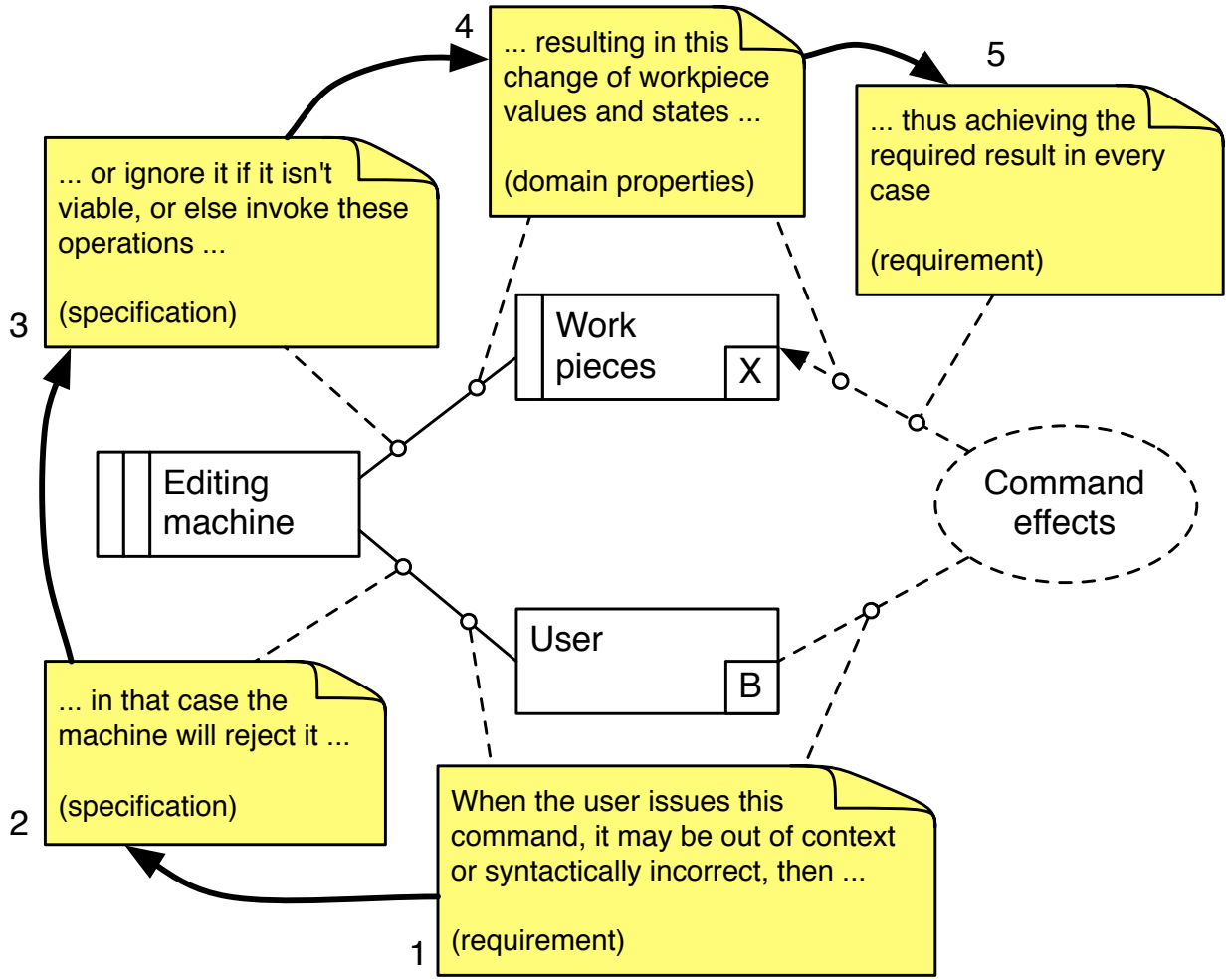


Patient Monitoring Problem: Problem Frame



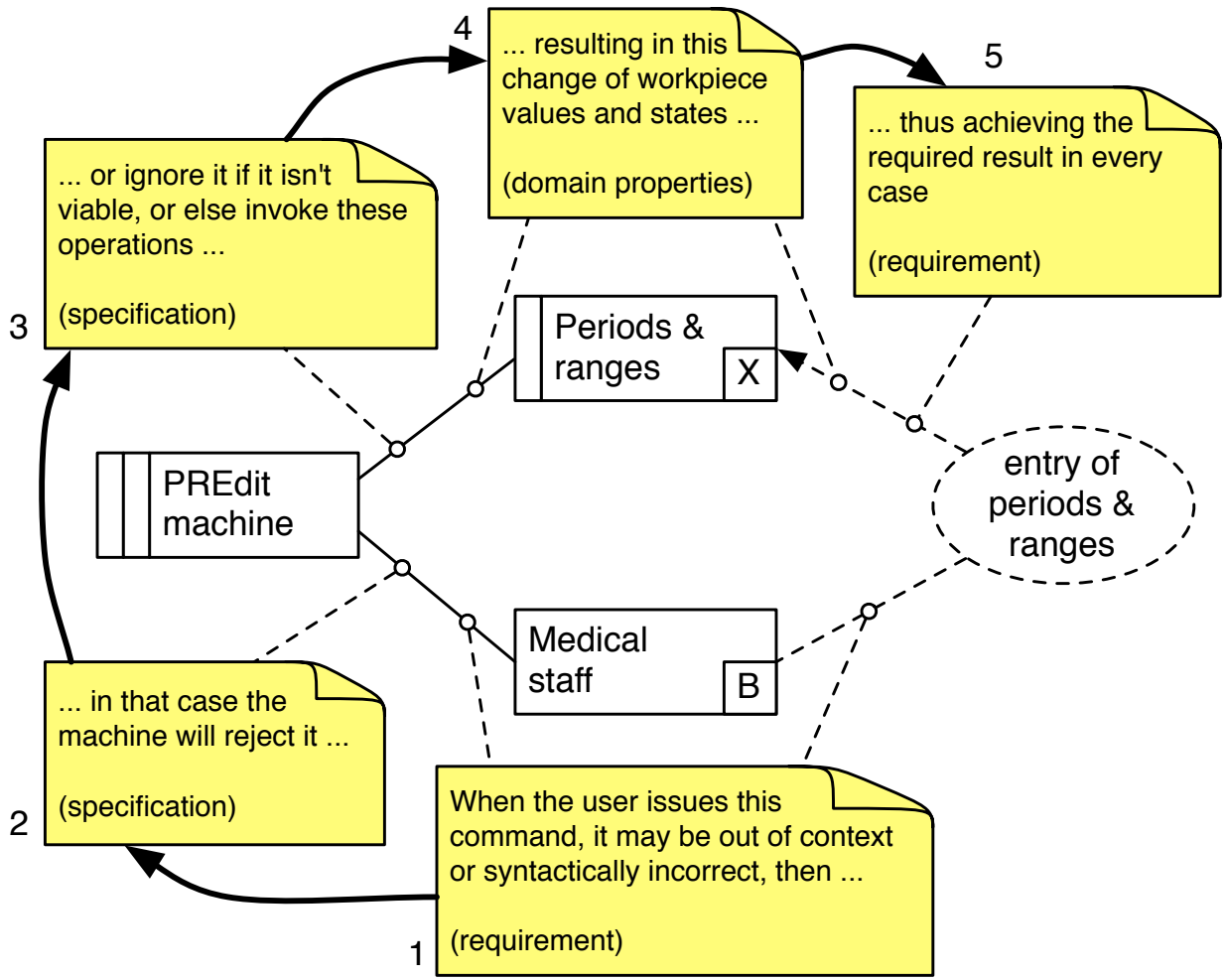
A *problem frame* captures and defines a commonly found class of simple subproblem.

Patient Monitoring Problem: Frame Concern



The *frame concern* provides an argument that the proposed machine ensures that the requirements are satisfied.

Patient Monitoring Problem



We can now apply the frame concern to our specific subproblem.

Contents

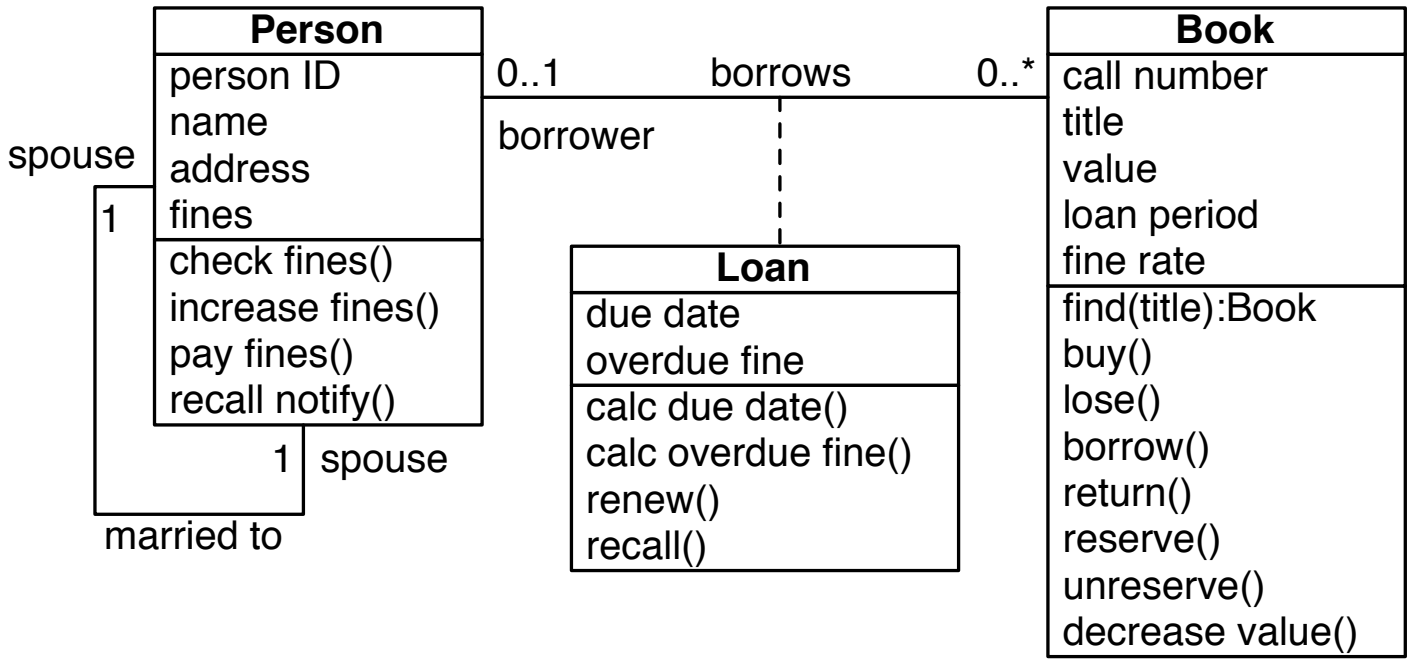
Modelling With The UML

UML:

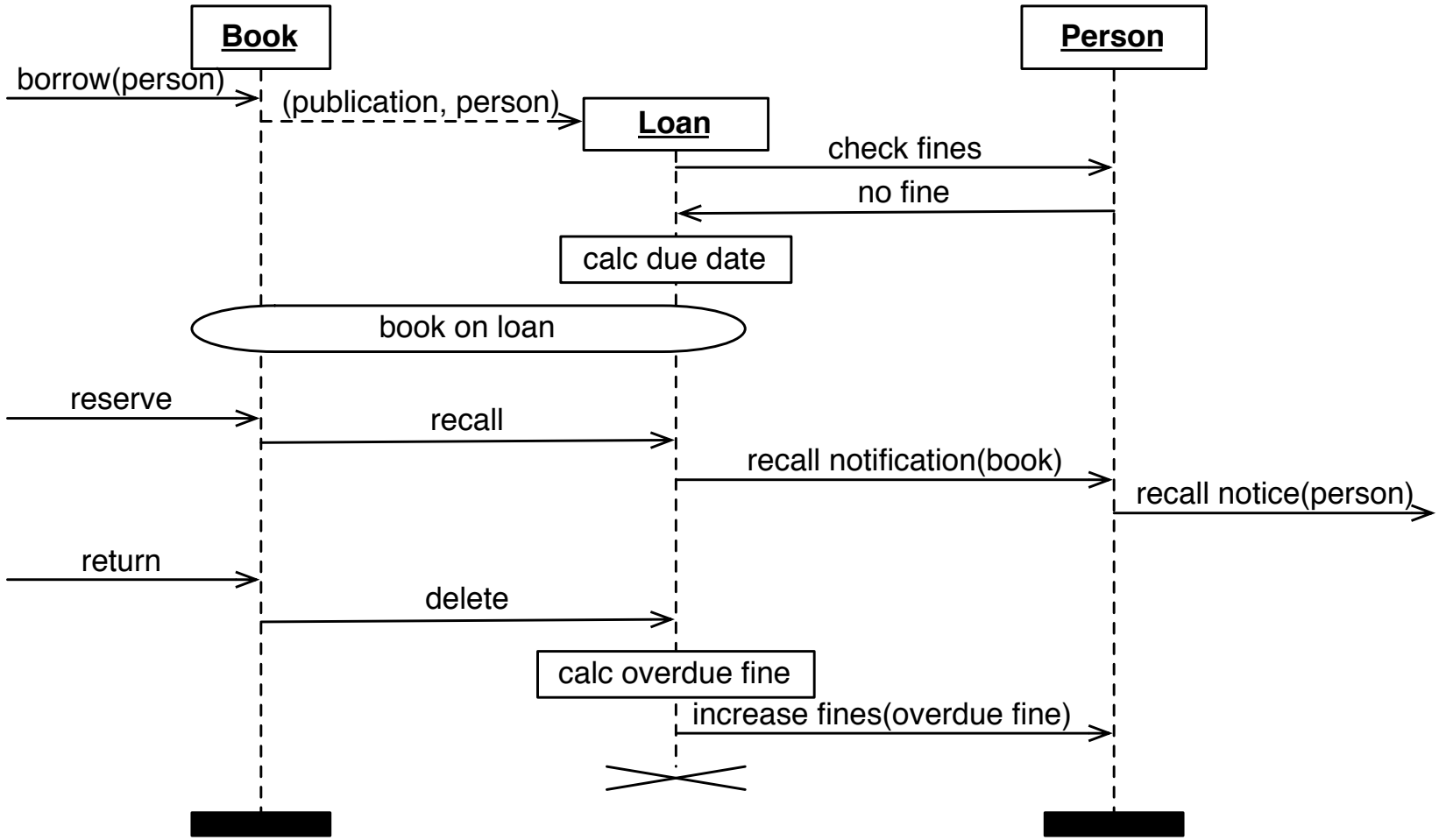
- Collection of notations used to document software specifications and designs
- Oriented towards implementation on a computer
- We look at:
  - Class diagrams
  - Message sequence charts
  - State chart diagrams
  - Use cases
  - The object constraint language

Contents

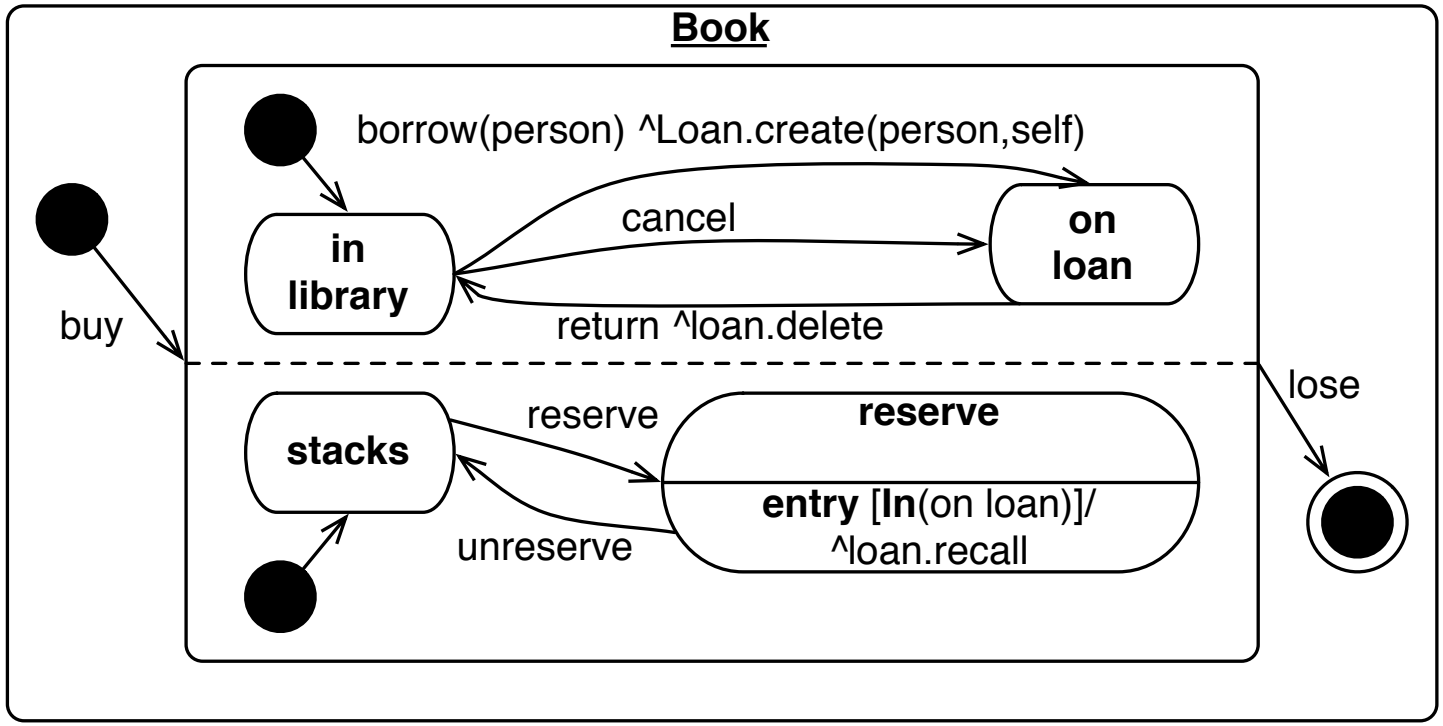
Library: Class Diagram



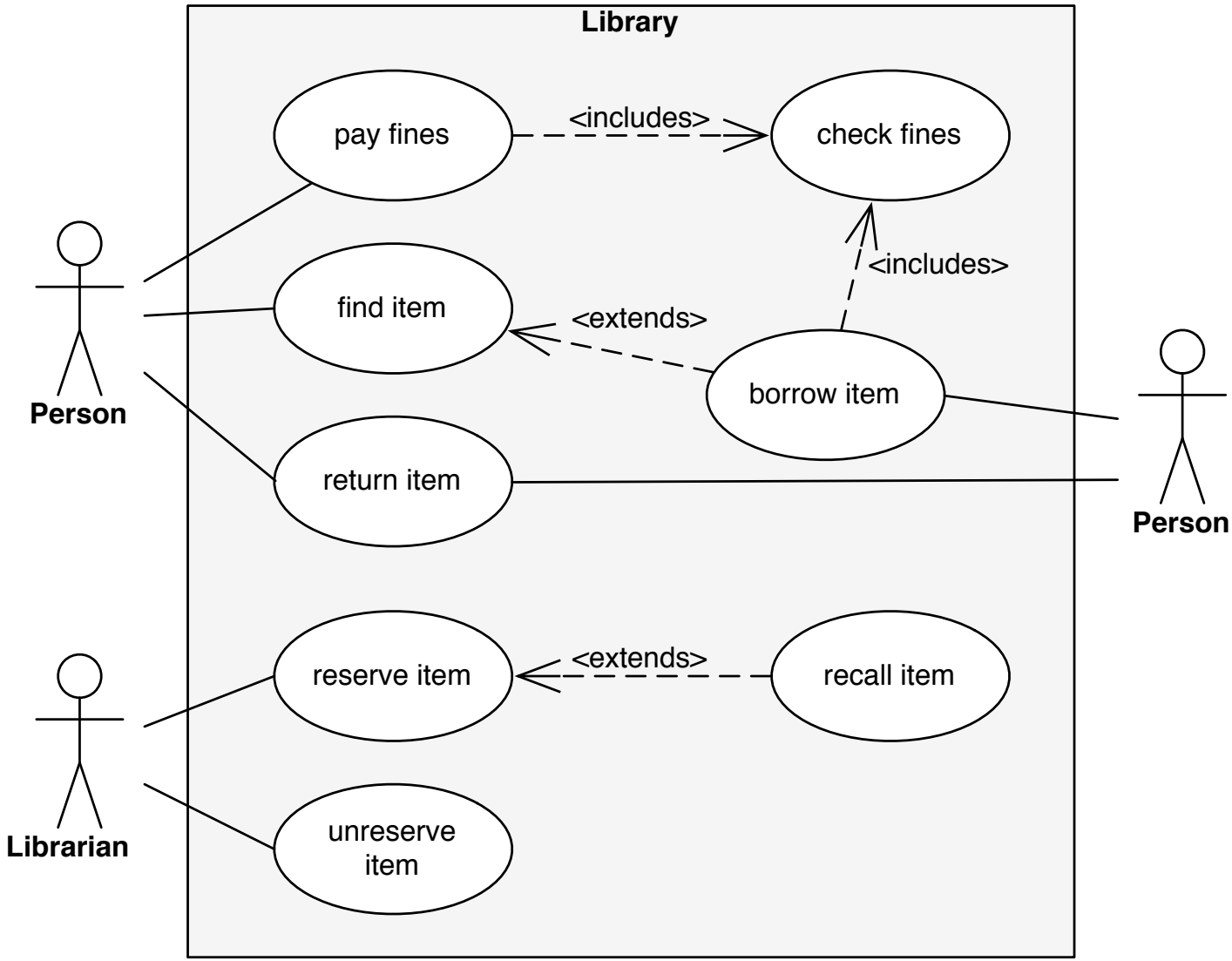
Library: Message Sequence Chart



Library: State Chart Diagrams



Library: Use Cases



Library: Object Constraint Language