# 3

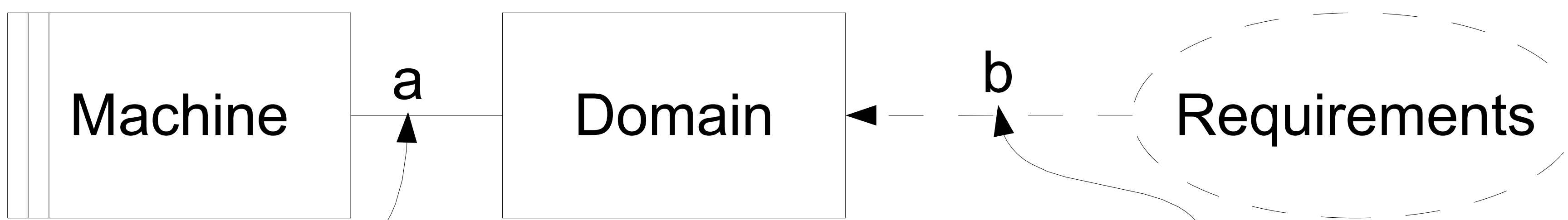
- The Problem Frames approach
  - definition
  - (further) examples

# *Contex diagram vs. Problem diagram*

- The diagrams we have seen so far are **context diagrams**, framing the problem in the real world
  - summary: domains and interfaces
- Problem diagrams supplement those with requirements
  - expressed in terms of interfaces
  - referencing the non-machine domains

# Context diagram vs. Problem diagram

- Notation:



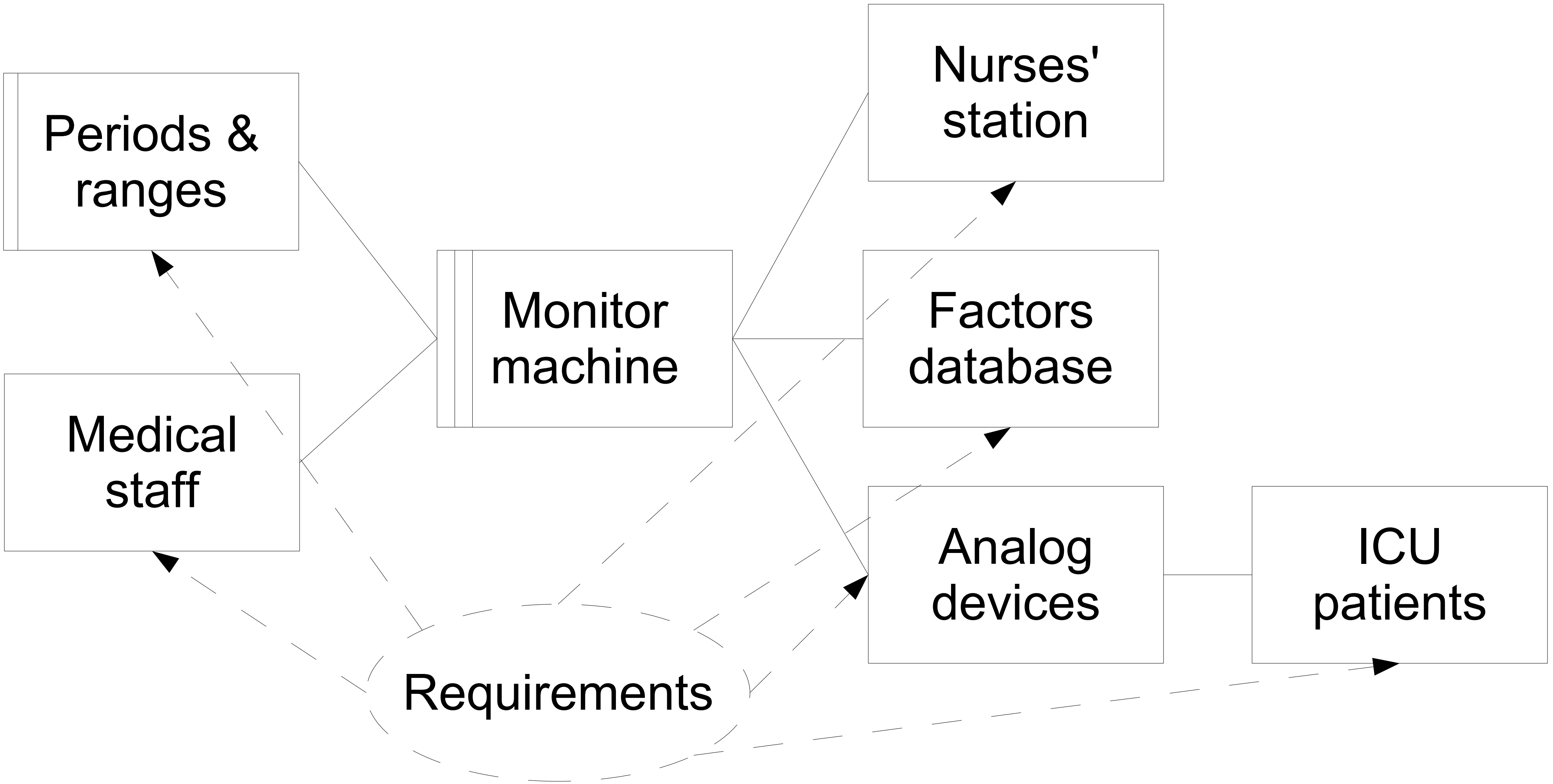

a shared phenomena reference, these phenomena are shared between M and D, and controlled by one of them. Notation: M!name (controlled by Machine) or D!name (controlled by Domain)

a requirement reference, i.e. a predicate that is desired (by someone) to be true in the Domain, once the Machine is in place. Notice that the requirements can only refer to D's phenomena (not to the machine internal state).

# *Complex problems*

- Of course, just adding a “Requirements” bubble, connected with all the domains, does not help much
- Since we included in the context diagram *all* the domains of relevance for the requirements, by definition we will have arrows from Requirements to all of them
  - not very useful, just adding complexity

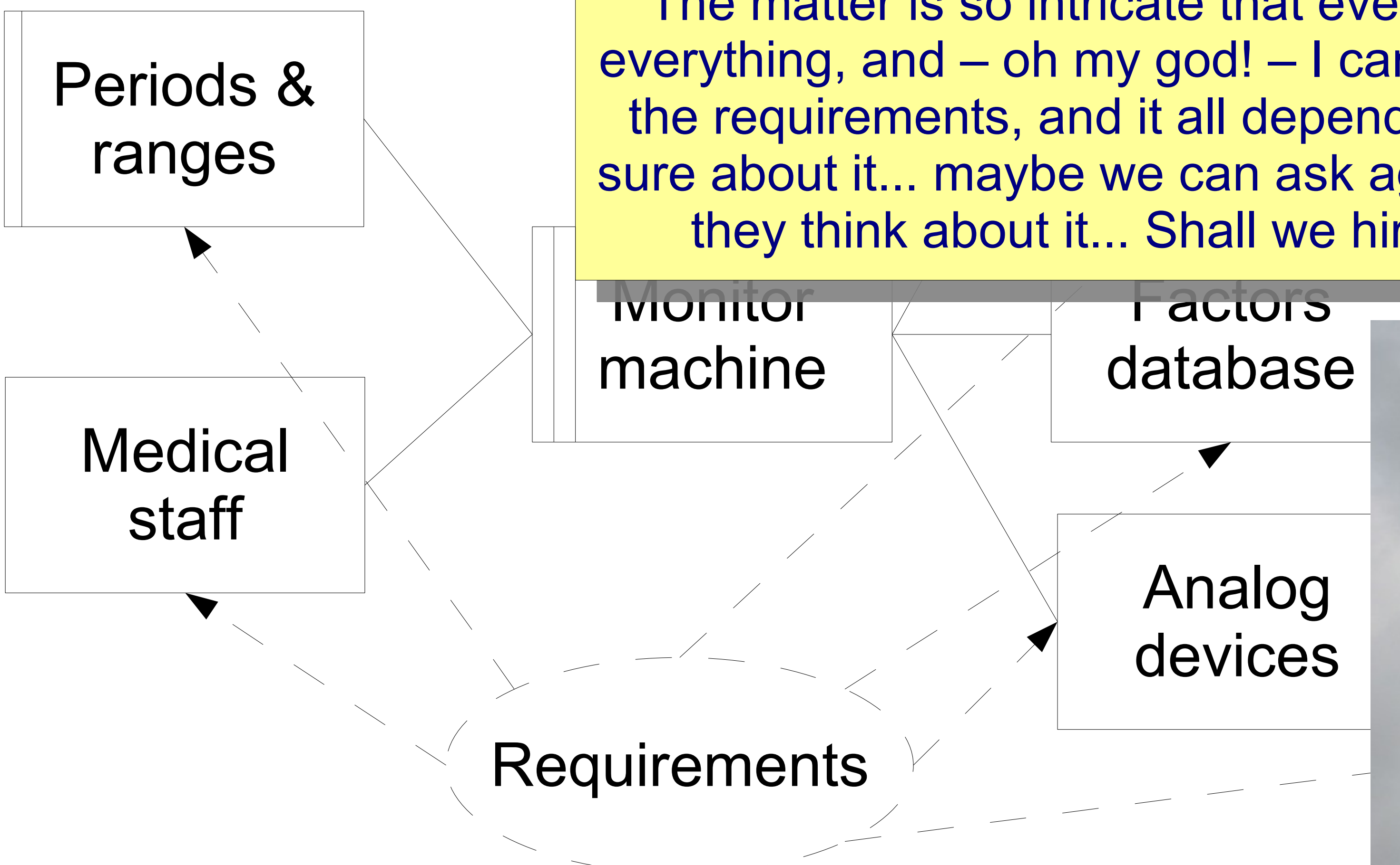
# Complex problems

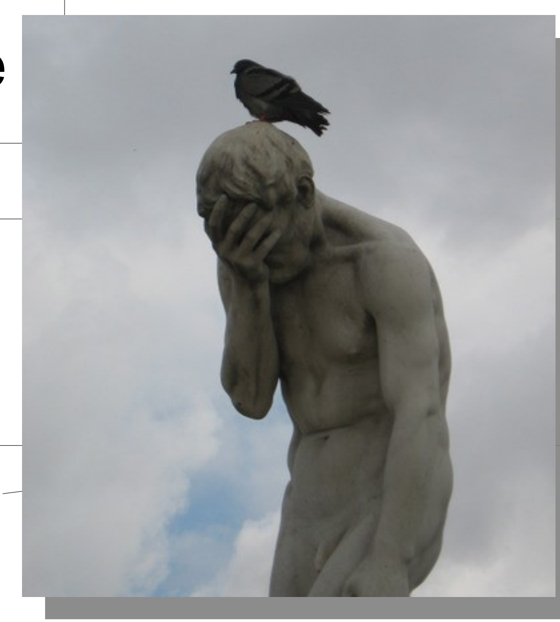


# *Complex problems*

**<u>Spaghetti requirements</u>**

“The matter is so intricate that everything is related to everything, and – oh my god! – I cannot keep track of all the requirements, and it all depends, and no, I am not sure about it... maybe we can ask again everybody what they think about it... Shall we hire a consultant?”



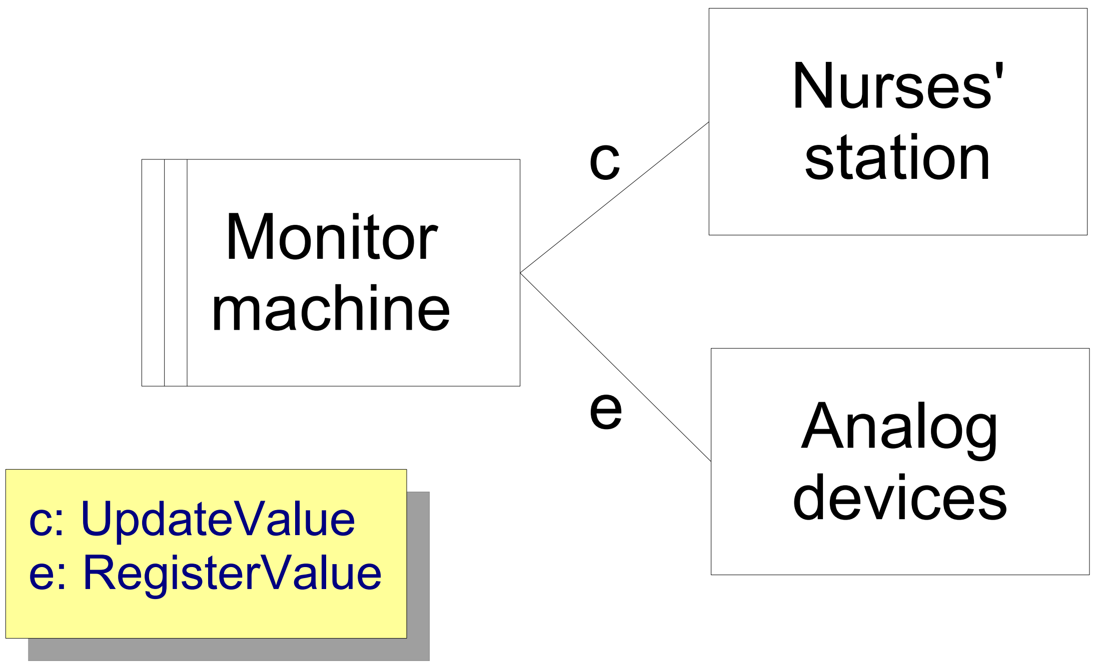



# *Complex problems*

- The classical way to manage complexity is by **decomposition** into sub-problems
  - by analogy: if a task is complex, divide it into simpler steps
  - in contrast: steps are sequential and distinct, subproblems are often not
- We will say more about decomposition later on; for now let us focus on simple sub-problems

# Example (simpler)

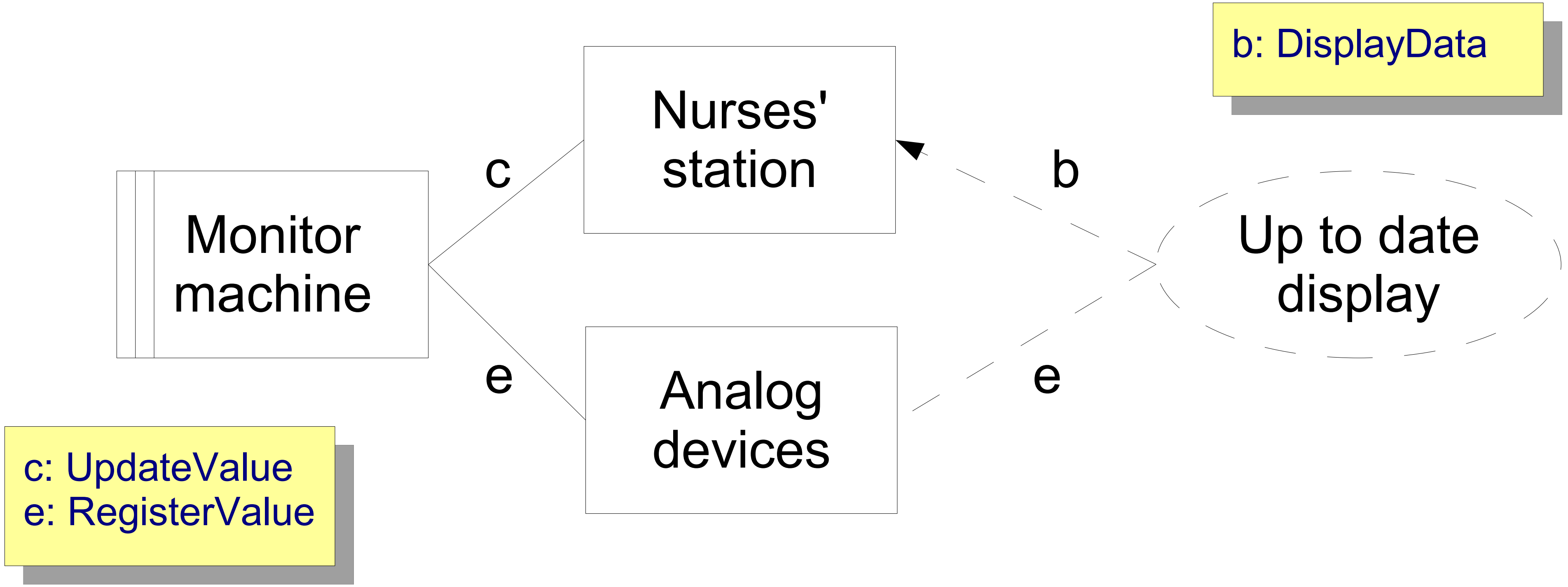
- Let us consider a simpler (related) problem, i.e. showing the raw values of the analog sensors' readings on the nurse station




- This is the context diagram
- What about the requirements?

# *Example (simpler)*

- Let us consider a simpler (related) problem, i.e. showing the raw values of the analog sensors' readings on the nurse station

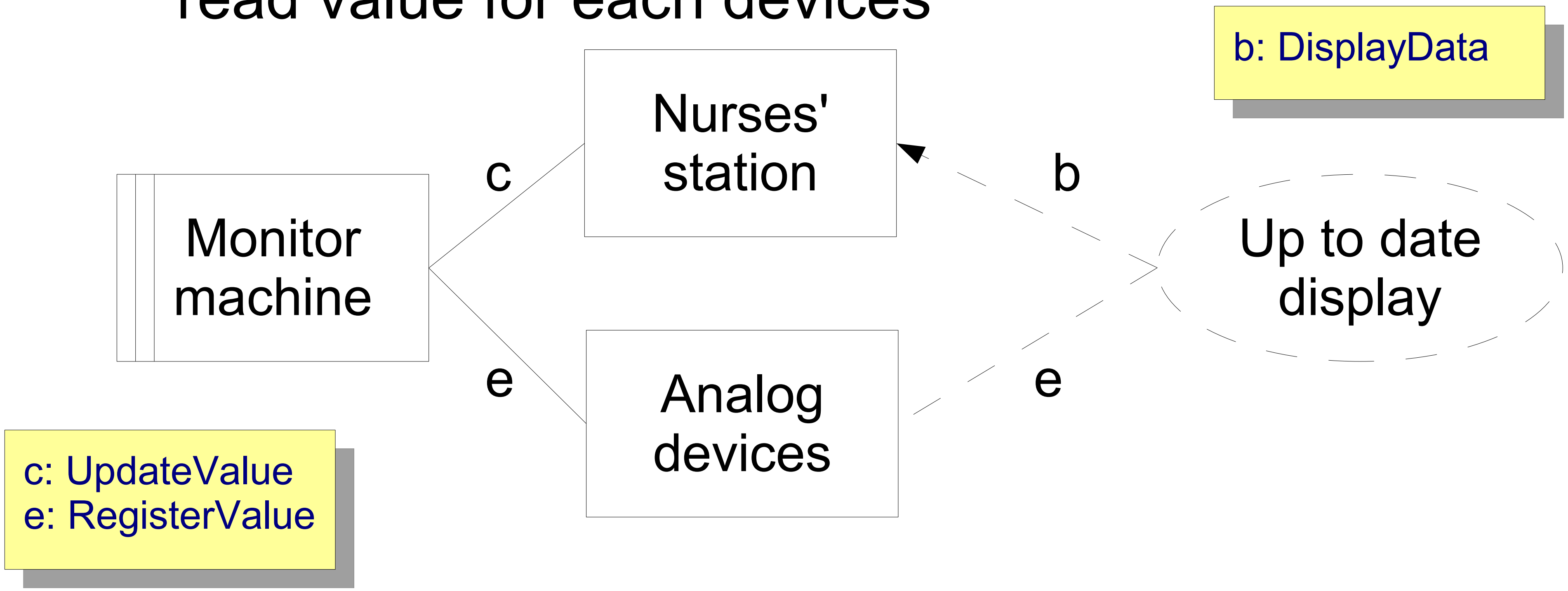


# *Writing the requirements*

- But how are requirements written?
  - Not really relevant for our discussion
  - Main goal: relationships between interface phenomena of the domains must be clear
- Formality?
  - Sure, if you need the assurance and can handle it
  - Logics, automata, state diagrams, equational, ...
- Informality?
  - Sure, as long as it is rigorous enough to support implementing the specification
  - Natural language, sketches, ...

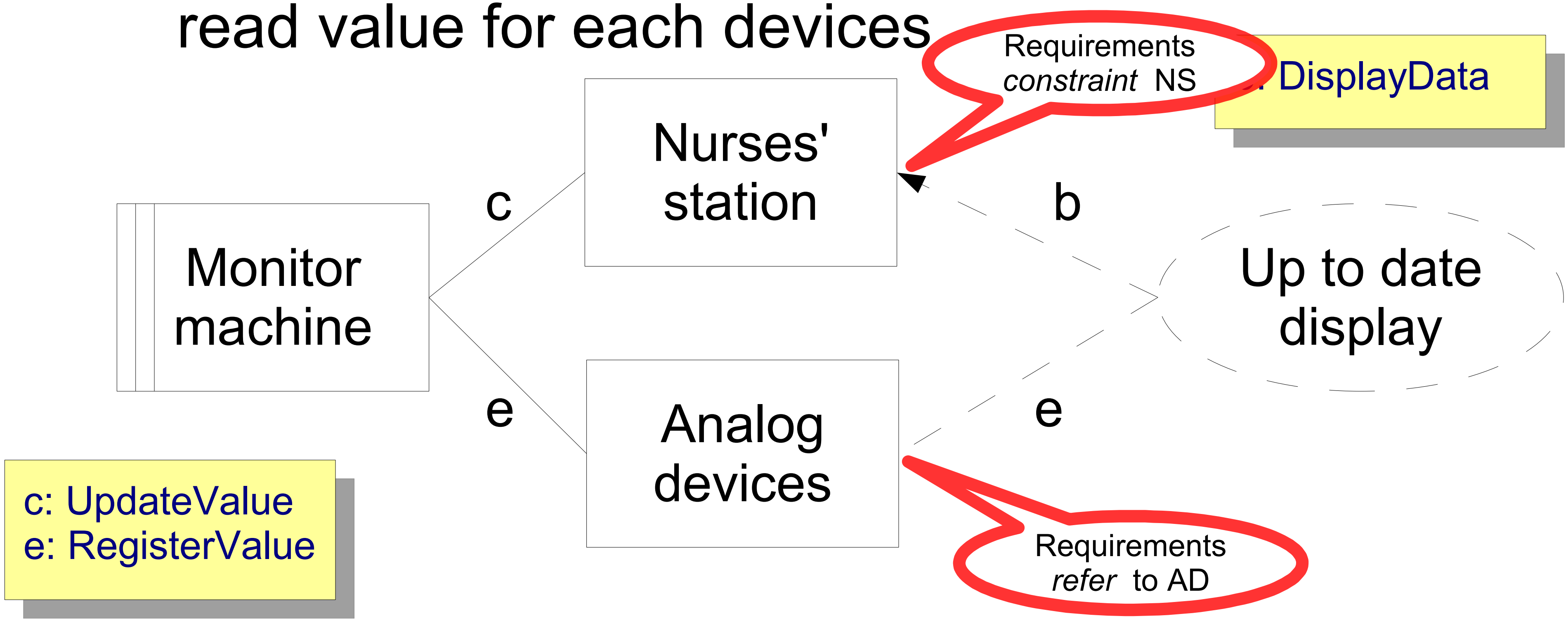
# Example (simpler)

- Up to date display:
  - $\forall\, AD!RegisterValue(factor, v).\, NS!DisplayData(factor) = v$
  - Nurses' station must display the most recently read value for each devices



# *Example (simpler)*

- Up to date display:
  - $\forall\, AD!RegisterValue(factor, v) . NS!DisplayData(factor) = v$
  - Nurses' station must display the most recently read value for each devices



# *Solving the problem*

- We need to
  - Describe the requirements (optative description, how the customer would like the world to be)
  - Describe the domain properties (indicative description, how physical domains will react to phenomena)
  - Build the machine specification (optative description, how the machine should react at its interface)
- Once more: $S \cup D \models R$

# Solving the problem

- R - Requirements: (up to date display)

  $\forall\, AD!RegisterValue(factor, v).\ NS!DisplayData(factor) = v$

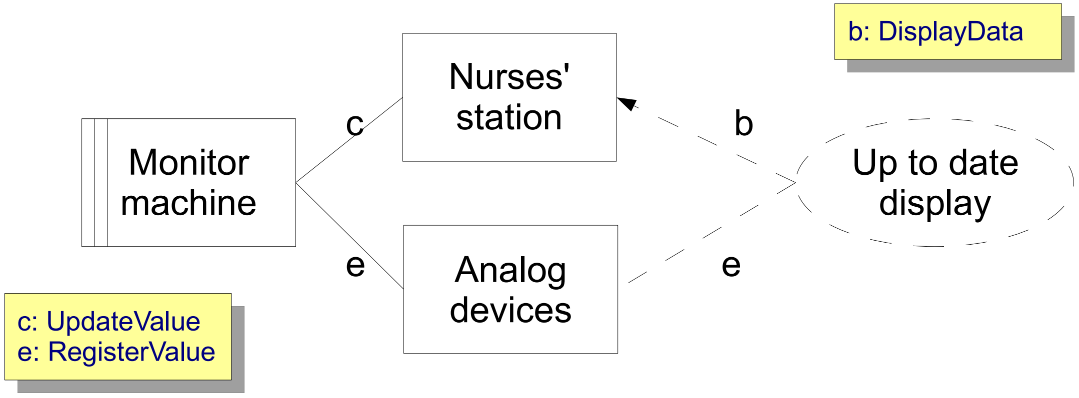
- D - Domains: (nurses' station is working)

  $MM!UpdateValue(factor, v) \Rightarrow NS!DisplayData(factor) = v$

- S - Specification: (program to write)

  $AD!RegisterValue(factor, v) \Rightarrow MM!UpdateValue(factor, v)$

- Hence: $S \cup D \models R$



# Solving the problem
## *(the small print)*

- Notice that what we have presented is a simplified version (for clarity)
- Not a sub-problem of the original problem
  - Some difference:
    - The original problem stated that the periods of samples where to be configured, hence it was a *pull* model
    - In this latter version, we assumed a *push* model, with the sensors sending the value: $AD\,!\,RegisterValue(factor, v)$
  - In the original, causality would follow a different chain
  - More on this later on

# Problem solved?

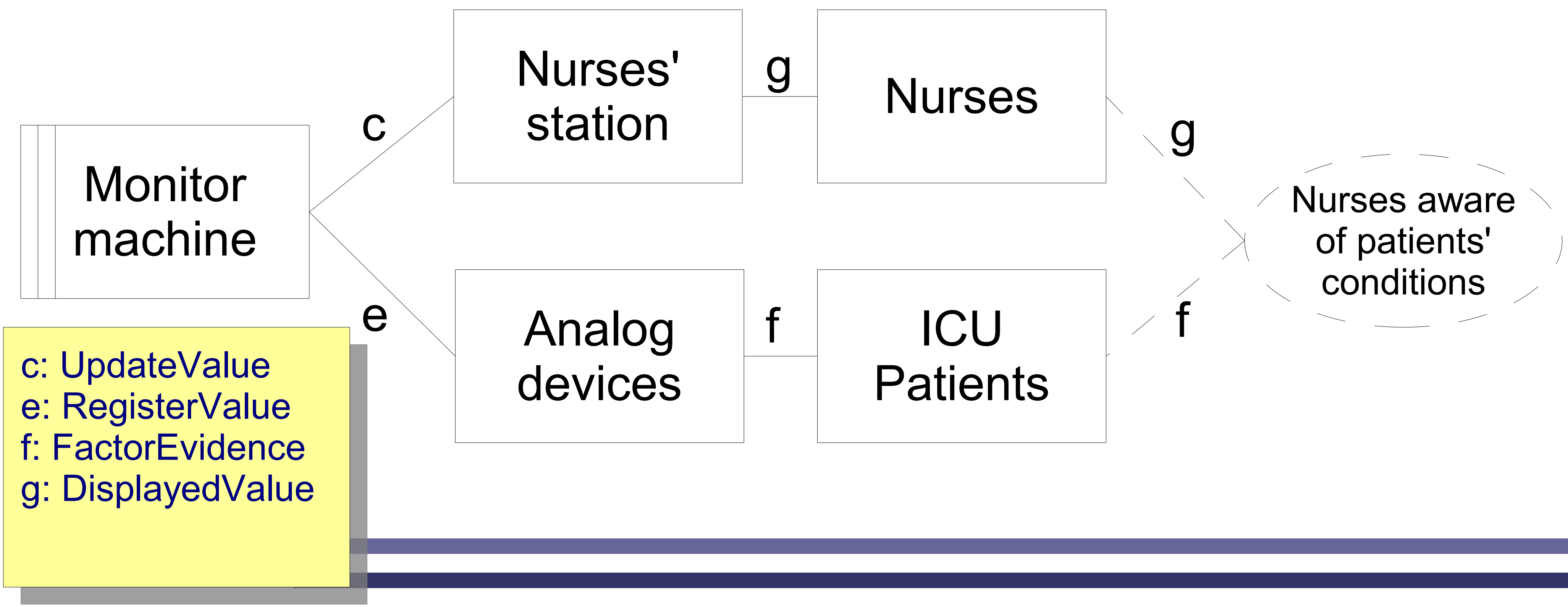
- From a purely requirements view, yes
  - The previous problem diagram contains enough information to realize the specification
  - Plus, of course, needed “technical” details
- From a human-centric view, no
  - We have not taken into account any human-related issue
  - We have solved the *correctness* problem
  - Did not do any *elicitation* really
  - How do human issues fit?

# *Problem solved?*

- What about distribution?
  - We have ignored how different pieces of equipment interact
  - Those were really parts of a (miniature) distributed system
  - Sensors and related electronics
  - Nurses' station
- Shall we focus more on distribution here?
  - Hard to give a 'blanket' answer
  - How often do communication infrastructure break?
  - Does it introduce significant delays?
- Probably not (here)

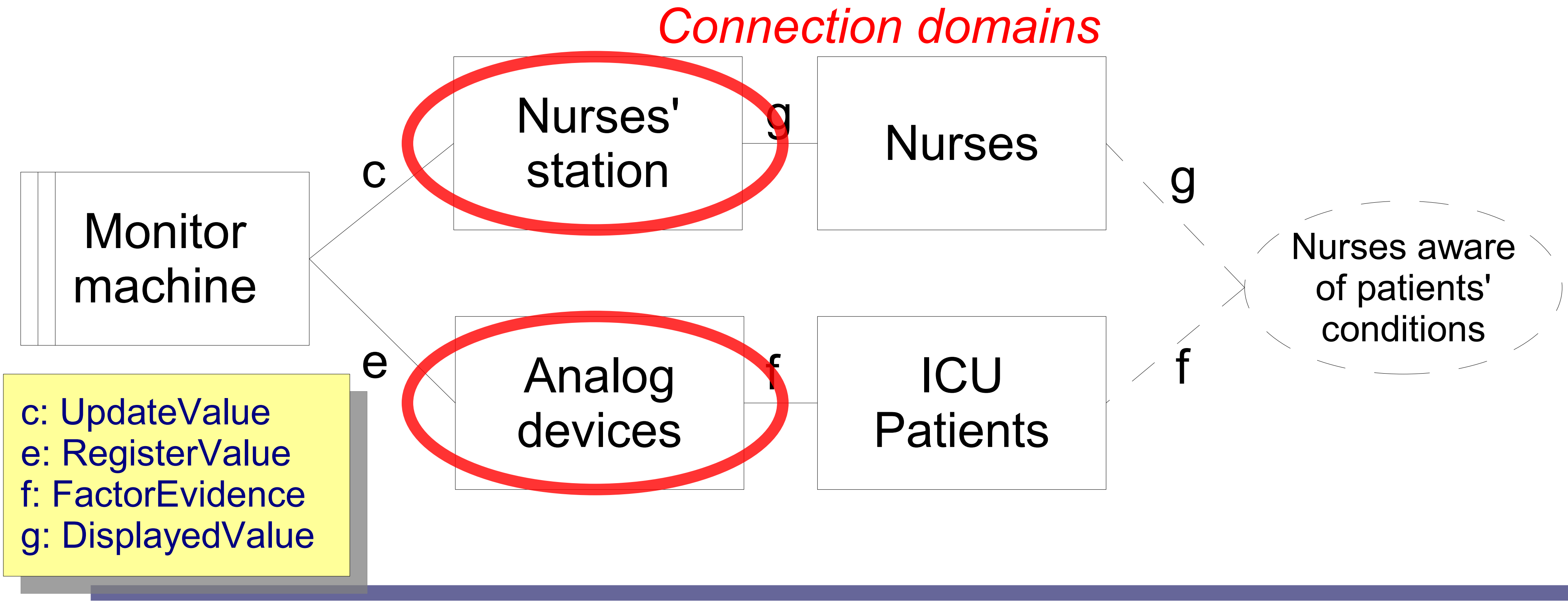
# *Example (simpler): adding humans*

- Extending the problem with humans
- New requirements:
  - Nurses must be aware of each patient's condition



# Example (simpler): adding humans

- Extending the problem with humans
- New requirements:
  - Nurses must be aware of each patient's condition



# *Connection domains*

- Connection domains are characterized as:
  - They connect two other domains
  - They transfer phenomena on one interface to phenomena on another
  - They have properties that is worth modelling (otherwise, they can be omitted)
    - Failures
    - Delays
    - Filtering
    - Etc.
- Distinction is conceptual, not formal

# *Connection domains*

- Most communication infrastructure can be considered a connection domain
- However, there are **further** properties of distributed systems to take into account
  - Communication is only one of them
  - What about separate memory spaces?
  - Different processing speed?
  - Different environmental conditions?
    - e.g., parts of a distributed system could sit inside the melting reactor of a nuclear power plant, others out of it

# Solving the problem

- R - Requirements: (up to date display)

  $DisplayedValue(patient, factor) = FactorEvidence(patient, factor)$

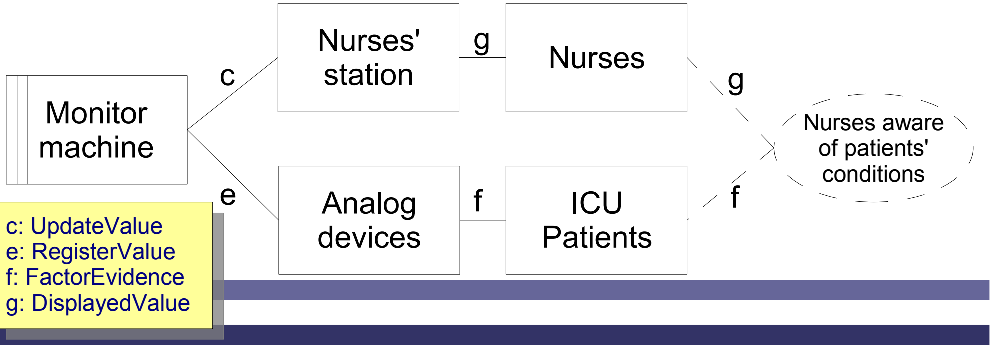
- D - Domains:
  - (Nurses' station is working)

    $MM!UpdateValue(p, f, v) \Rightarrow NS!DisplayedValue(p, f) = v$
  - (Analog devices are working)

    $IP!FactorEvidence(p, f, v) \Rightarrow AD!RegisterValue(p, f, v)$
  - (Nurses are paying attention)

    ...
  - (Patients are attached to the devices)

    ...



# *Solving the problem*

- R - Requirements: (up to date display)

  $DisplayedValue(patient, factor) = FactorEvidence(patient, factor)$

- D - Domains:
  - (Nurses' station is working)

    $MM\,!\,UpdateValue(p,$
  - (Analog devices ar

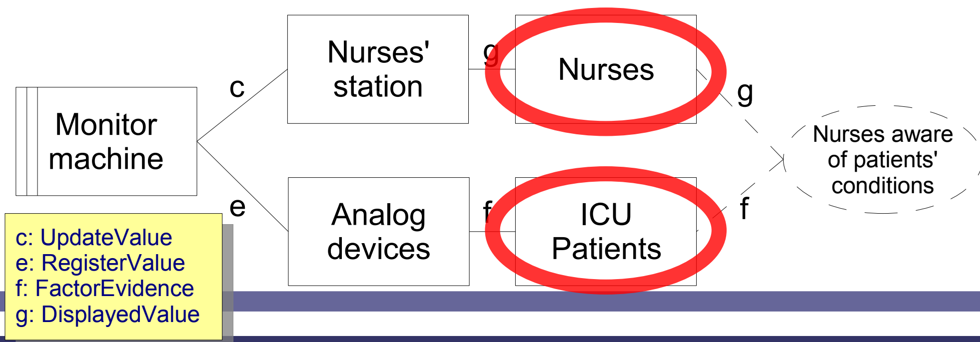
    $IP\,!\,FactorEvidence(p$
  - (Nurses are paying

    ...
  - (Patients are attached to the devices)

    ...

**Human domains**

We have a problem here: human domains are **biddable**, not **causal**, hence we can only hope (and not guarantee) that they will behave as expected



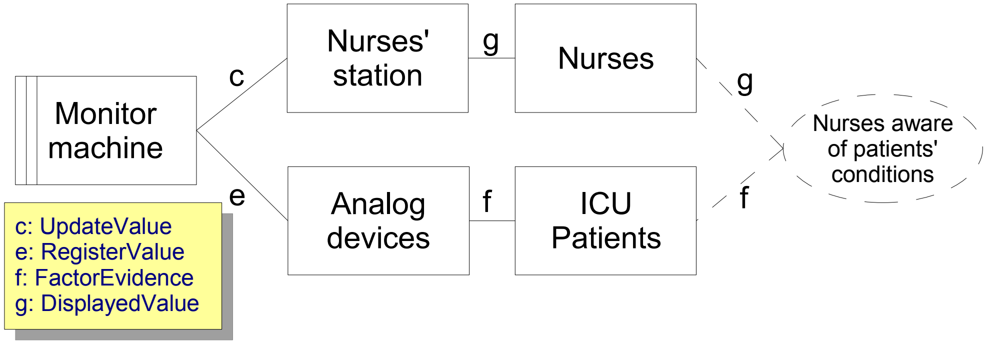
# *(Not) solving the problem*

- Once biddable domains come into the picture (and humans are always, *at most*, biddable), we cannot develop a specification that will guarantee $S \cup D \models R$
- Two alternatives:
  - Renounce solving the problem
  - Develop means to introduce quasi-causal behaviour in a biddable domain
- We will obtain best-effort, approximate satisfaction of the requirements (at most)

# *Causality and almost-causality*

- Notation:
  - $\models$, $\Rightarrow$ = causal entailment, implication
  - $\models$, $\Rightarrow$ = quasi-causal entailment, implication (best effort to make it behave as causal)
- Effectiveness of quasi-causality should be considered explicitly
  - In particular, the **risk** of quasi-causality being broken should be assessed
  - **Mitigation** and **counter-measures** established
    - These would usually add to requirements

# *Causality and quasi-causality*




- Patient p presents a certain (medical) condition $\Rightarrow$ [IP]
  IP!FactorEvidence(p,f,v) $\Rightarrow$ [f]
  AD!RegisterValue(p,f,v) $\Rightarrow$ [e]
  MM!UpdateValue(p,f,v) $\Rightarrow$ [c]
  NS!DisplayedValue(p,f)=v $\Rightarrow$ [g]
  Nurses aware of patient's condition [N]

$$MM \cup N, NS, AD, IP \models R$$

# *Example (quasi-causality)*

- How can we ensure that displaying the data on the NS will cause nurses to know of a patient condition?
- Typical HCI issue:
  - Add a second channel, beyond static display
    - Audio alarm, to be played whenever a value change
    - Blink a new value on screen for the first 5 seconds
    - Implant nurses with a microchip which will give out a moderate electrical shock when a value change
  - Separate physical location from notification
    - Provide all nurses with portable displays, so they don't need to sit at the station

# *Example (quasi-causality)*

- Will such devices solve the problem?
  - No, they can all fail
    - Deaf nurse (so popular among patients!)
    - Nurse distracted, does not look at screen during blink
    - Nurse faints when given electrical shock
    - Portable display's battery exhausted, or device out of reach
- Real world is always *much* more complex than we can model
  - What we can do is to understand the extent of the ***safe bounds*** for our system's operations

# *Example (quasi-causality)*

- Possible mitigations:
  - Louder ring tone, using induction loop, repeated alarms
  - Blink until explicitly acknowledged
  - Make sure electrical shock is not a health hazard
  - Have backup battery on board, signal when main exhausted, or provide better radio coverage
- Possible counter-measures:
  - On evidence of a changed value being ignored for some time, notify someone else through other means (e.g., message doctor)

# *Example (quasi-causality)*

- Possible mi
  - Louder rin
    alarms
  - Blink until
  - Make sure electrical shock is not a health hazard
  - Have backup battery on board, signal when main exhausted, or provide better radio coverage
- Possible counter-measures:
  - On evidence of a changed value being ignored for some time, notify someone else through other means (e.g., message doctor)

**GUIs are only part of the story!**
As shown in the example, interaction may involve multiple channels and behaviours.
Never think in terms of a given GUI toolkit only (unless your problem is a very standard one, for which a known GUI-based interaction pattern is well established)!

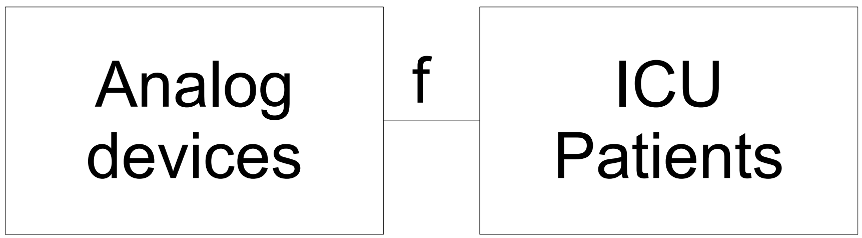
# *Example (quasy-causality)*

- Can we render the communication links quasi-causal as well?
  - Much harder: if a wire is cut, it's hard to pump bits through it
- We can work on the **mitigation** side
  - e.g.: ensure we have a PING or a carrier over the wire
  - So that the system can detect when the connection is broken and alert the nurses
    - OMG... alerting is quasi-causal!

# Exercise

## *biddable domains and quasi-causality*

- What can we tell about how to make the ICU Patient domain quasi-causal?




f: FactorEvidence

# State of practice