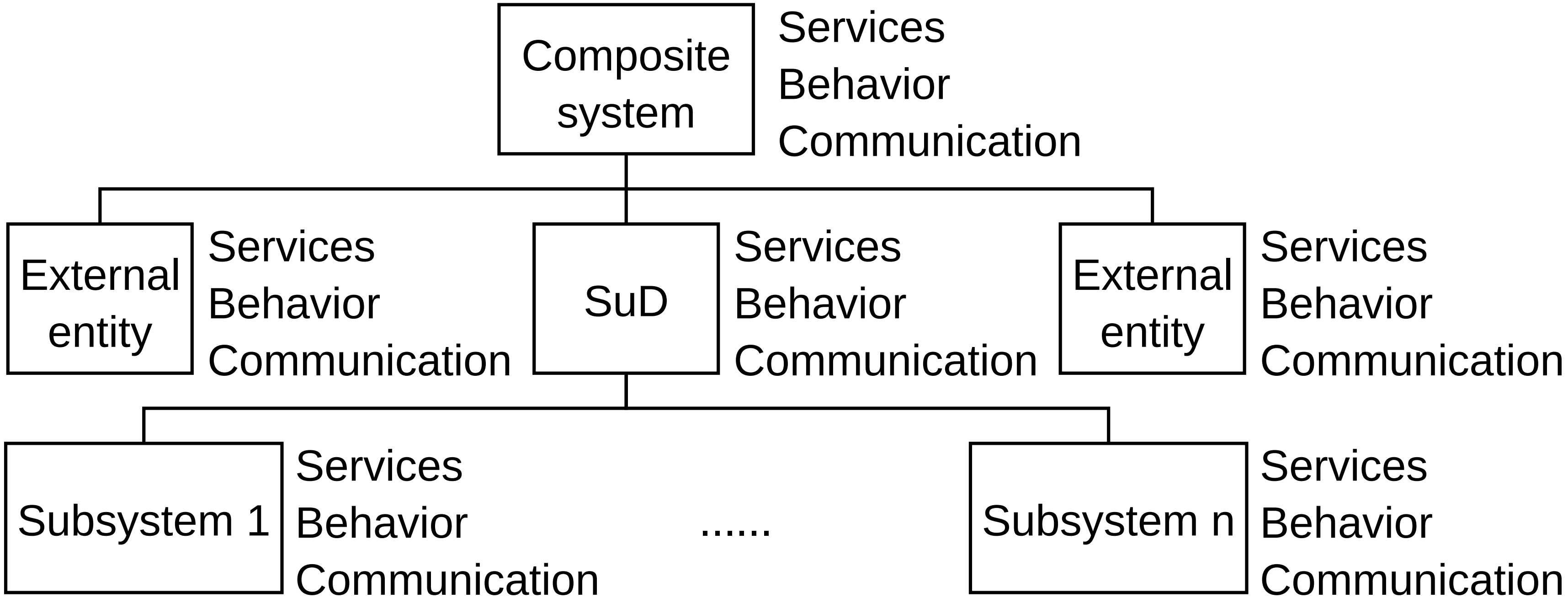
Part V. Communication Notations



Uses of communication notations

Context modeling

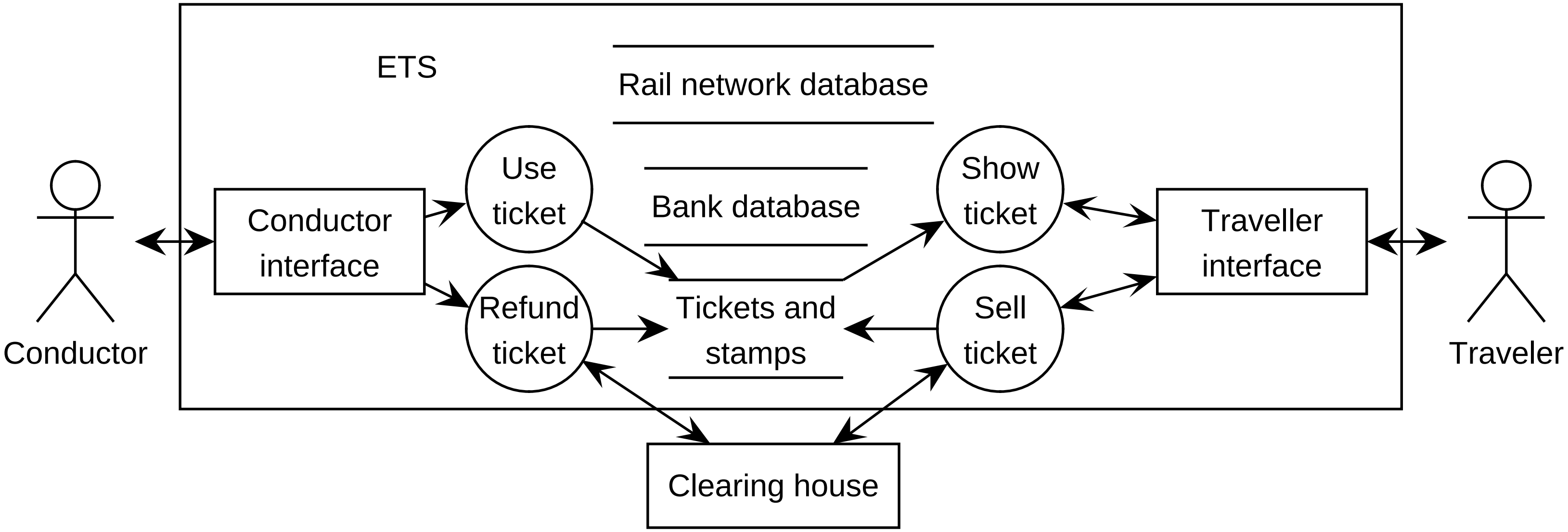
- Communications in environment
- Communications between SuD and environment

Architecture design

- Decomposition of stimulus-response pairs into communication between components
- Communications between components and external entities

Example: Electronic Ticket System (1)

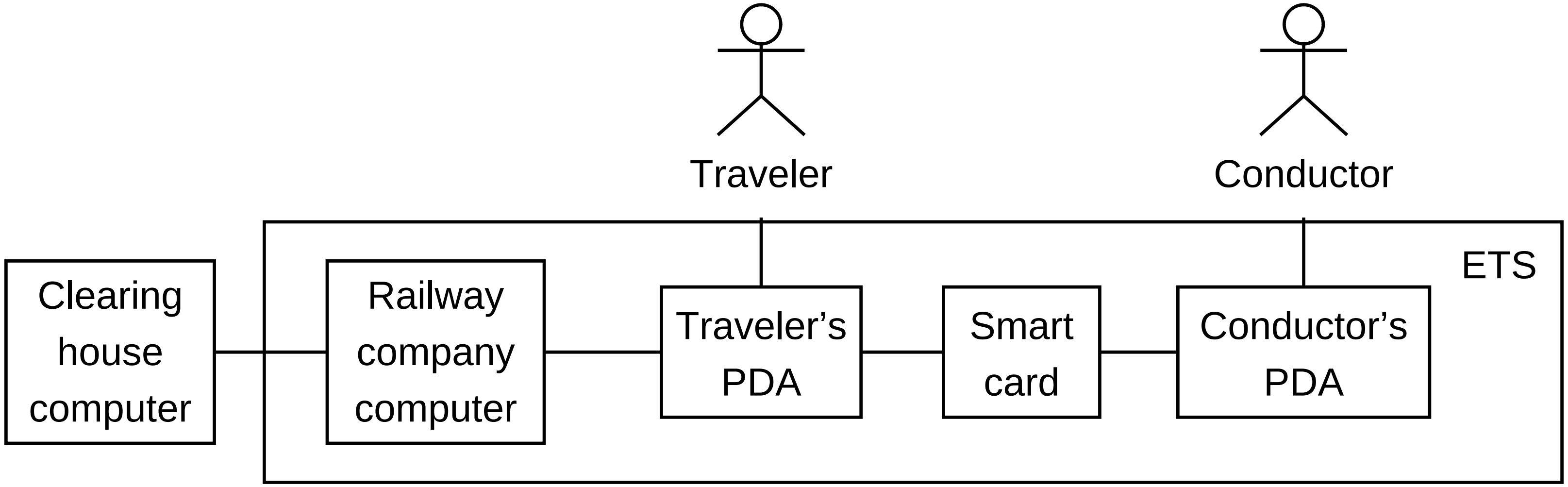
Requirements-level architecture



- Independent from physical architecture
- Independent of software implementation platform
- Motivated only in terms of environment and requirements

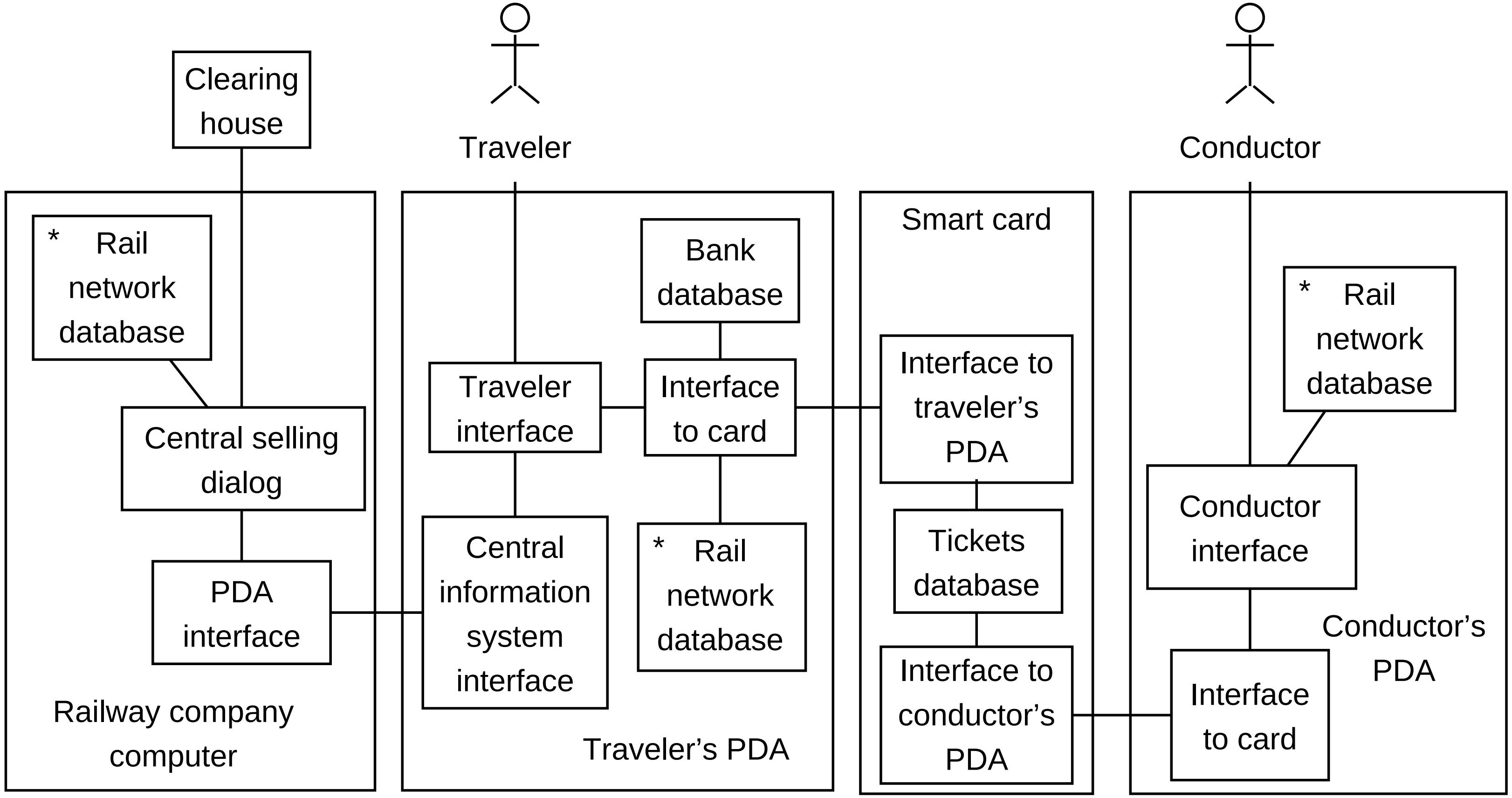
Example: Electronic Ticket System (2)

Physical network architecture



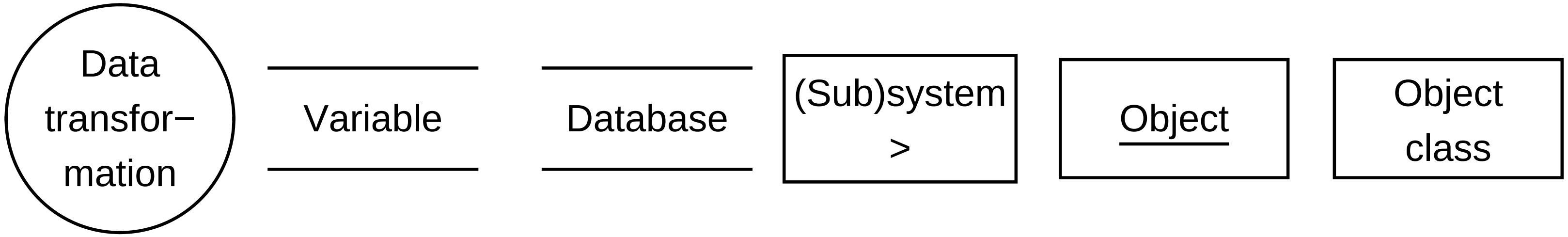
Example: Electronic Ticket System (3)

Allocation of essential components to physical components



Components introduced to deal with internal interfaces. Replicated components indicated by asterisk.

Icons used



- There is a small number of icons used to represent the communication between components in an architecture.
- Data flow diagrams (DFDs): No boxes.
- Architecture diagrams: All icons.

Structure of part IV

- Data flow diagrams (DFDs)
- Architecture diagrams
- Execution semantics
- Context modeling guidelines
- Architecture design guidelines

Main points

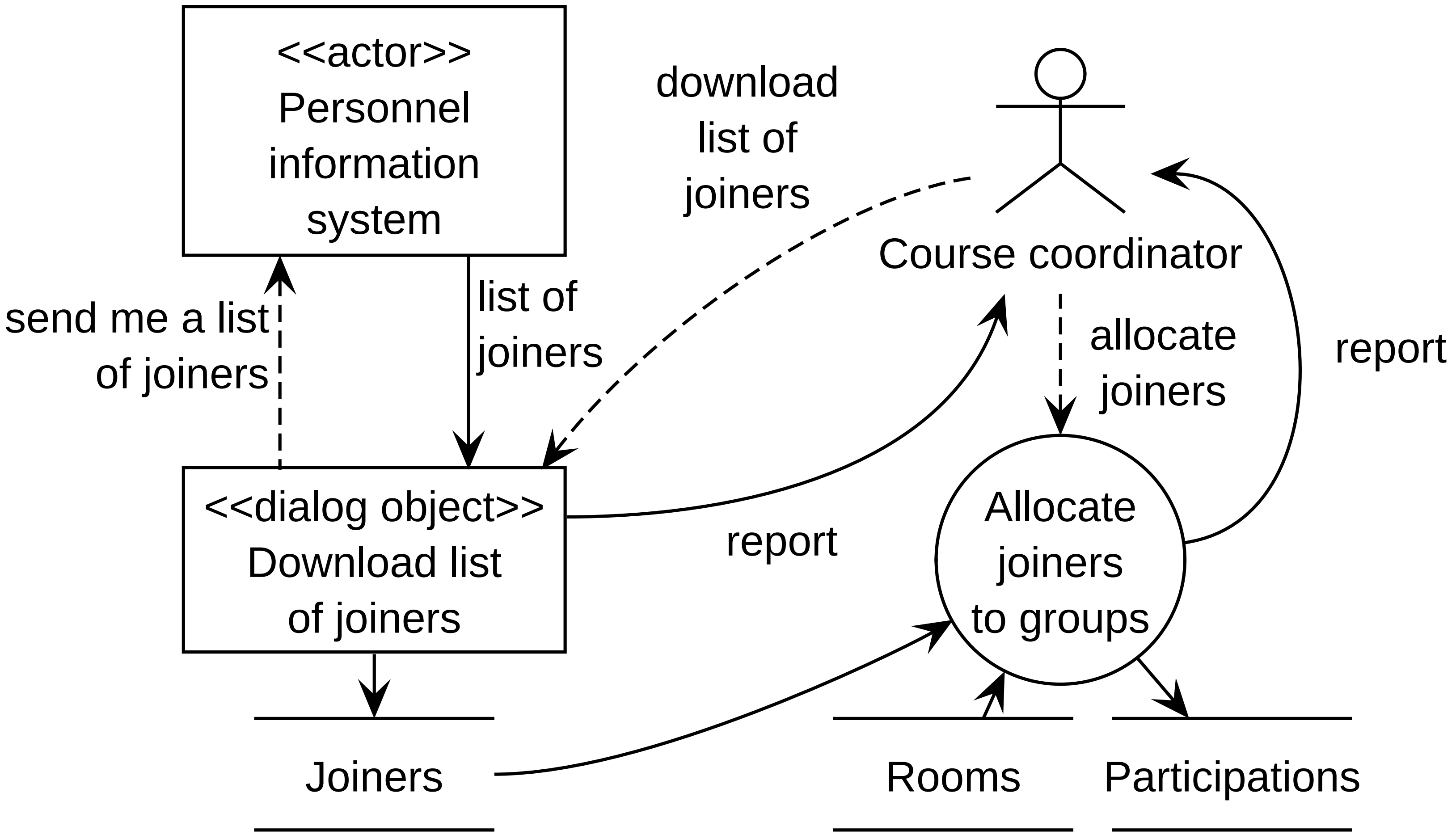
- Communication occurs in the environment and at all levels in the system.
- Find essential system decomposition by
  - Analyzing required external system communication and
  - modeling desired environment communications.
- Communications can be described by DFDs or architecture diagrams.

Chapter 15. Data Flow Diagrams

- Introduced in the 1970s to represent logical software decomposition.
- Still widely used.
- Ontology built into the notation: Software consists of data stores and data transformations.

(Ontology = classification of kinds of things.)

Fragment of architecture of Training Information System

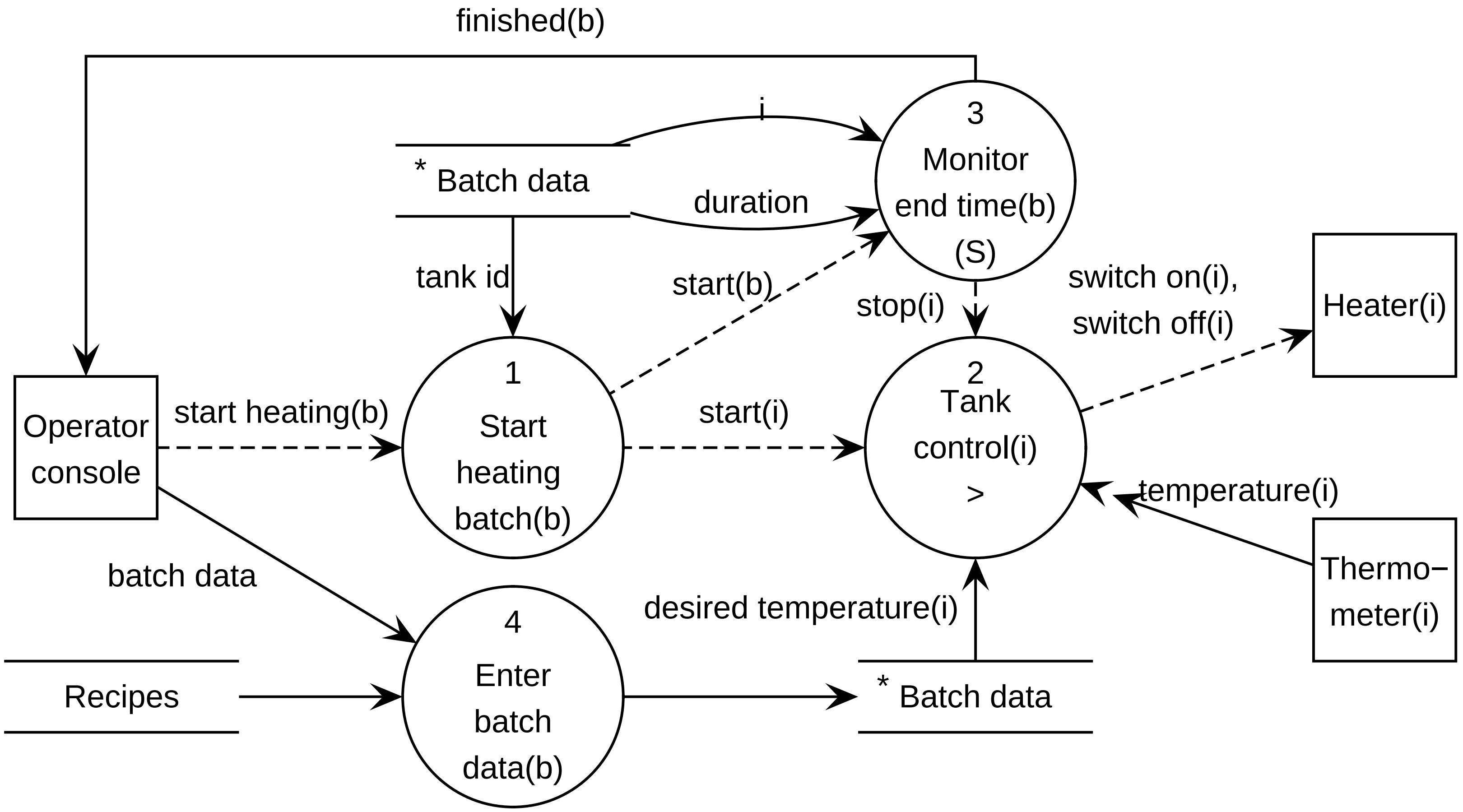


Typical information system architecture

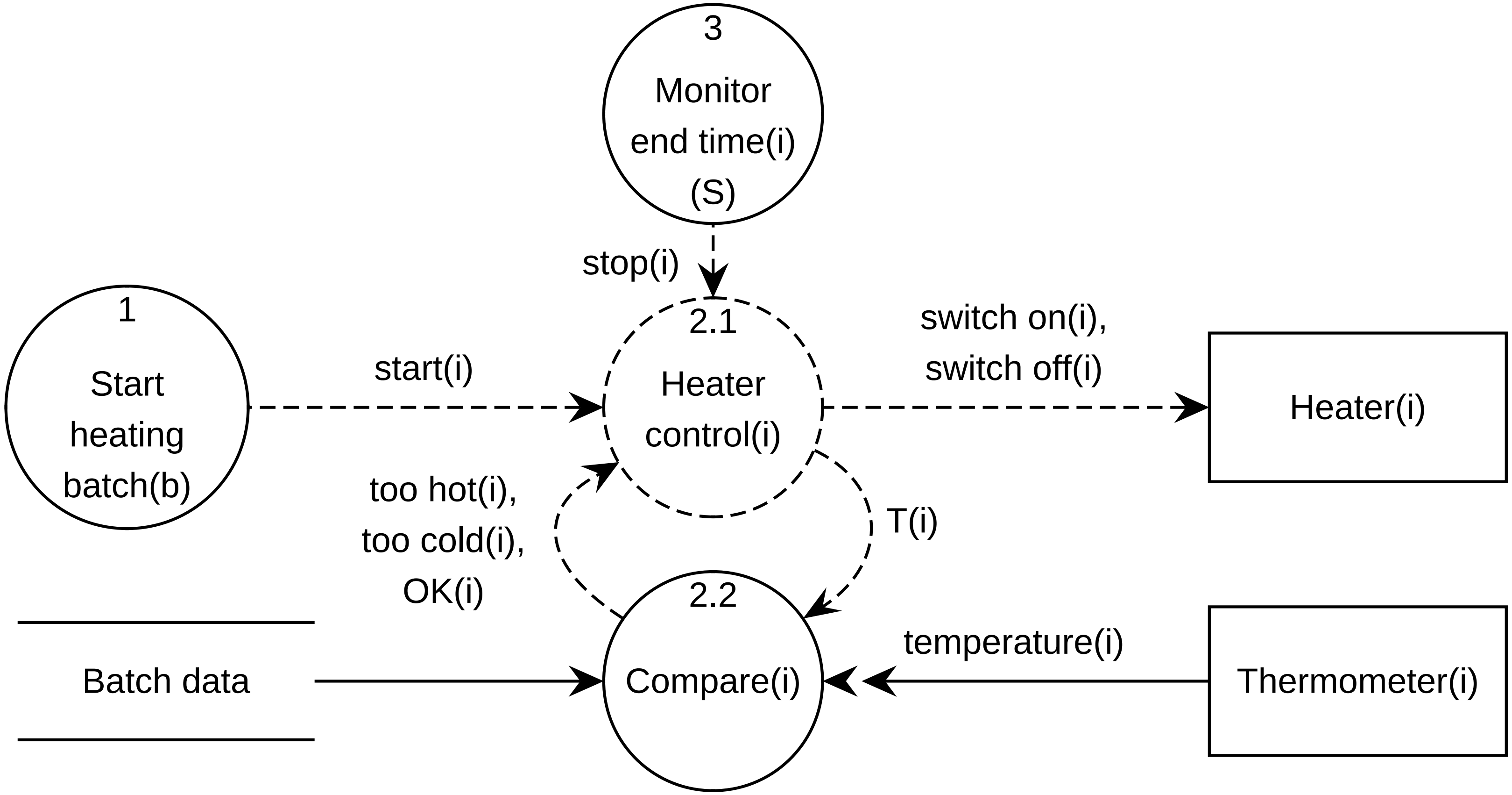
- Collection of data stores holding info about subject domain.
- Collection of data transformations accessing the stores on behalf of external entities.

Often more insightful to present functional DFD fragments separately.

Requirements-level architecture of heating controller



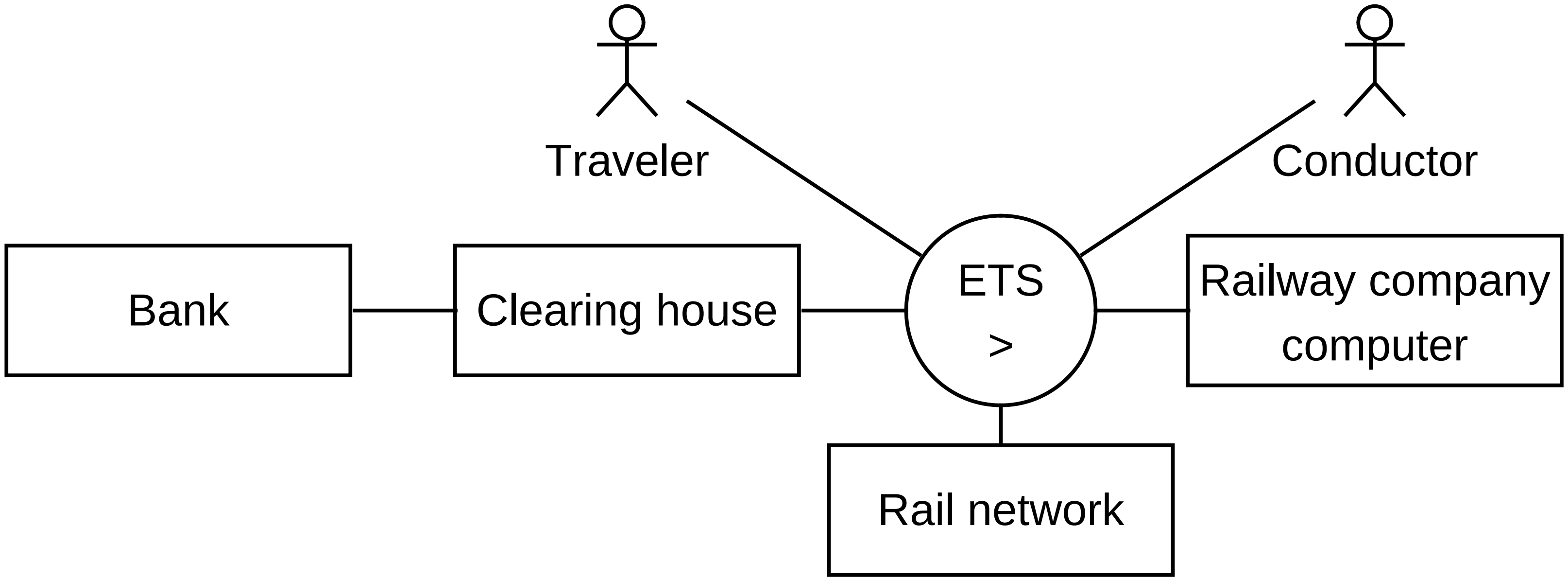
Decomposition of **Tank control(i)**



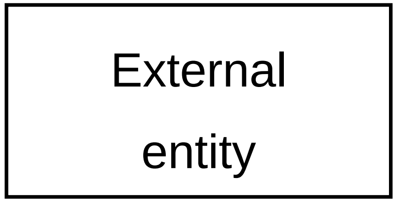
Typical control system architecture

- One or more control processes.
- Collection of interface processes connecting the control processes to external devices.

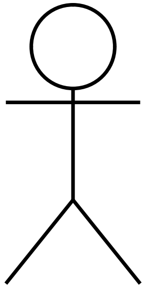
Context diagram of ETS



Icons for external entities

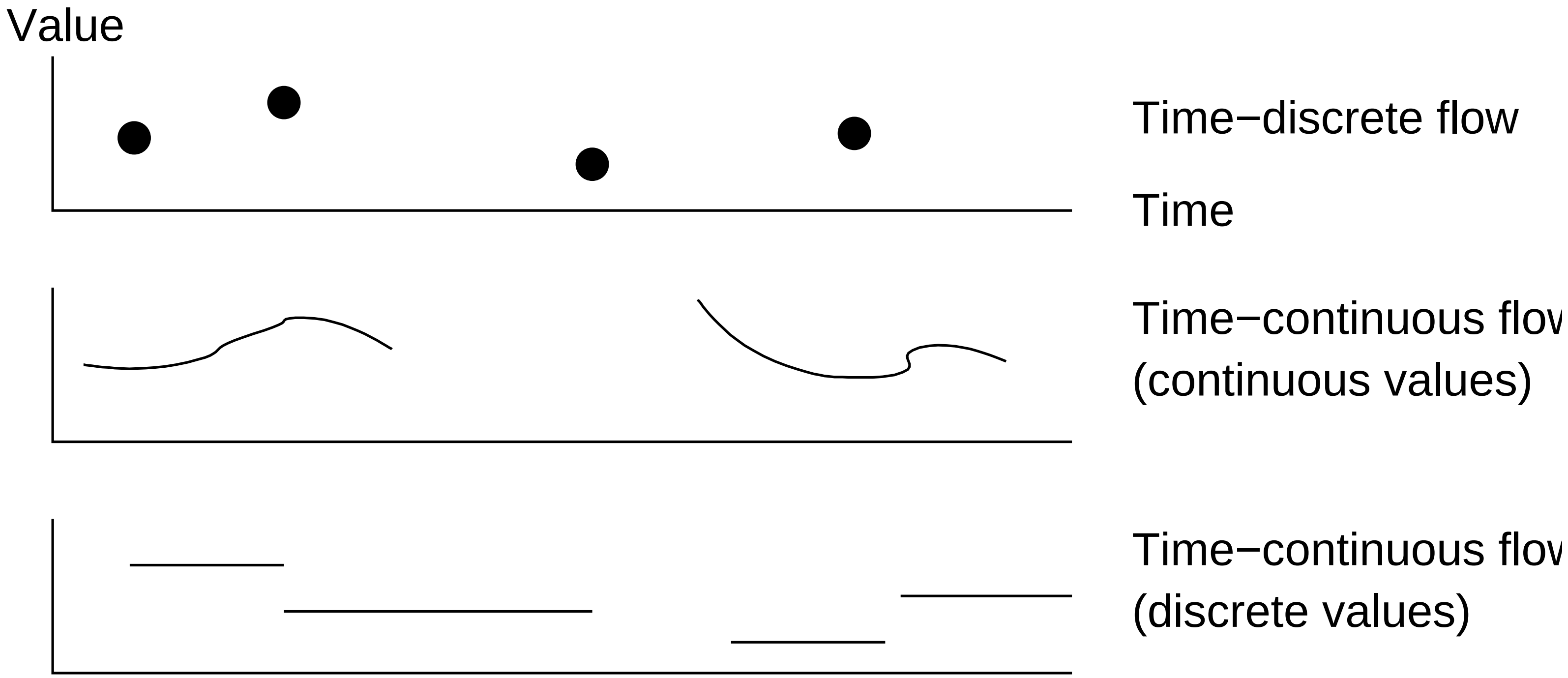


External entity



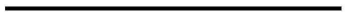
External entity

Time-behavior of flows



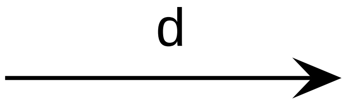
NB. Time-discrete and time-continuous flows all carry data values.

Icons for flows



Communication channel

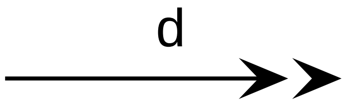
Connected entities can communicate.



Time-discrete data flow

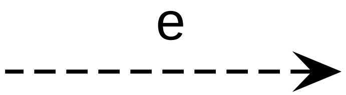
Sender can share info with receiver.

Data is present at discrete instants of time.



Time-continuous data flow

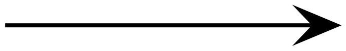
Data present during periods of time



Event flow

Sender can cause receiver to do something.

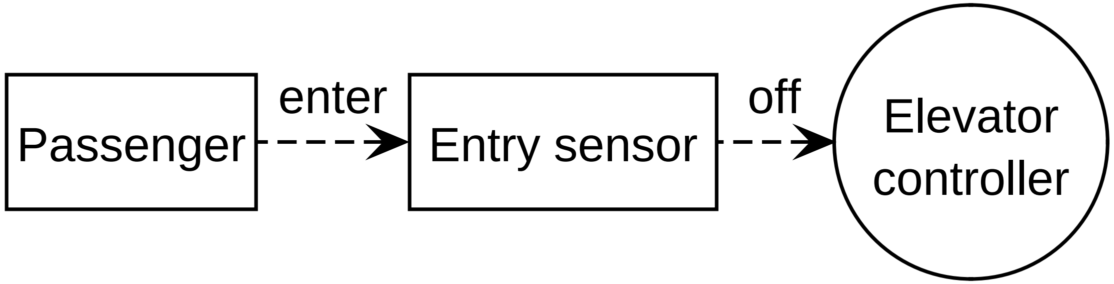
Named after effect or after cause.



Compound flow

Bundles a collection of time-discrete flows.

Event flows

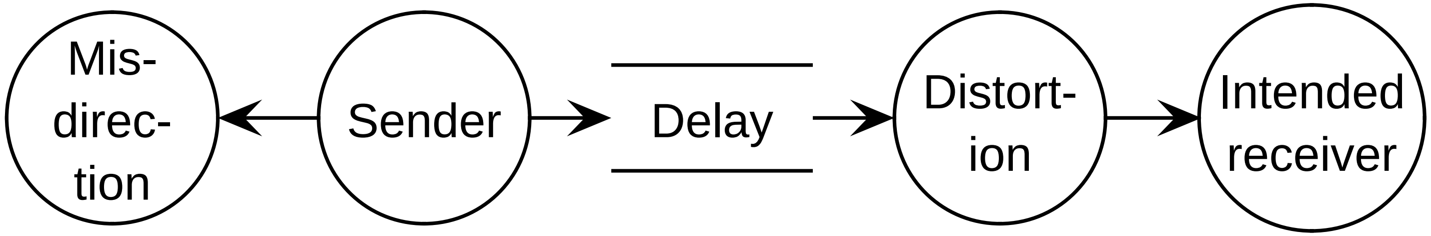


- In our approach, event flows can carry events with parameters.
- Not allowed in Yourdon approaches.
- In our approach, the difference between event flows and data flows is only in the naming.

Communication semantics of flow lines

A **flow** is an instantaneous and reliable communication channel between two elements.

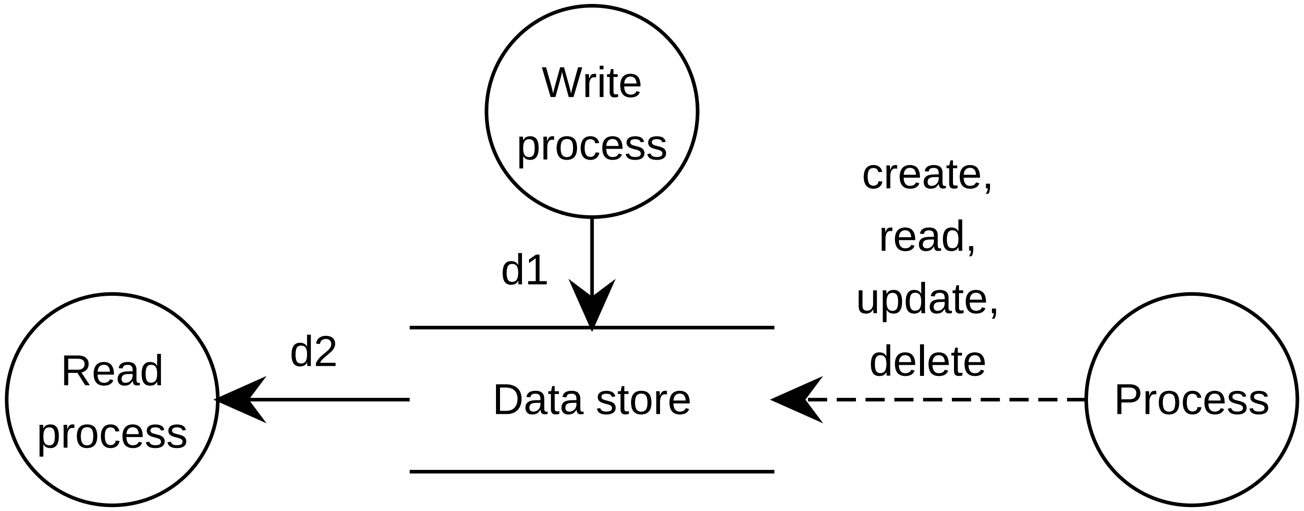
- Introducing delay and unreliability:



- A flow is an abstraction!
- We always must choose *some* abstraction level.

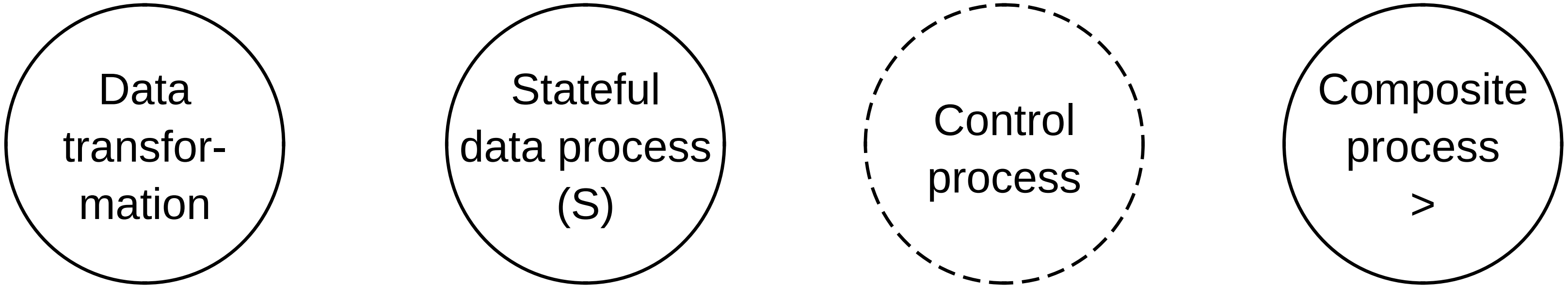
Stores

Remembers data until deleted explicitly.



- NB. Event flow access not allowed in Yourdon methods.
- Conceptual structure of subject domain data is represented by subject domain ERD.

Processes



- **Data process** transforms input data into output data.
  - *Stateless:* Data transformation. Can be triggered by a prompt T.
  - *Stateful:* Can be triggered by an enable/disable prompt E/D.
- **Control process** transforms input events into output events. Specified by an STT or STD.
- **Composite process** is specified by a lower-level DFD.

Specification of a data transformation

2.1: Compare(i).

- When triggered, then let d be the desired batch temperature from the Batch data store:
  - if temperature(i) $<$ d $-$ 5 then too cold(i),
  - if temperature(i) $>$ d $+$ 5 then too hot(i).

- Mathematical input-output relation.
- Interface must match DFD.

Specification of stateful data process

3: Monitor end time.

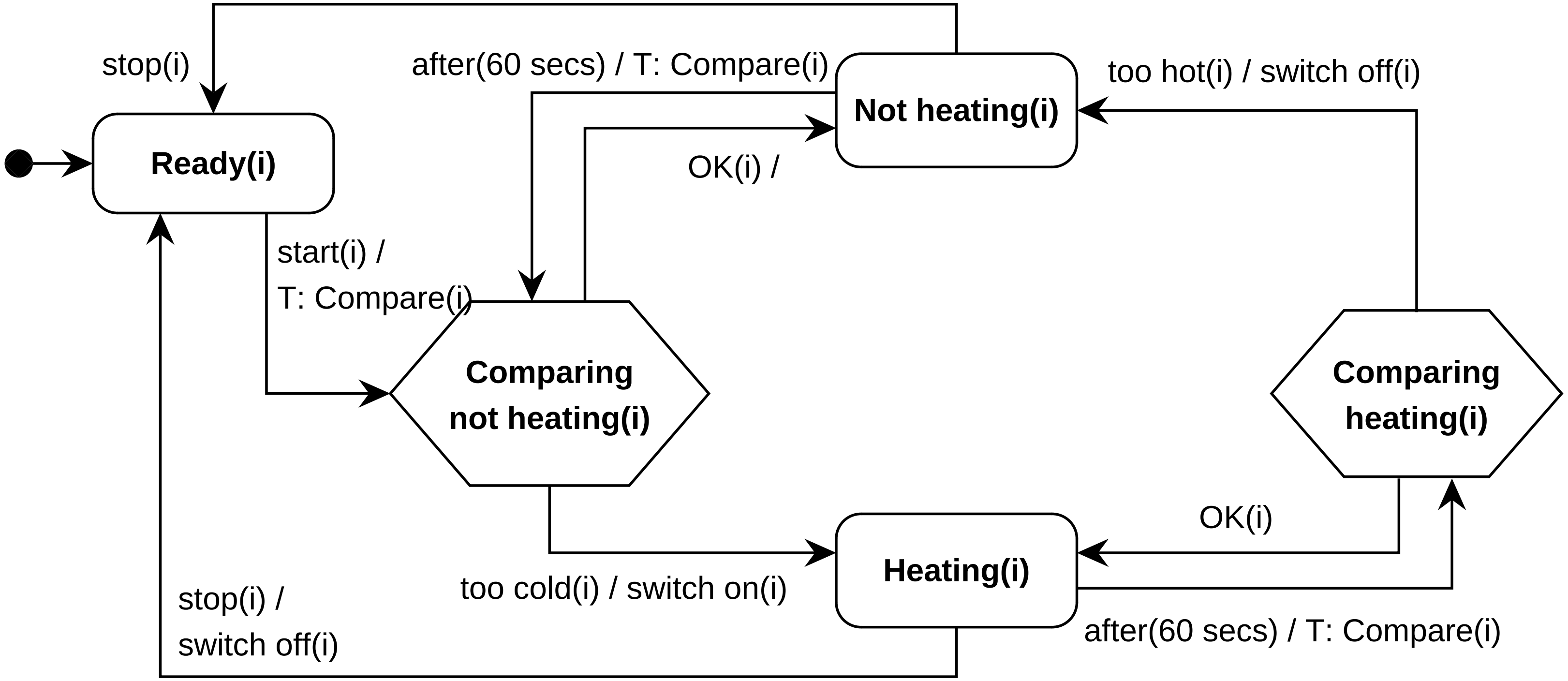
- Local variable: end_time.

  This process is started by the reception of b and then is active until a temporal event occurs.

- Initialization: When b is received, then
  - Read duration of b from Batch data and set end_time := now + duration.
- Process: When end_time occurs, then
  - read tank_id's of b from batch data,
  - for each tank_id do stop(tank_id),
  - send finished(b).

- Initialization is followed by state-dependent transformation process.
- Interface must match DFD.

Specification of control process by Mealy diagram



- Interface must match DFD.
- Yourdon only allows Mealy diagrams. We allow any STT or STD.

Parametrized DFDs

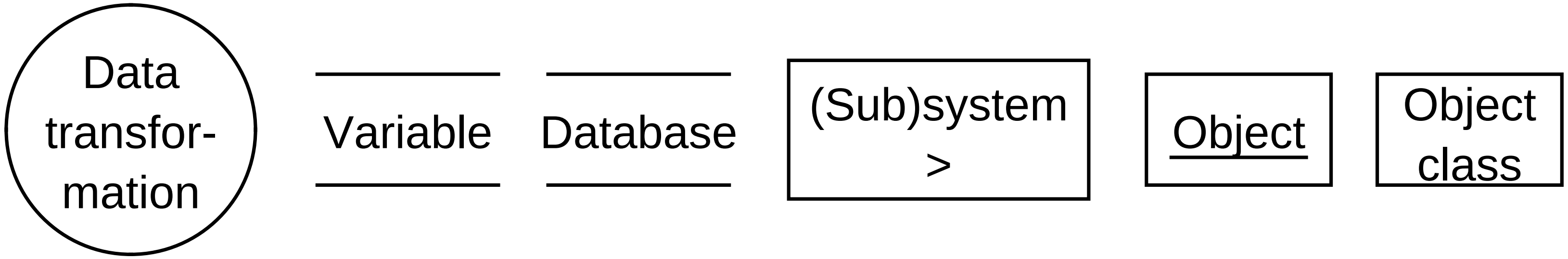
DFD is instance-level diagram. To simulate types of elements:

- Process names in a diagram can be parametrized by a process identifier.
- This causes some flow names to contain process identifiers as well.

Main points

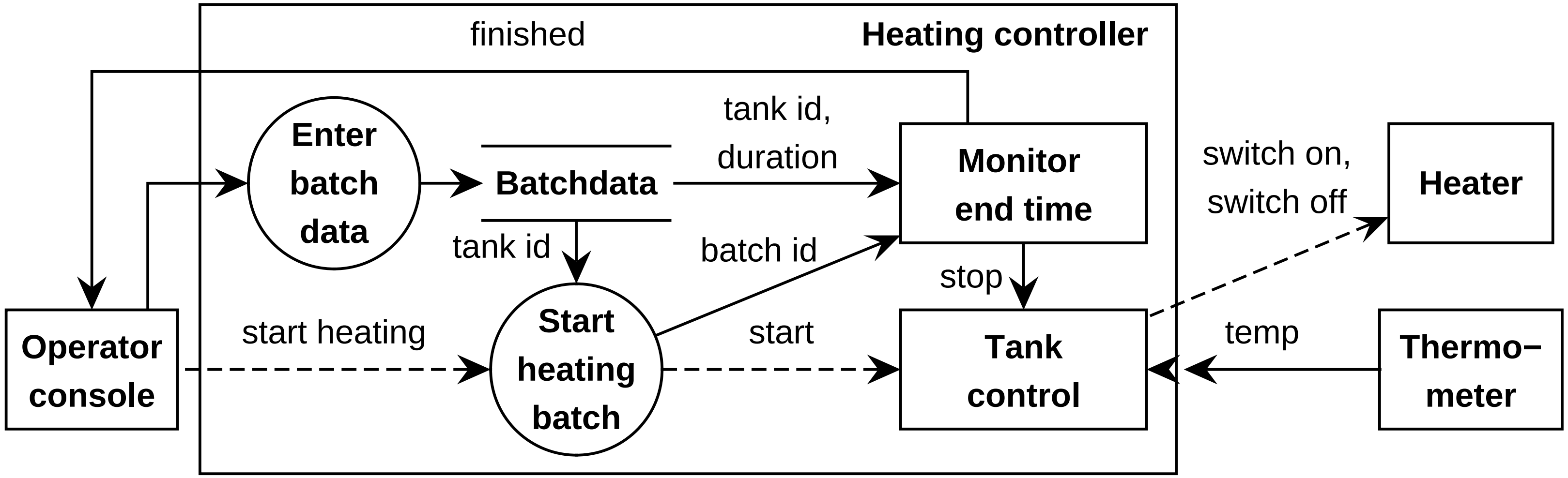
- DFDs model a system as a collection of communicating data stores and processes.
- Various kinds of processes, depending upon how they are specified.
- DFDs can be hierarchical.
- Detailed information about flows can be expressed.
- Instance-level notation.

Chapter 16. Communication Diagrams

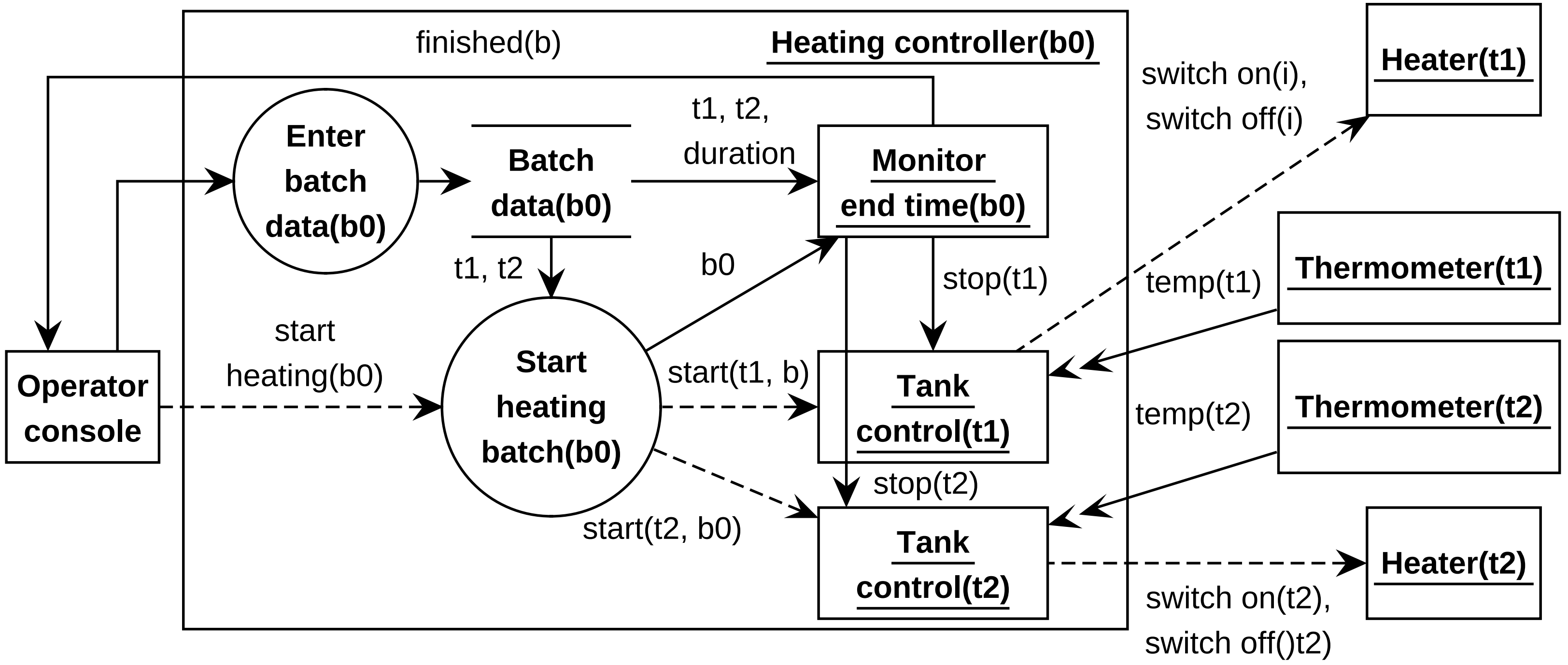


- Difference with DFD: Type-level diagram; more icons.
- Difference between variable and database is that database is (viewed as) a set of instances.
- An object is a subsystem not decomposed by a communication diagram.
- Object classes are not components of a system. They are types of such components.

Communication diagram of heating controller requirements-level architecture



Instance diagram of a particular heating controller in a particular context



Components

Very overloaded term. For us: component = part of an executing SuD that delivers service to its environment.

- Data transformation
- Data store
  - Variable
  - Database
- Subsystem
- Object

To deal with a time-varying collection of components, we can also show object classes. These are not components themselves. (They are *types* of components.)

Communication channels

Same as in DFDs.

- Event channel. Named after cause or effect.
- Data channel. Time-discrete or time-continuous.

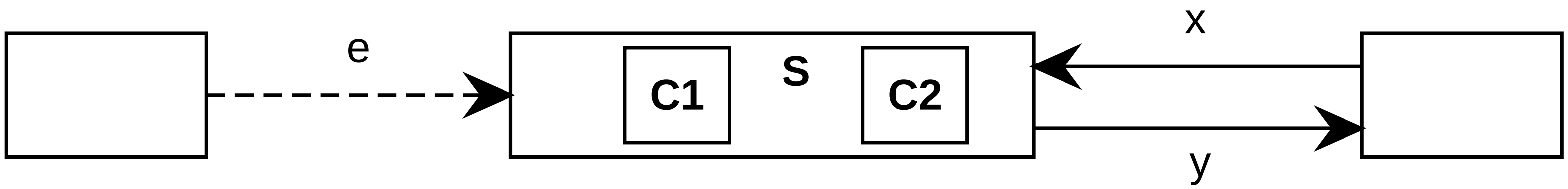
There are two kinds of addressing.

- **Channel addressing.** Item is sent to channel. Used in DFDs.
- **Destination addressing.** Item is sent to individual destinations. Used in the UML.

(De)composition

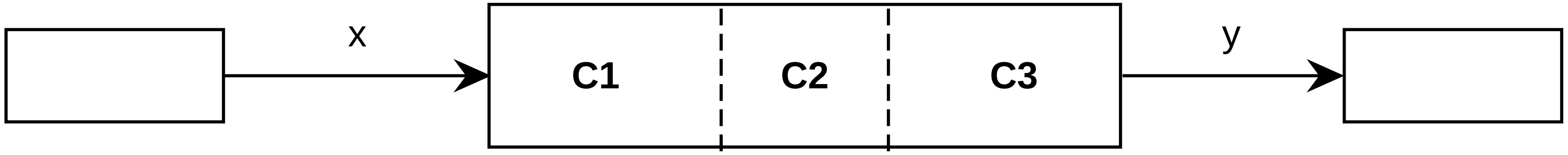
Represented by

- containment of nodes
- or simply specifying the component elsewhere.



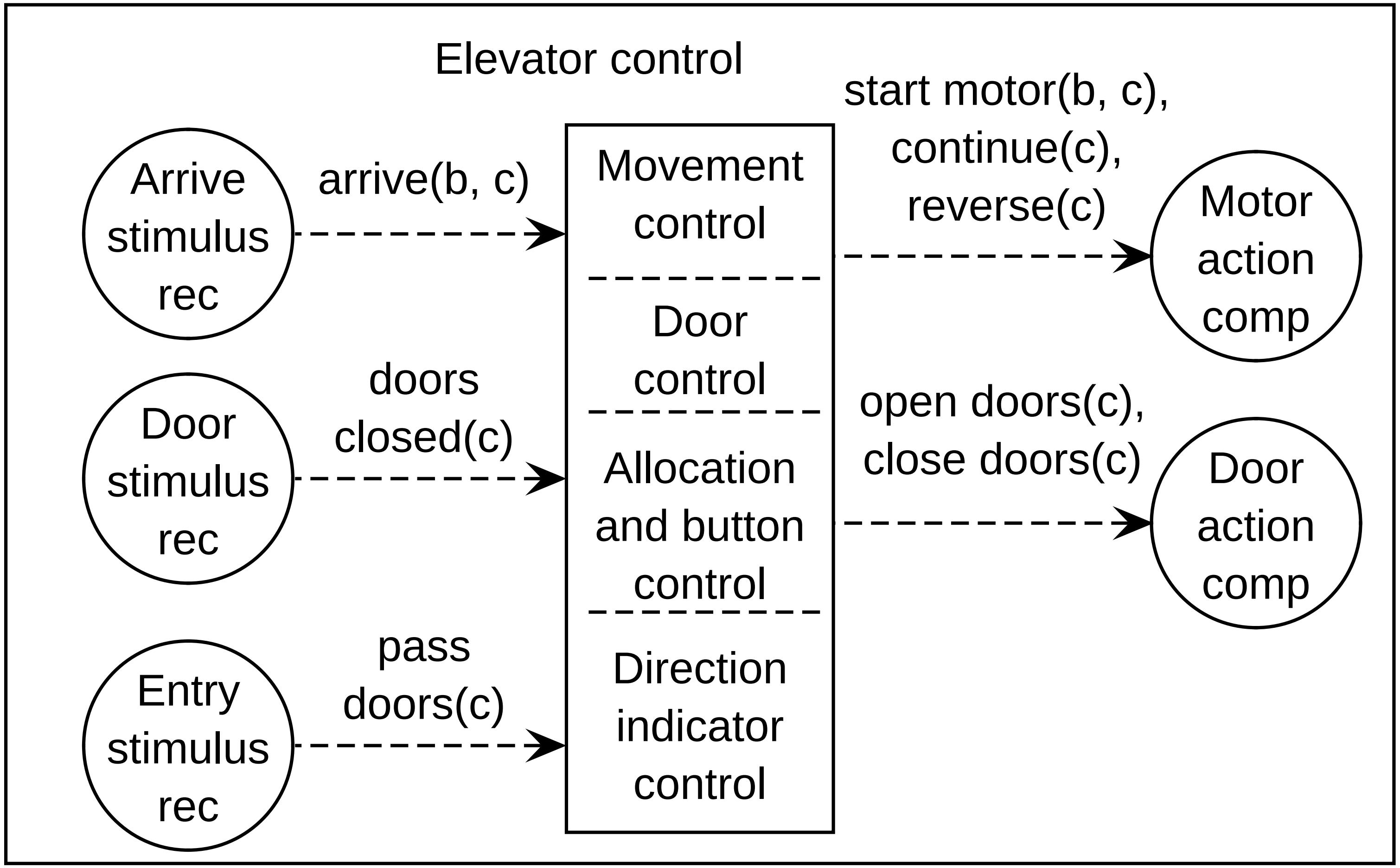
- C1 and C2 can both be triggered by an occurrence of event e.
- C1 and C2 can both read the value of x.
- C1 and C2 can both write a value to y.

Closely coupled components



- C1, C2 and C3 have same interface.
- Events generated in one component are sensed by the other closely coupled components. (Event broadcasting)
- The state of any closely coupled component is readable to any other closely coupled component. This permits the use of in(State).

Elevator controller composition fragment



Arrows show possible interfaces of all four closely coupled components.

Allocation of services to components

Components	**Functions**				
	Create batch data	Start heating	Switch on heater	Switch off heater	Finish heating
Enter batch data	X				
Batch data	X	X			
Start heating		X			
Tank control			X	X	
Monitor end time					X

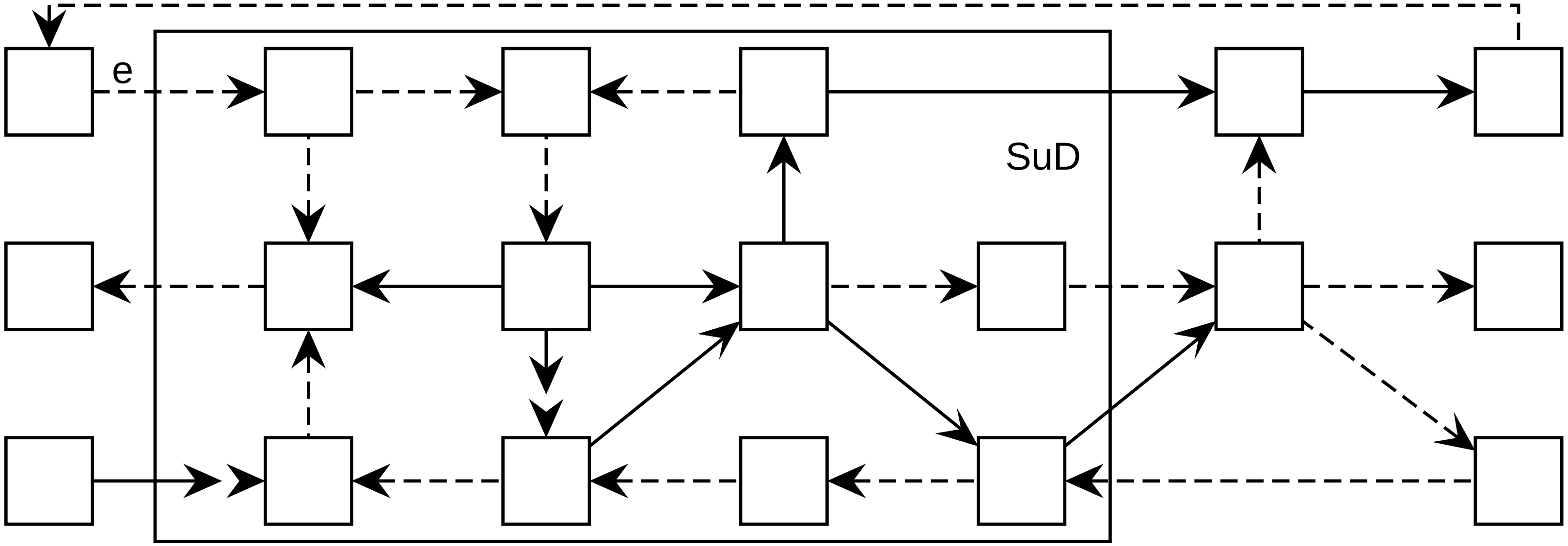
Flowdown

Components	**Functions**				
	Create batch data	Start heating	Switch on heater	Switch off heater	Finish heating
Enter batch data	Accept and store data.				
Batch data	Store data	Provide data			
Start heating		Start heating			
Tank control			Switch on when too cold	Switch off when too hot	
Monitor end time					Stop when time is up.

Main points

- Communication diagrams generalize DFDs.
- They allow us to represent objects and their classes in the diagram, and to represent close coupling.
- Instance-level diagram represents snapshot of the system.
- Decomposition can be represented by containment of nodes.
- We must allocate and flow down system services to component services.

Chapter 17. Communication Semantics



What is the response to e?

- Depends upon the behavioral semantics of the components.
- We assume the semantics of component specifications is fixed, and concentrate on the remaining choices.

Input buffers: problem and possible solutions

Most combinations of semantic choices require a component to deal with a backlog of events that occurred but have not yet been responded to. Suppose components have **input buffers.**

- Size of buffer?
- Structure of buffer? (Set, bag, queue)
- Removal policy. (Even for queues, i.e. deferred events.)

Statemate: set of unlimited size.

UML: Queue (more or less) of unlimited size.

Component input and output: problem and possible solutions

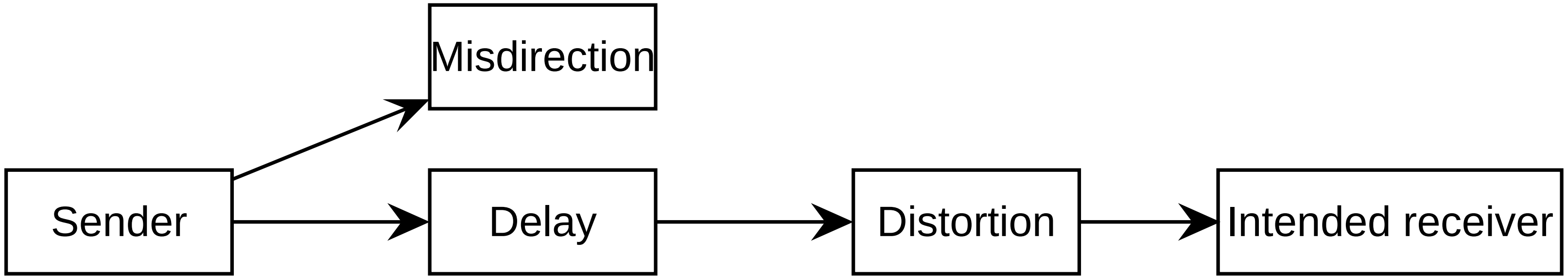
Computations during a step may need more input, or may produce output.

- **Eager read/lazy write:** All input values are read eagerly, at the start of the step, and all output (actions and data) is written lazily, at the end of the step. Statemate.
- **Lazy read/eager write:** Input is read lazily, when needed, and output is written eagerly, immediately when available. UML.

Semantics of communication channels

- There is no delay between sending and receiving,
- A message always arrives at its destination(s),
- It arrives at no other destinations and
- A message is never distorted.

We can always introduce these things explicitly:



Addressing mechanisms

- **Channel addressing.** “Deliver this communication to all components at the other side of this channel.” DFDs and Statemate. Good for broadcast, bad for point-to-point communication; good for encapsulation.
- **Destination addressing.** “Deliver this message to this component.” Good for point-to-point, bad for encapsulation. UML.

Channel capacity: semantic options

- **Zero channel capacity:** Message immediately arrives at receiver. If the receiver has no input buffer, a channel represents a **synchronous communication** in which the sender cannot put an item in a channel if the receiver does not take it out at the same time.
- ***n*-item channel capacity:** A channel can contain n items at the same time. When a sender wants to write something to a channel that is full:
  - **Overwrite semantics.** (Statemate and UML)
  - **Refusal semantics.**

Blocking: semantic options

What happens when for some reason, a sender attempts a communication through a channel, but this communication is not possible?

- **No blocking:** Discard immediately.
- **Finite blocking:** Wait a finite time, then discard. else.
- **Infinite blocking:** Wait.

Statemate and UML: Not applicable, because they use overwrite semantics.

Temporal semantics of network behavior

The SuD is a network of communication components.

Options for temporal semantics:

- Global time is indicated by global clock, immediately accessible to all components.
- Local clocks may differ.

Step semantics of network behavior

- **Network step semantics.** The SuD is ready to respond when any of its components is ready.
- **Network multistep semantics.** The SuD is ready to respond when all its components are ready.

Propagation strategy:

- **Breadth-first:** Execute a triggered component after all components triggered earlier, are executed.
- **Depth-first:** Execute a triggered component immediately.

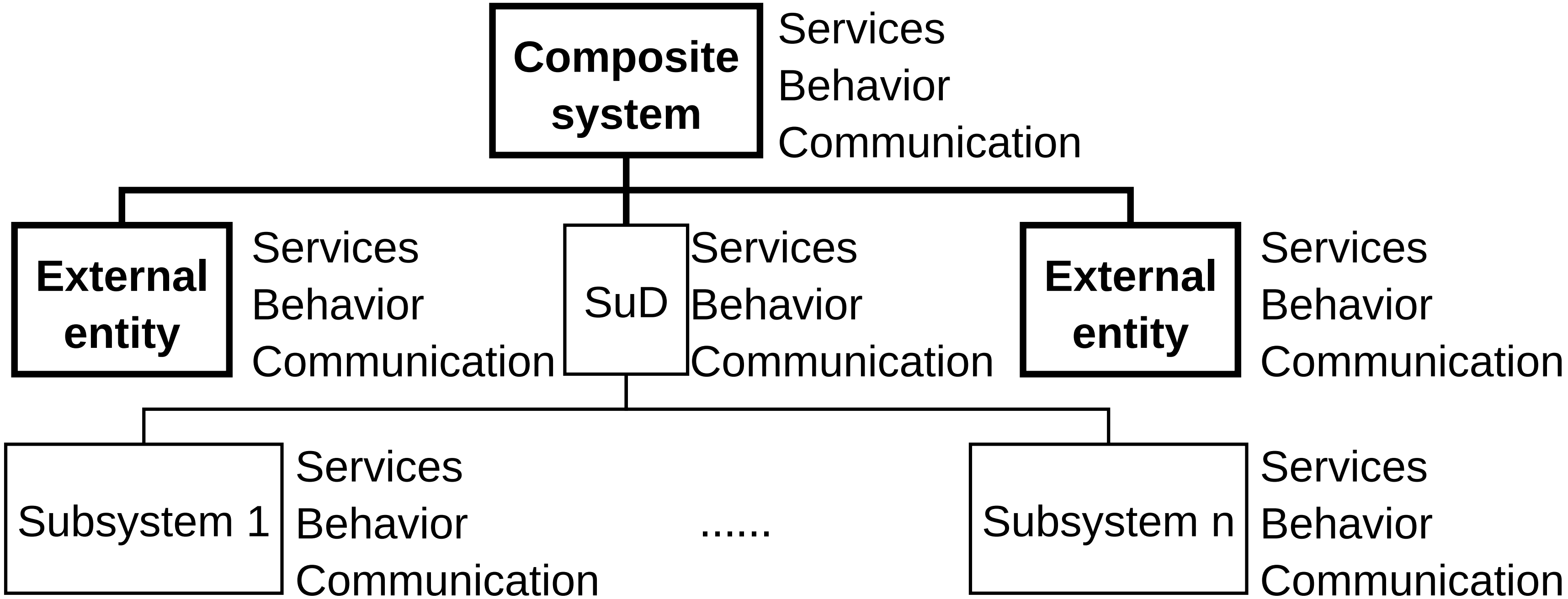
The Environment

- A communication diagram is not intended to specify environment behavior. But it does make some assumptions about environment behavior.
  - Environment is able to absorb response at all times.
  - The environment can produce stimuli at all times.
- The environment is continuous, the system is discrete.
  - Arrival order of stimuli over different channels may be unknown by the SuD.
  - Stimuli arriving over same channel may be lost.
  - Time-continuous input is sampled.

Main points

- Components may have input buffers; may have size restrictions and access policies.
- Communication channels are reliable and instantaneous.
- Addressing may be by channel or by destination.
- Channel capacity may be restricted.
- Communications have a blocking semantics.
- Time may be global or local.
- Steps may propagate through the network breadth-first or depth-first.
- We assume that the environment is always able to absorb responses.

Chapter 18. Context Modeling Guidelines



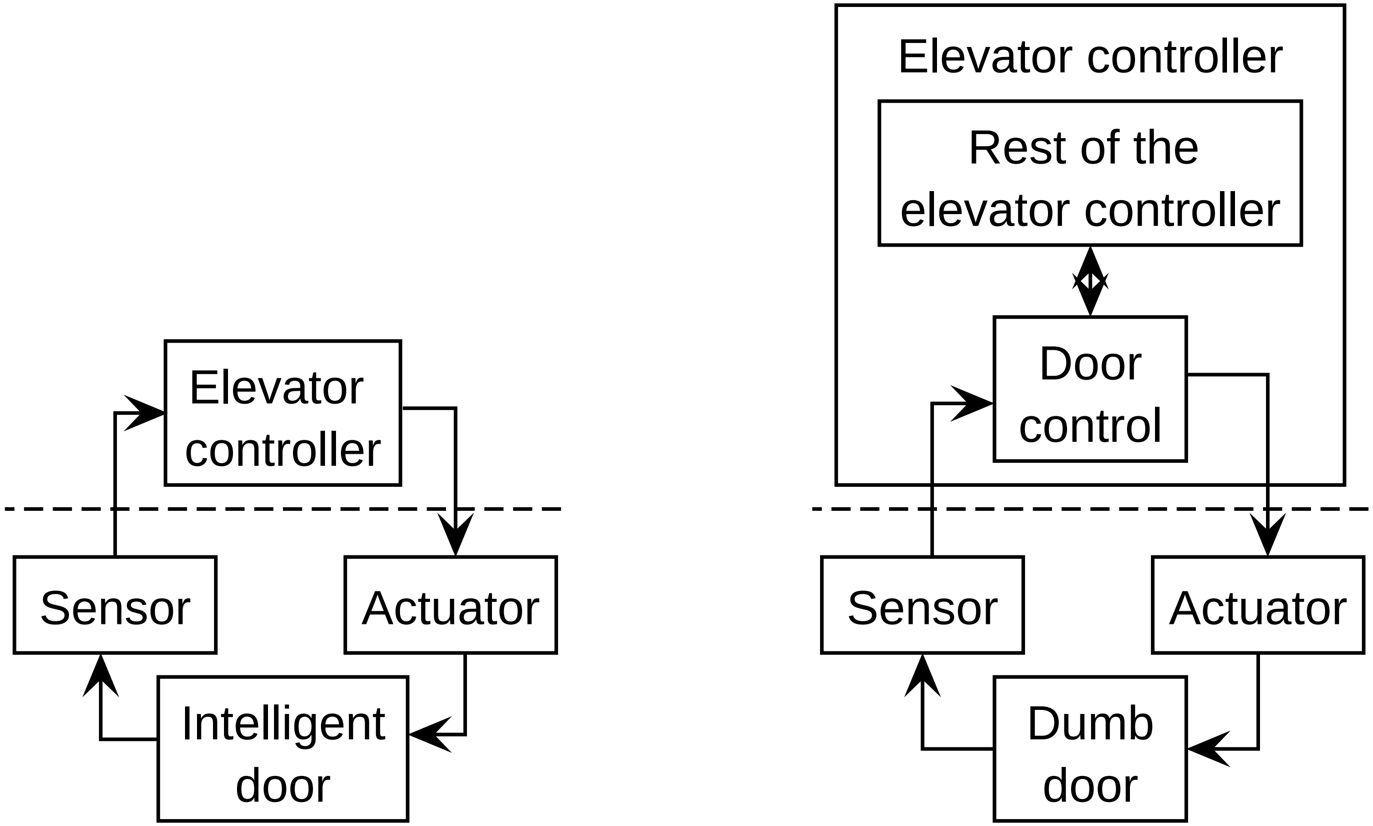
- To prepare for making design decisions, you first collect information about things given to you.
- This is modeling, not design.
- Consider system as black box.

Where is the system boundary?

- √ Use function refinement tree. This is a fully implementation-independent description of system boundary.
- √ Use list of stimuli and responses. This is a behavior description of the system boundary.

Tradeoff between environment and SuD

- System engineering argument: A and S entail E.
- Responsibility for emergent properties is distributed over environment (A) and system (S).



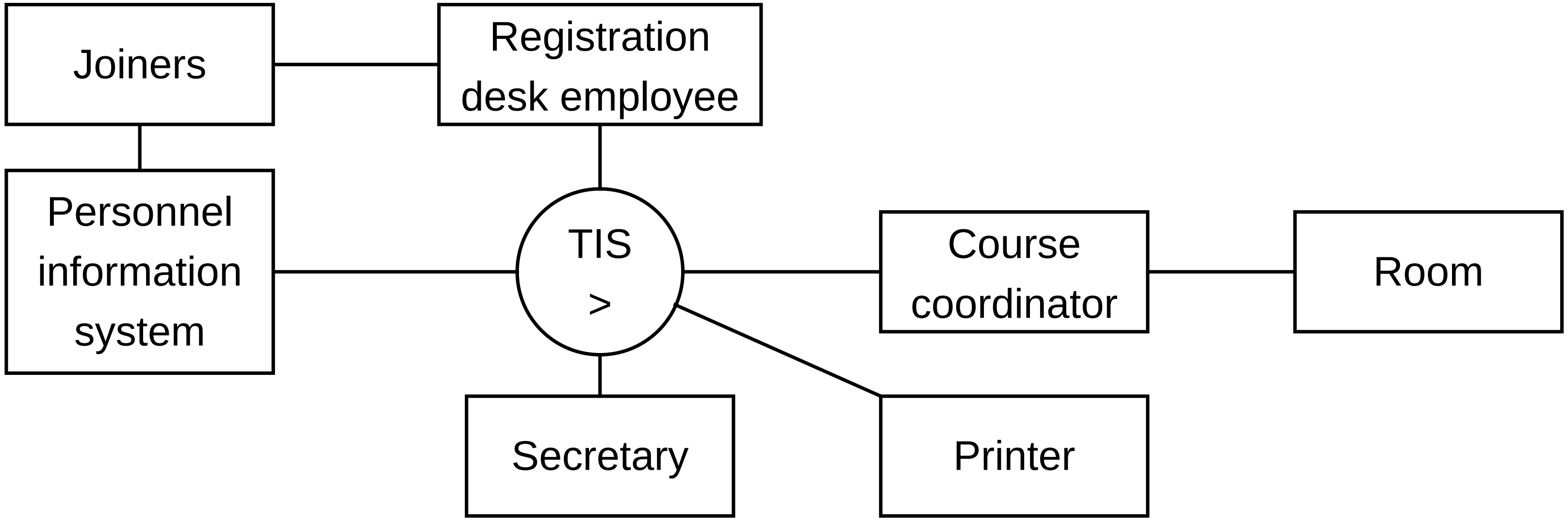
Context diagram

Represents communication among the relevant entities in the environment and the SuD.

- √ External entities are
  - — Physical entities (people, devices, natural objects),
  - — Conceptual entities (organizations, obligations, rights),
  - — Lexical entities (software, contracts, specifications).

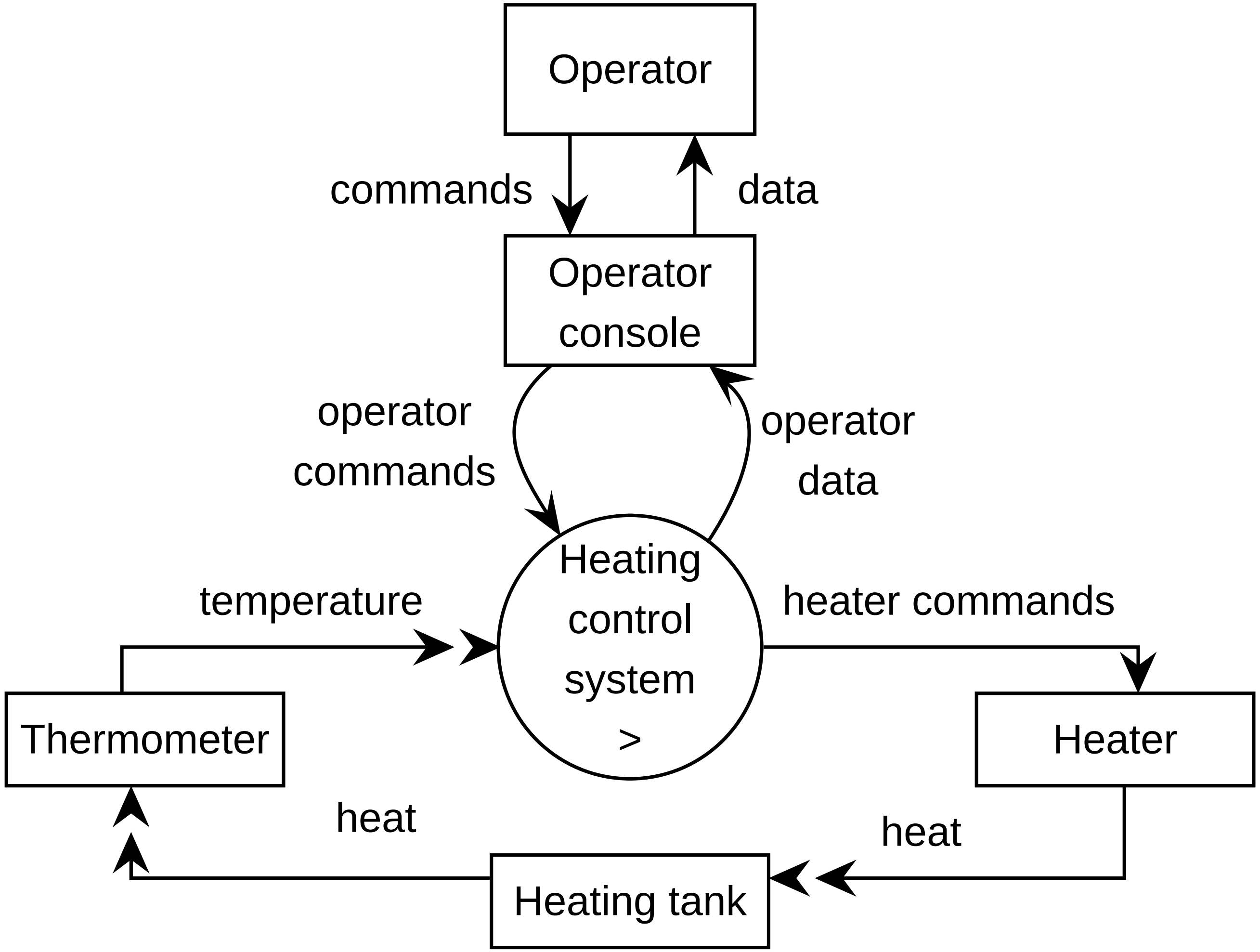
A context diagram must give overview; it should link clearly to the purpose of the system.

Teaching information system



Not much detail needed here.

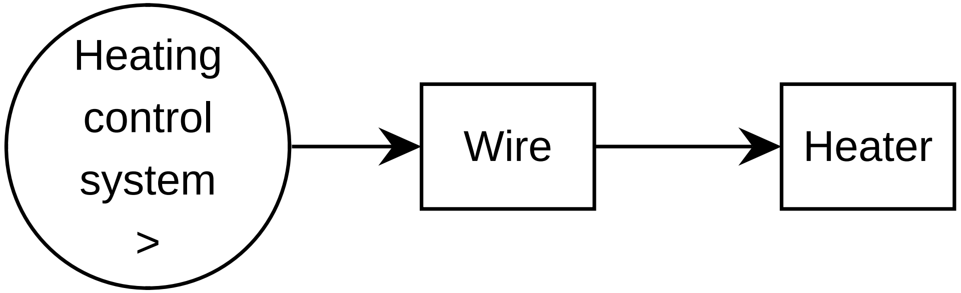
Heating controller



More detail needed here.

Choose an abstraction level

√ Communication channels represent a level of abstraction.



There is always a remaining level of abstraction.

Context boundary

- √ Include entities needed to achieve goals of composite system.
- √ Include entities in which SuD causes desired effect.
- √ Include entities about which SuD needs information to perform its work.
- √ Ignore entities where effect is not felt / not relevant.
- √ Ignore entities whose behavior is irrelevant for the task of the SuD.

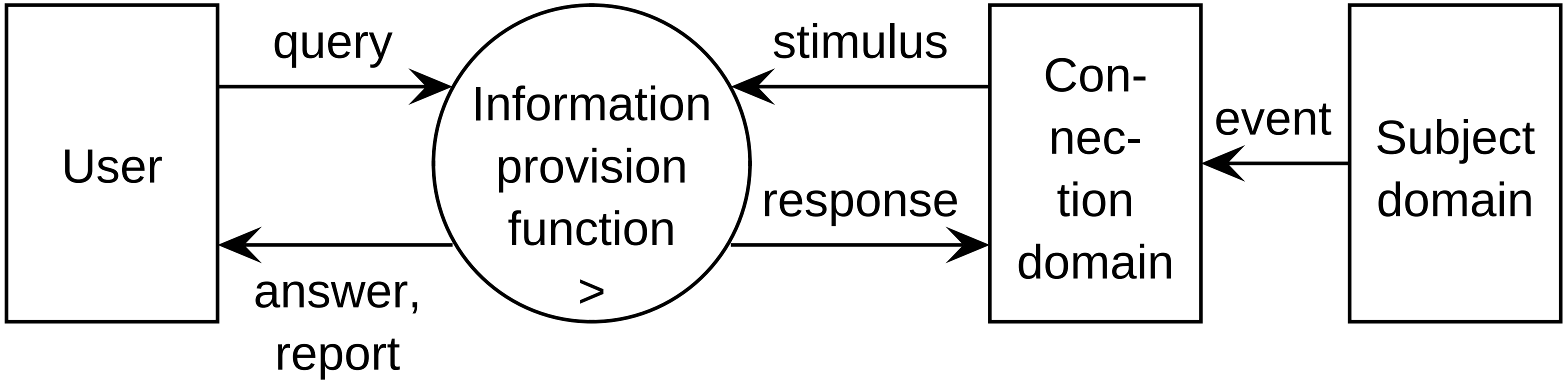
The context diagram shows which external entities are, together with the SuD, involved in achieving the desired emergent properties of the composite system.

Structuring the context

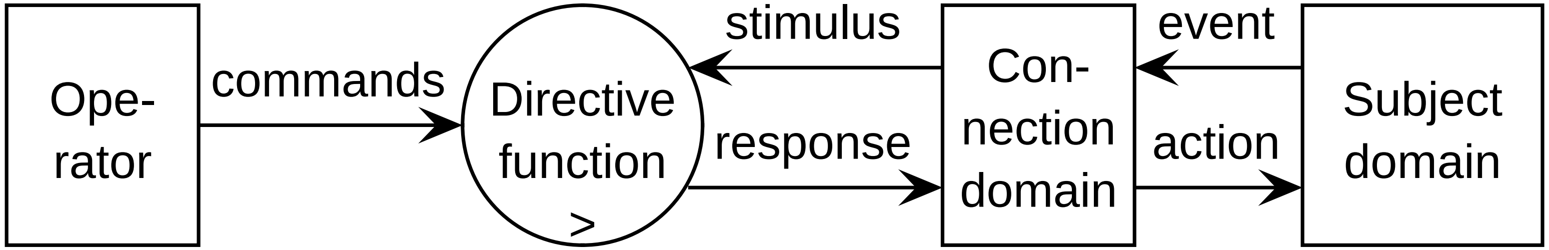
Three kinds of functionality:

- *Information provision.* To answer questions, produce reports, or otherwise provide information about the subject domain.
- *Direction.* To control, guide, or otherwise direct its subject domain.
- *Manipulation.* To create, change, display, or otherwise manipulate lexical items in the subject domain.

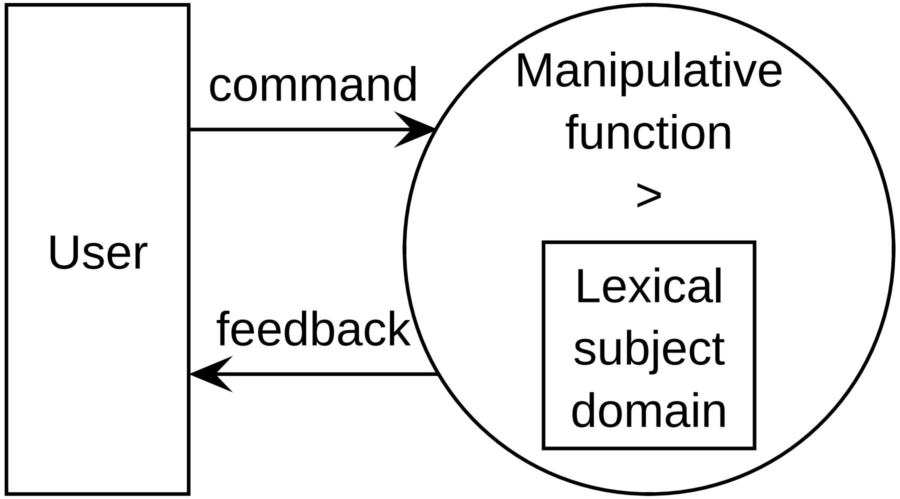
Typical structure in the context of an information-provision system



Typical structure in the context of a directive system



Typical structure in the context of a manipulative system



Main points

- Context is modeled, not designed.
- System boundary is determined by desired system services.
- Trade-off between functionality in the system and functionality in the environment.
- Context ends where relevant effects or relevant information ends.
- Context models have typical structures that depend upon typical system functionality.