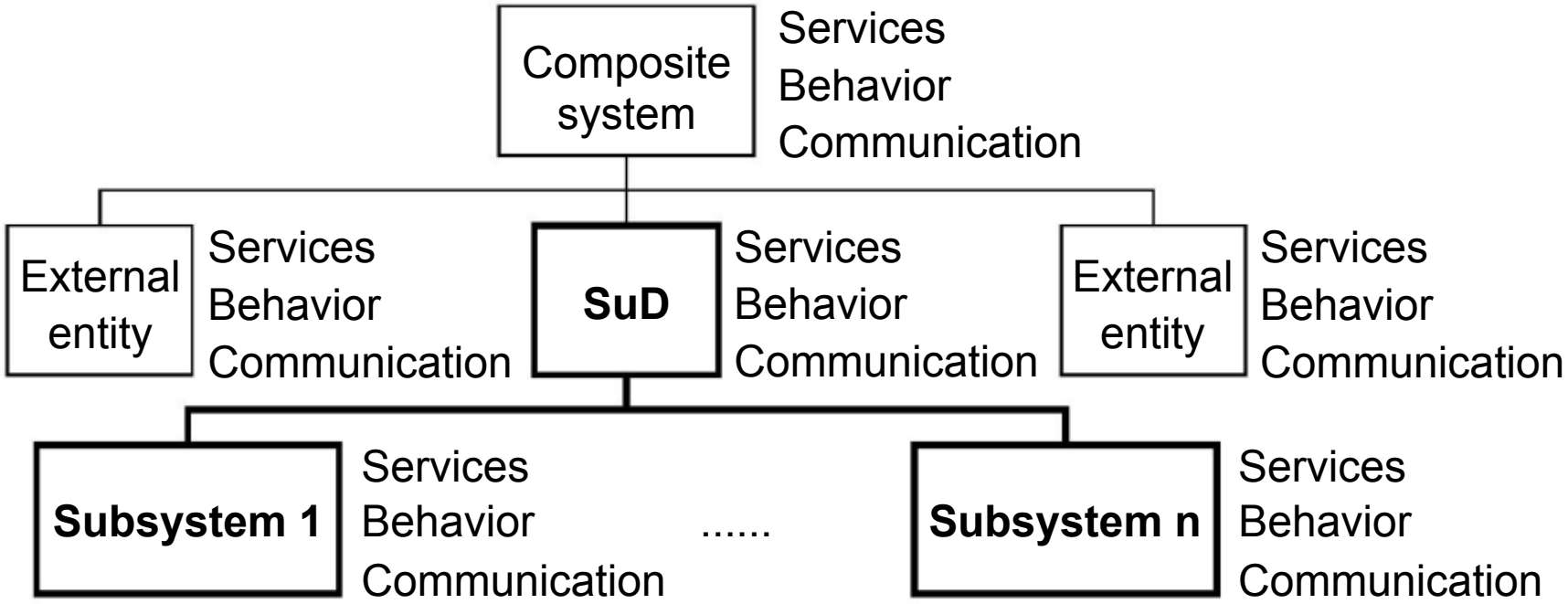
Slides for
Design Methods for Reactive Systems: Yourdon, Statemate and the UML

Roel Wieringa
Department of Computer Science
University of Twente,
the Netherlands
roelw@cs.utwente.nl
www.cs.utwente.nl/∼roelw

List of Slides

Chapter 19. Requirements-Level Decomposition Guidelines

This is design, not modeling.

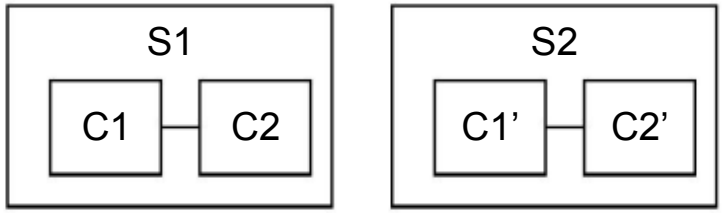


Architecture and architectural style

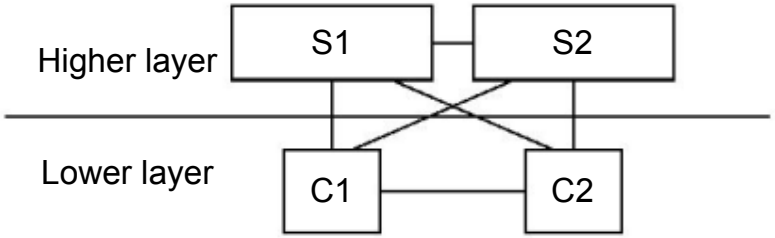
Architecture = a structure of elements and their properties, and the interactions between these elements that realize system-level properties.

- A system can have many architectures: e.g. requirements-level, implementation-level, execution-level, code-level.
- Elements in an architecture must have synergy. They must jointly produce emergent properties.
- An **architectural style** is a set of constraints on an architecture.

Basic style choice: Decomposition versus layering



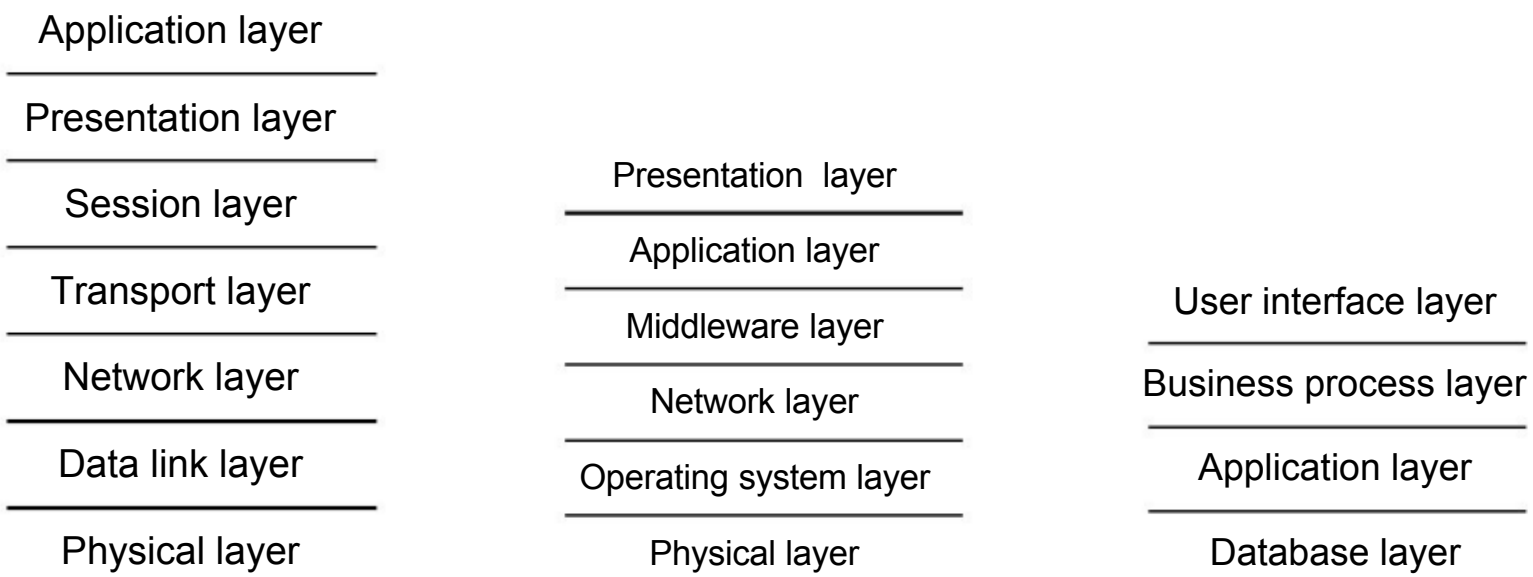
- C1, C2 deliver *services* to S1.
- C1', C2' deliver *services* to S2.
- S1 *encapsulates* C1, C2.
- S2 *encapsulates* C1', C2'.



- C1, C2 deliver *services* to S1 and S2.
- C1, C2 *are not encapsulated* by S1 or S2.

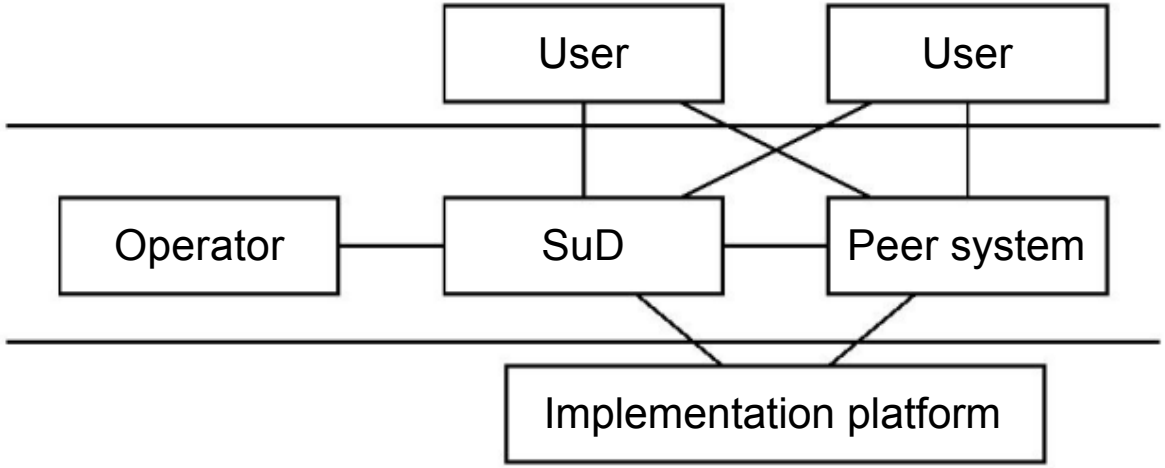
- Layering and decomposition can be mixed at various levels.
- Layering can be strict or loose.

Examples of layered contexts



(a) (b) (c)

Layered context diagram

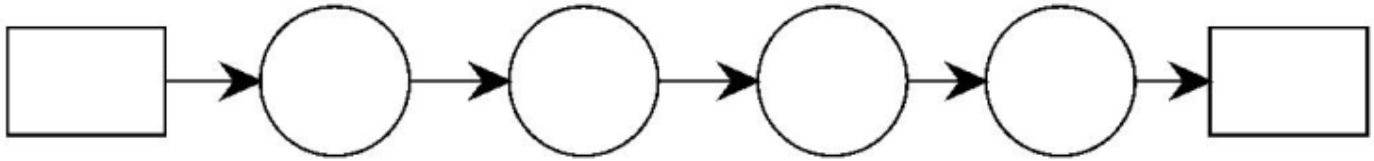


Structuring guidelines

- √ Choose a context structure that reflects the problem structure.
- √ Choose an SuD architecture that localizes changes.
  - Keep related data together
  - Keep related functions together

Architectural styles (1)

Data flow style:

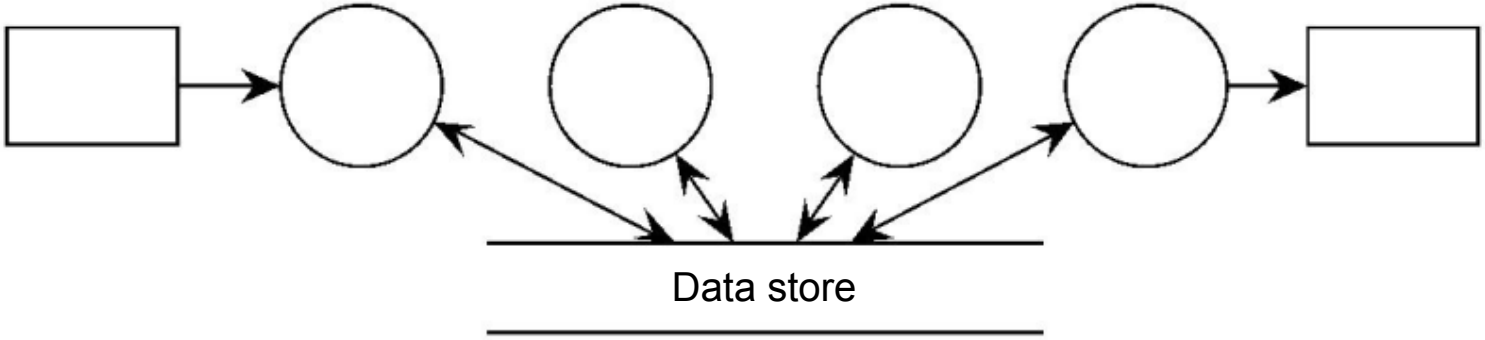


- Pure data flow style: data flows through a network of transformations (that have no persistent memory).
- Many variants: E.g. batch, pipe-and-filter.

Not applicable to reactive systems, because these need a (persistent) model of their environment.

Architectural styles (2)

Von Neumann style:



- This is the classical information system architecture style: Databases and application programs.
- Variant: *Blackboard style:* Intelligence in the data store(s). These can announce that they have been updated.

Architectural styles (3)

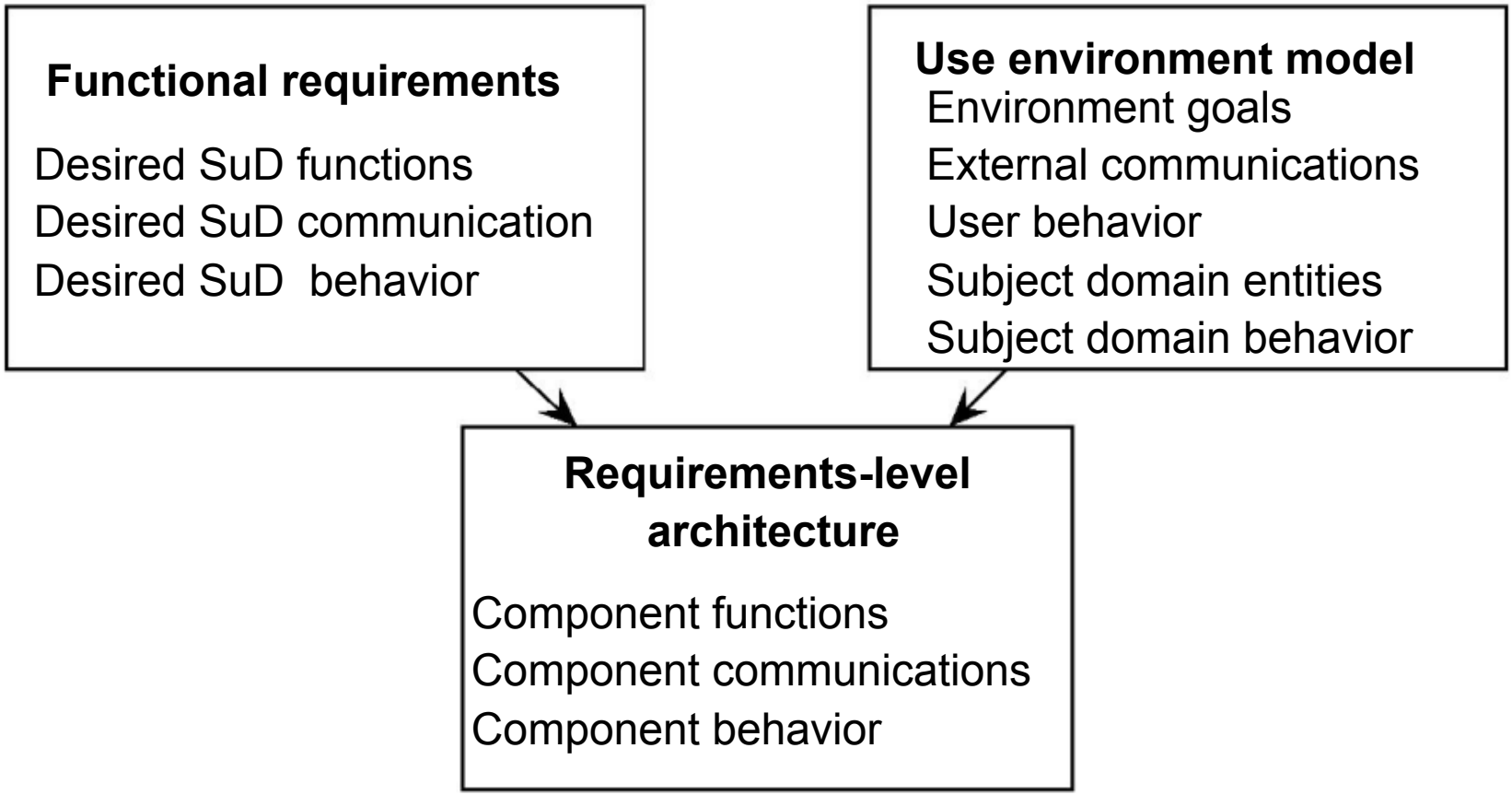
Object-oriented style.

- All components are objects.
- Usually with destination addressing.
- Variant: *publish-subscribe style:* kind of dynamically configurable channel addressing.

Requirements-Level Architecture Design Approach

- **Requirements-level decomposition:** Defined only in terms of requirements and environment models.
- Also called "conceptual architecture" or "logical architecture" or "essential system model".
- Requirements-level decomposition assumes "perfect technology" because it ignores technology. It is a restatement of the requirements in terms of a decomposition.
- **Implementation decomposition:** Mapping of requirements-level decomposition to some implementation platform.

Sources of design decisions



Classification of design decisions

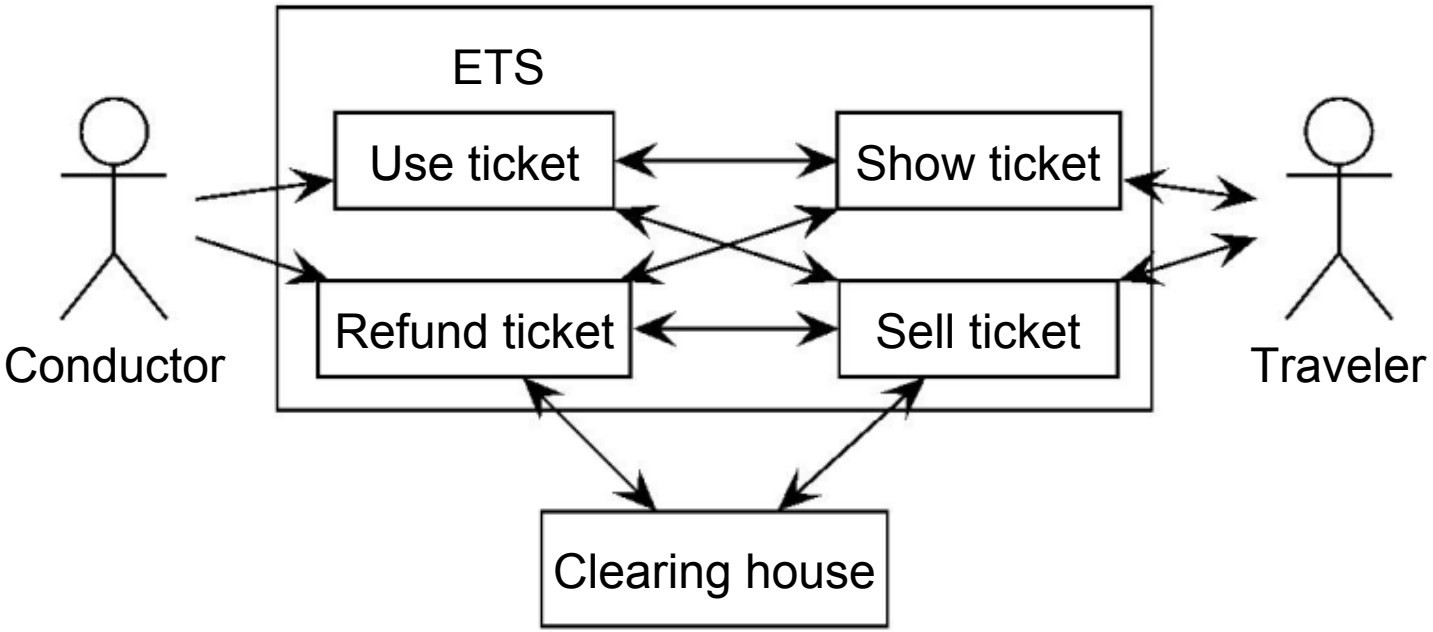
Design decisions for the requirements-level architecture are made in terms of

- Functions
- External communication
  - Events
  - Devices
  - Users
- External behavior
- Subject domain structure

Design guidelines

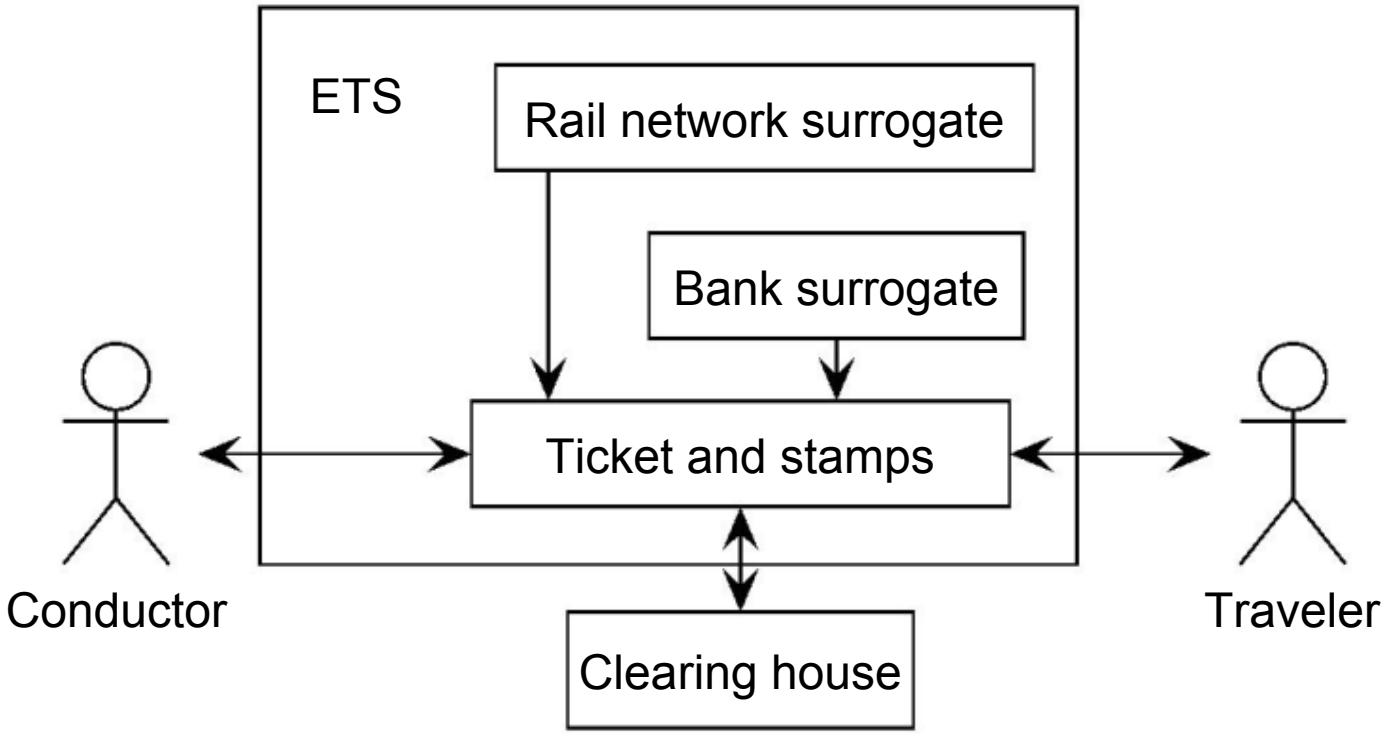
- √ Using **functional decomposition,** each system function is allocated to a different component.
- √ Using **subject-oriented decomposition,** each group of subject domain entities corresponds to a system component.

ETS example (1): Pure functional architecture —object-oriented style



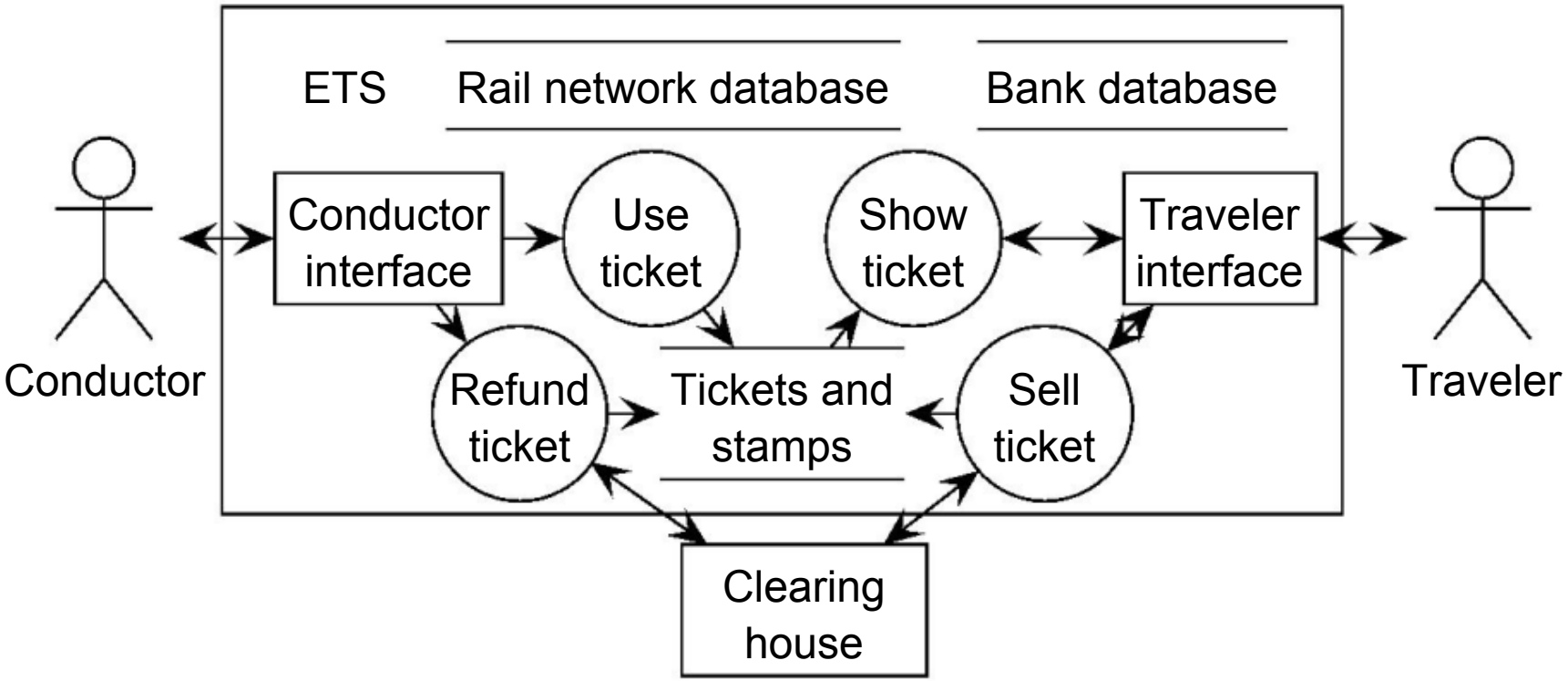
- All data encapsulated with functions!
- Inefficient.
- Does not adequately represent problem structure.

ETS example (2): Pure subject-oriented architecture —object-oriented style



- All functions encapsulated in data!
- Inefficient.
- Does not adequately represent problem structure.

ETS example (3): Mixed architecture —Von Neumann style



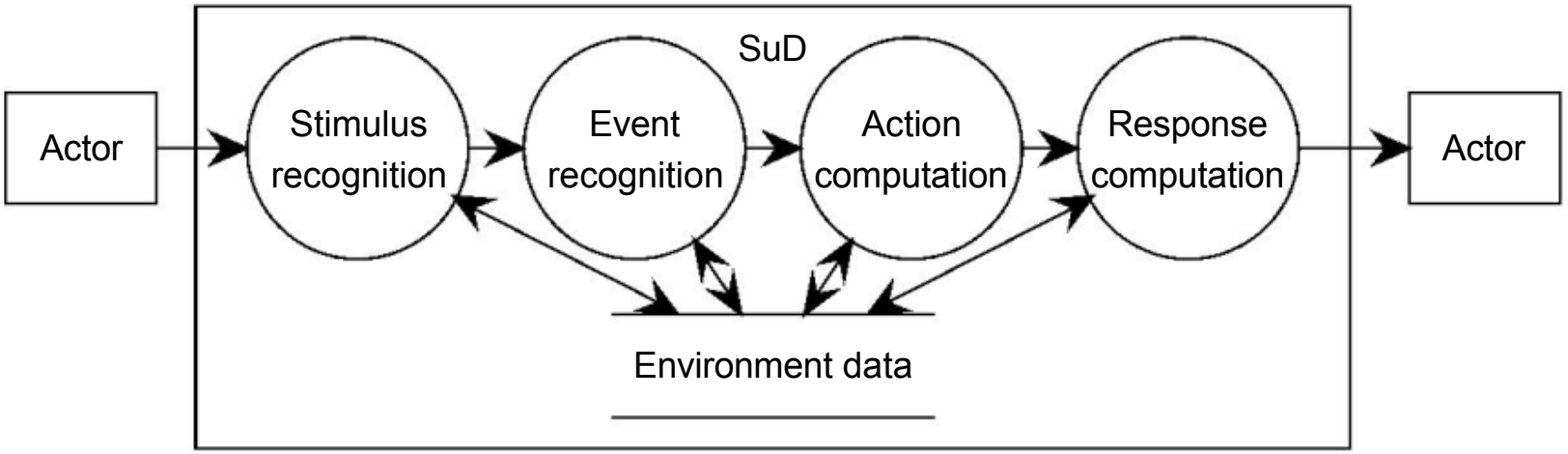
Criteria used:

- Functional decomposition
- Subject-oriented decomposition

- User-oriented decomposition

Communication-oriented decomposition

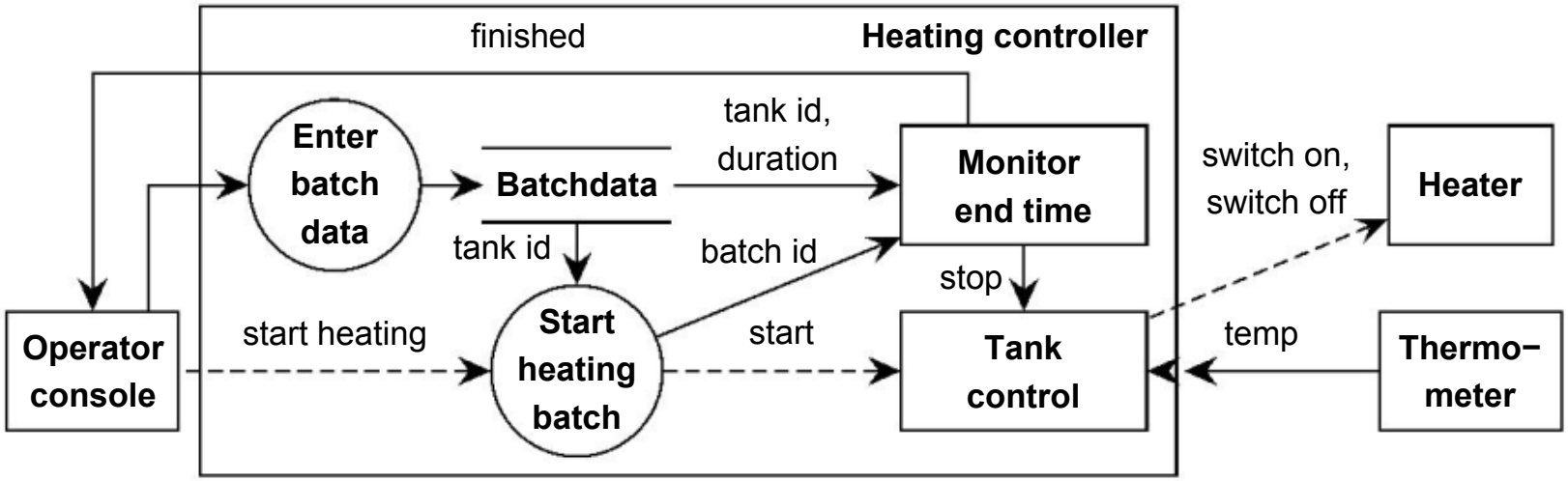
- √ Using **event-oriented decomposition,** each event is handled by a different component.
- √ Using **device-oriented decomposition,** each device is handled by a different component.
- √ Using **user-oriented decomposition,** communications with one kind of user are handled by one component.



Behavior-oriented decomposition

- √ In **behavior-oriented decomposition,** monitoring assumed behavior in the environment, or enforcing desired behavior in the environment, is allocated to one component.

Heating controller example (1): Mixing functional and subject-oriented decomposition

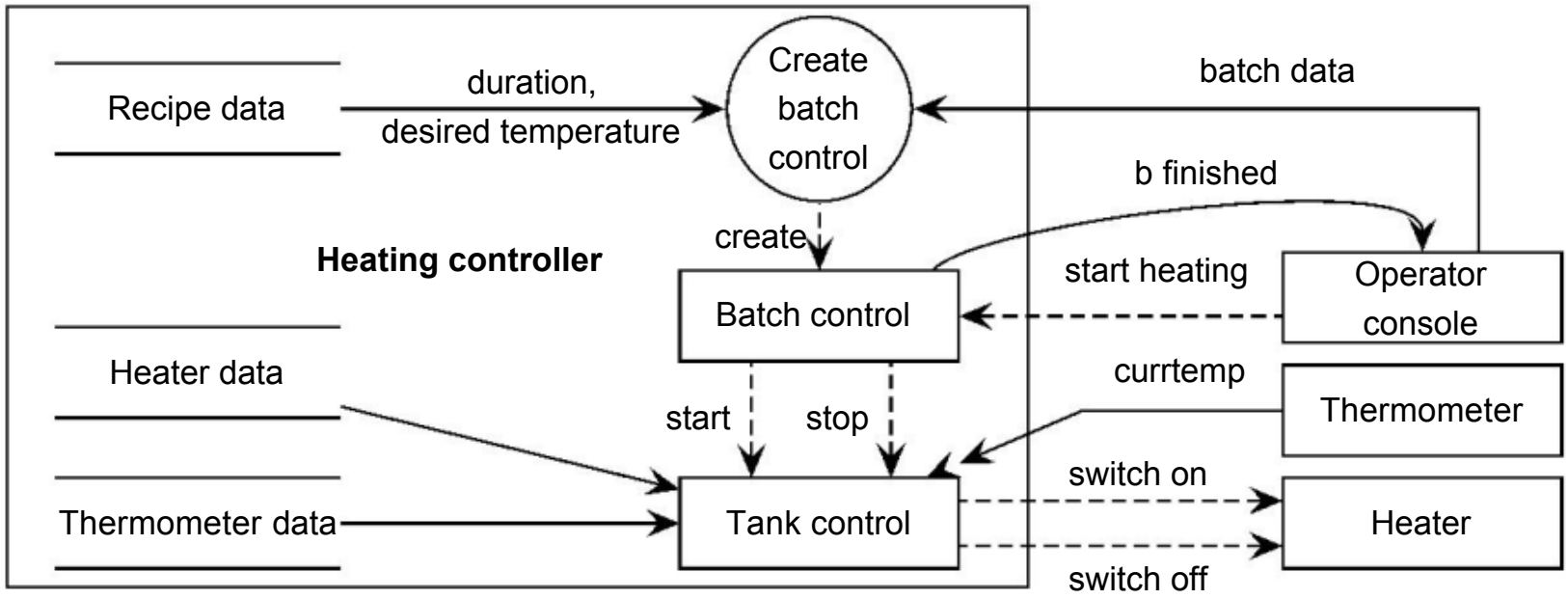


Guidelines used:

- Functional
- Subject-oriented

Different aspects of batch behavior are left scattered around.

Heating controller example (2): Including behavior-oriented decomposition



Guidelines used:

- Functional
- Subject-oriented
- Behavior-oriented

Evaluation criteria

- √ Check that all data used is created, that all data created is deleted.
- √ Execute the model.
- √ Produce a correctness argument that C_1 and ... and C_n entail S.
- √ Check that quality attributes (efficiency, safety, reliability, etc.) are realized.
- √ Build a throw-away prototype and experiment with it.

Main points

- Distinguish requirements-level decomposition from implementation decomposition.
- Basic architectural choice is between layering and encapsulation.
- Other major architectural styles: Data flow, Von Neumann, Object-oriented.
- Requirements-level decomposition guidelines:
  - Functional,
  - Subject-oriented,
  - Communication-oriented: events, devices, users
  - Behavior-oriented

  These correspond to major system aspects & subject domain.

Chapter 20. Postmodern Structured Analysis (PSA)

History of structured analysis;

- Introduced in the 1970s as requirements specification method.
- Abstracts structured programming to requirements level.
- Idea (1): SuD structure should match problem structure rather than structure of implementation platform.
- Idea (2): The SuD should be modular.

  Yourdon-style structured analysis claims that modularity is achieved by functional decomposition. But:

  – Functional decomposition is not modular if there are many interfaces between the functions.

  – Chapter 19 (Requirements-Level Architecture Guidelines) lists many other decomposition guidelines.

Postmodern structured analysis (PSA)

Differs from classical structured analysis:

- ERD used for subject domain only.
- Extended context diagram.
- Event flows may contain data.
- STDs can be statecharts.
- STDs may have local variables.

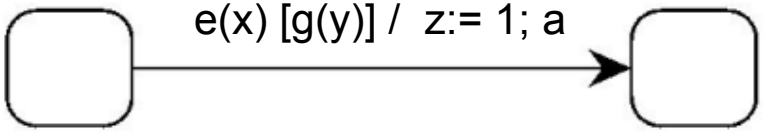
Notations used in PSA.

Design level	Notation
Environment	• Context diagram • ERD of subject domain • Event-action lists of desired subject domain behavior • Event-action lists of assumed subject domain behavior
Requirements	• Mission statement • Function refinement tree • Service descriptions • Stimulus-response list of desired system behavior
SuD decomposition	• DFD SuD • STTs or STDs of control processes • Dictionary

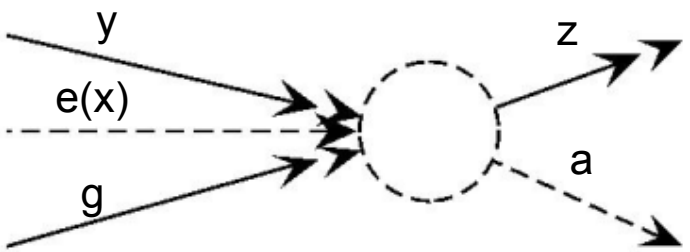
Coherence rules

- *Environment models.*
  - Context diagram shows relevant communication paths between SuD and subject domain.
  - Event-action pairs refer to subject domain entities involved.
- *Requirement specifications.*
  - Mission statement = root of function refinement tree.
  - Service descriptions = leaves of FRT.
  - Each function is triggered by stimulus in SR list.
  - Each stimulus-response pair is part of a function.
- *Decomposition specifications.*
  - Each control process is specified by a behavior description.
  - Each behavior description describes a process in the DFD.

Relation between an STD and the control process that it specifies



(d)



(e)

Coherence across models

- *Environment and requirements.*
  - Event stimulus and response action ...
  - and each of these is a path in the context diagram.
- *Requirements and decomposition.*
  - For each SR pair, there is a path through the DFD.
- *Decomposition and environment.*
  - Context diagram is abstraction from DFD.
- *Dictionary.*
  - The dictionary defines at least the relevant subject domain terms for entities and events.

Flyweight to heavyweight

The weight of professional boxers is classified according to the following scheme:

flyweight	≤ 112 pounds
bantamweight	≤ 118 pounds
featherweight	≤ 126 pounds
lightweight	≤ 135 pounds
welterweight	≤ 147 pounds
middleweight	≤ 160 pounds
heavyweight	> 160 pounds

We can learn two things from this classification:

1. Lightweight is not the lightest weight.
2. Heavyweights can be as heavy as they want.

Flyweight to heavyweight use of notations in PSA

Flyweight	Featherweight	Middleweight	Heavyweight
Context diagr	Context diagr	Context diagr	Context diagr
	ERD	ERD	ERD
			EA list
Mission stmt	Mission stmt	Mission stmt	Mission stmt
	Function reft	Function reft	Function reft
	Service descs	Service descs	Service descs
			SR list
		DFD	DFD
			Behavior descs
	Dictionary	Dictionary	Dictionary

Main points

- PSA is an extension of Yourdon structured analysis that uses all notations of this book except communication diagrams.
- PSA can be used with any desired level of weightiness.
- PSA can be used with any of the decomposition guidelines of chapter 19.

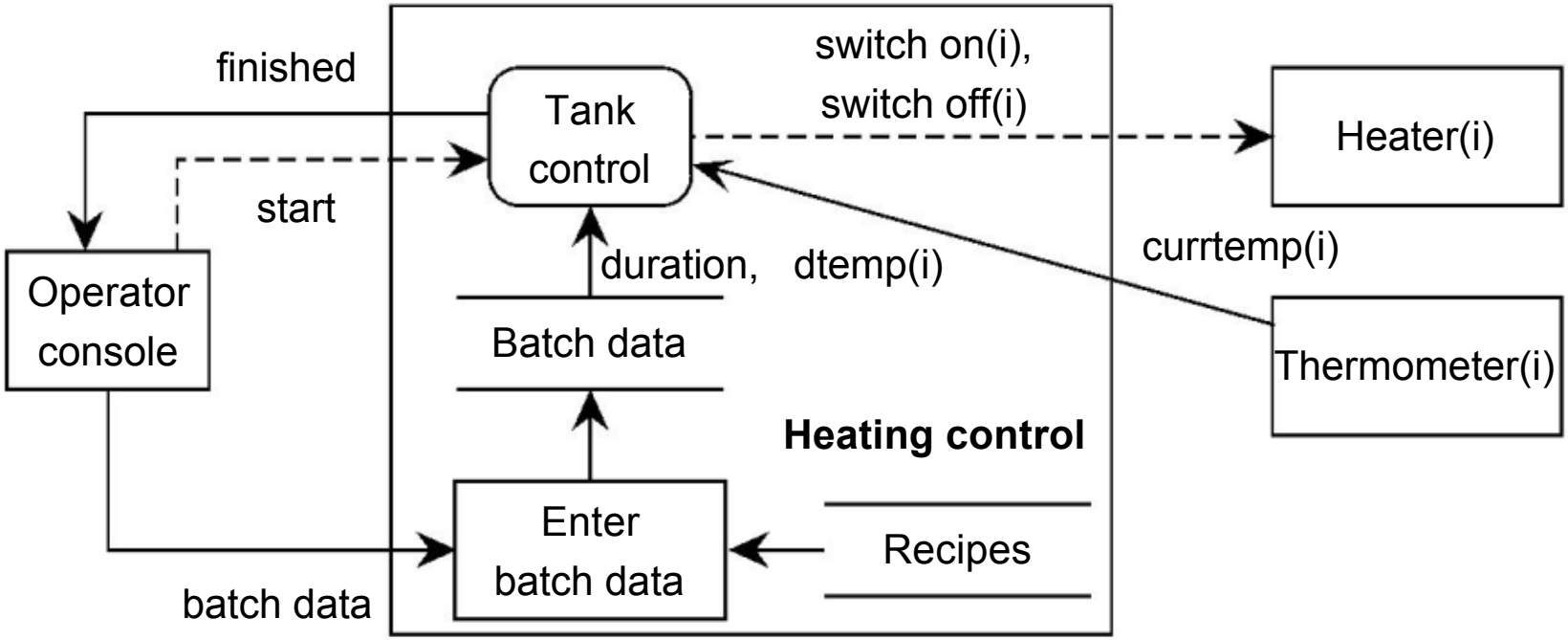
Chapter 21. Statemate

- Developed in the 1980s based on the idea of higraph: Hierarchical hypergraphs.
  - Hyperedges.
  - Nodes can contain nodes, as in Venn diagrams.
- Activity charts are hierarchical DFDs.
- Control activities are specified by statecharts.
- There is a precisely defined execution semantics for activity charts and statecharts.
- Module charts represent computational resources (not treated here).

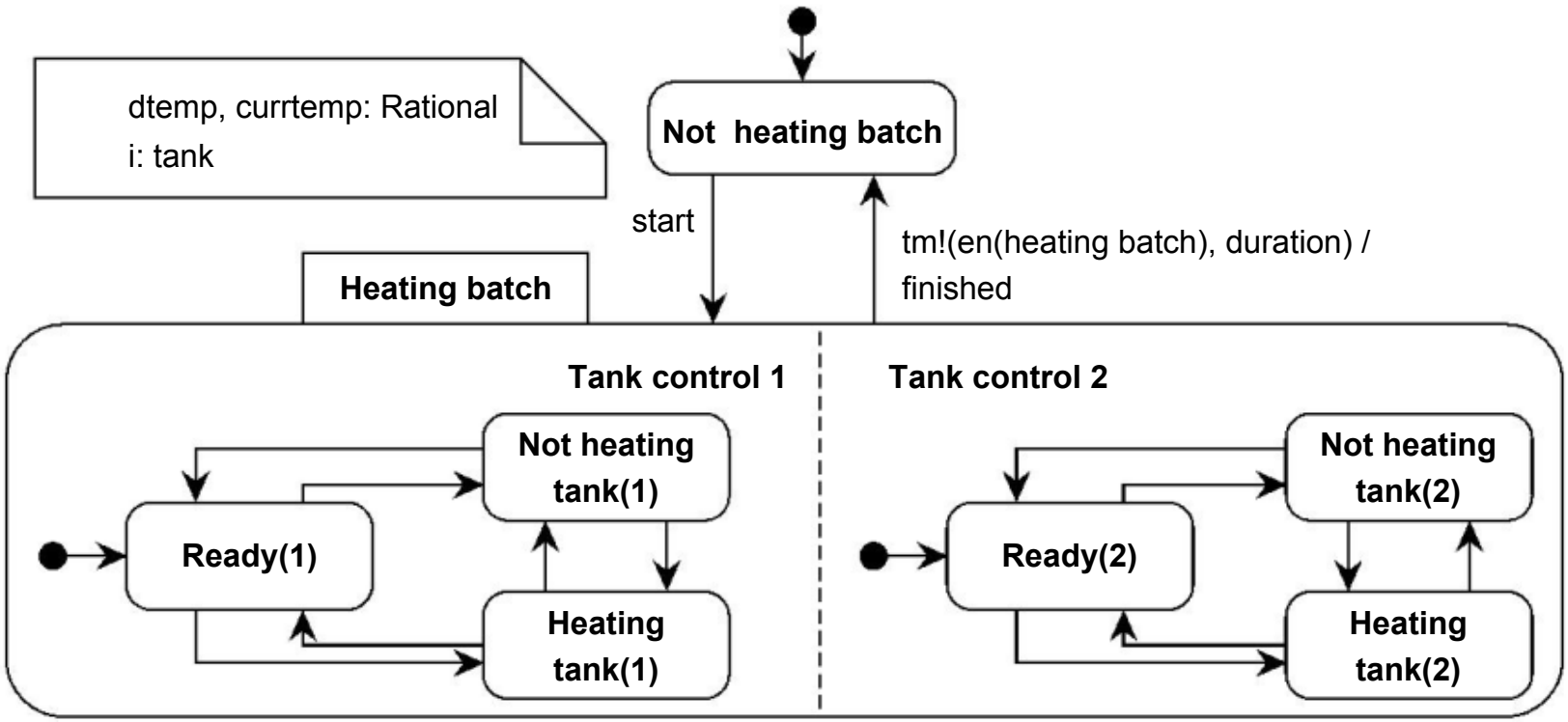
Statemate can be used in combination with other PSA notations.

Activity chart

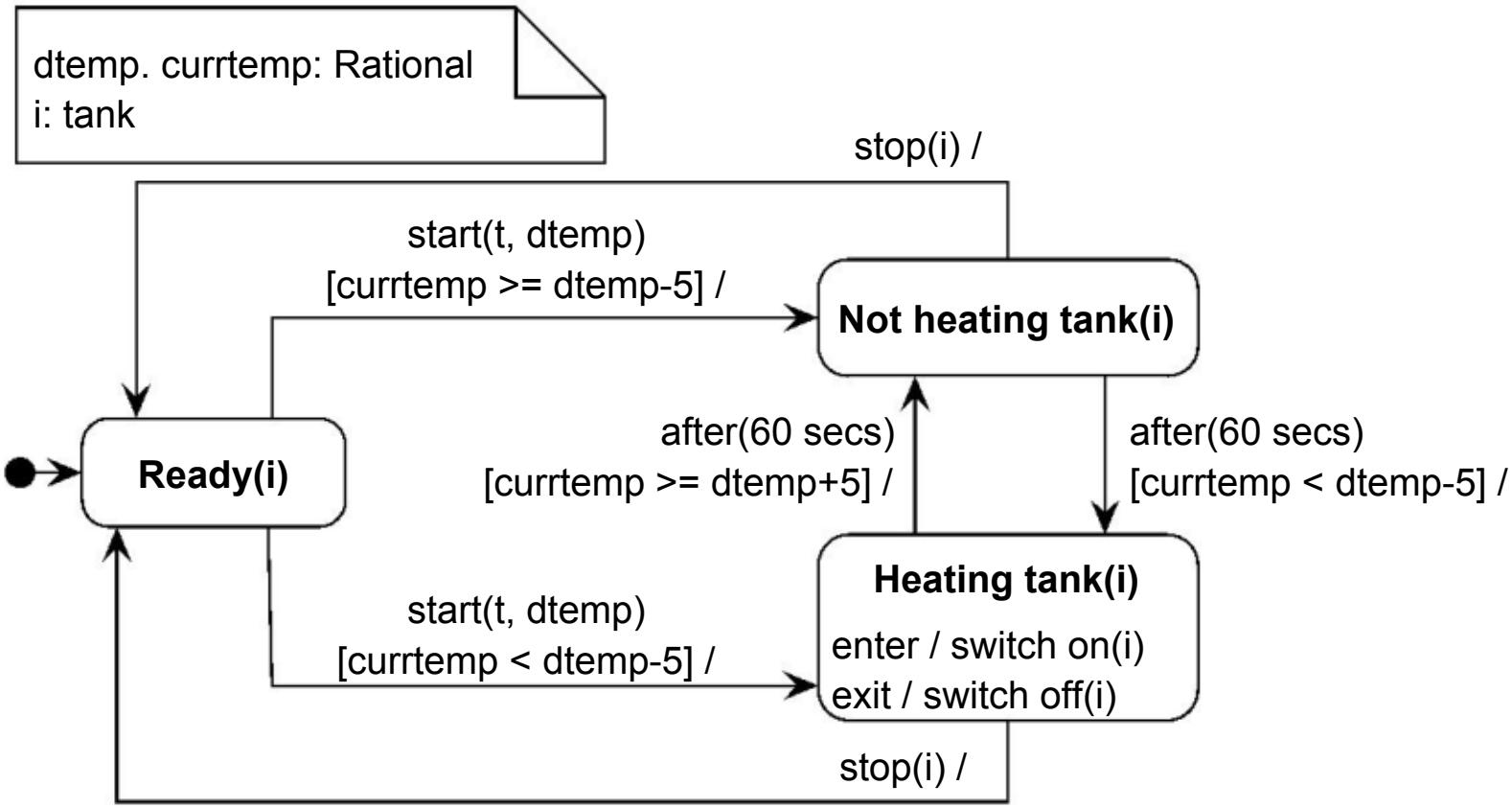
Statemate analogon of DFDs.



Statechart for the control activity



Substatecharts



Temporal events

Timeout:

- For each event e and natural number n, timeout(e, n),

  abbreviated to tm!(e, n), occurs n time units after the most recent occurrence of the event e.

An after(n) event that leaves state S can be defined as tm!(en(S), n).

- When a statechart enters a state S, Statemate generates the

  event en(s).
- When it exits S, it generates the event ex(S).

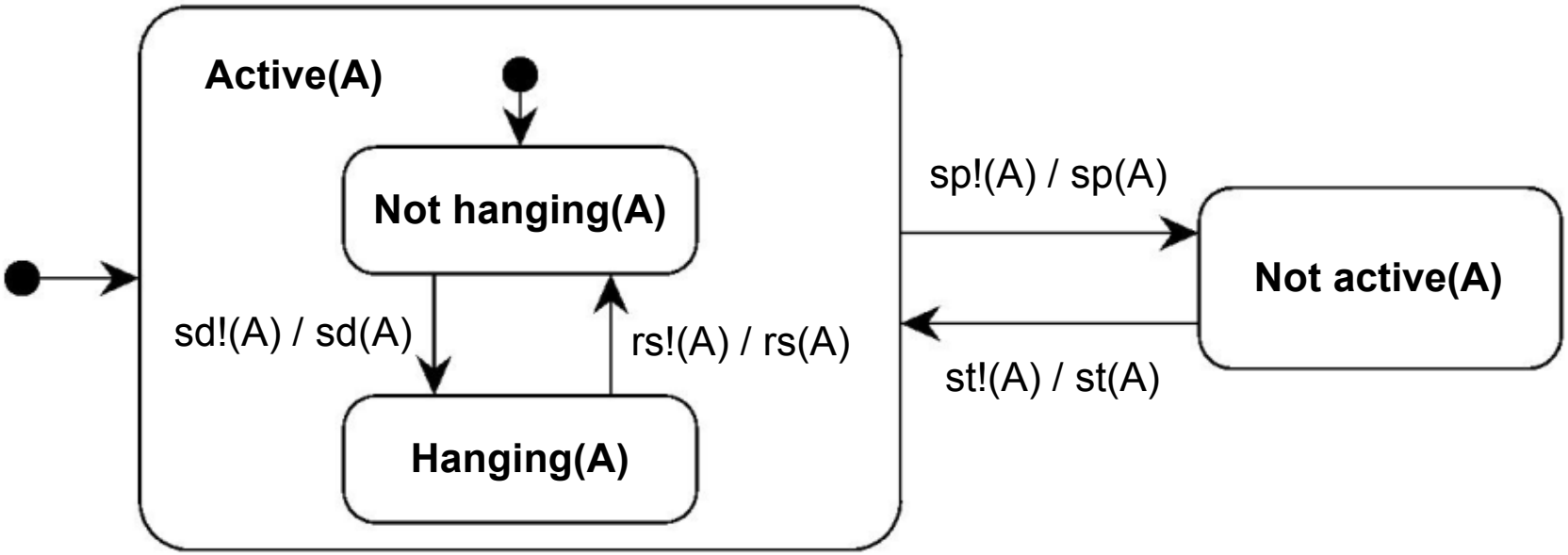
Scheduled action:

- For each action a and natural number n, the action schedule(a,

  n), abbreviated sc!(a, n), schedules the action a to occur exactly n time units later.

when(n) / a can now be expressed as sc!(a, n).

Activity states



Changing activity status

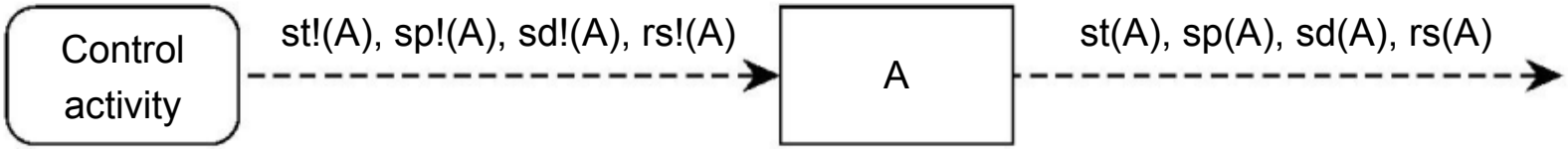
Actions that will cause a change in state of an activity:

sp!(A)	stop(A)
st!(A)	start(A)
sd!(A)	suspend(A)
rs!(A)	resume(A)

Events generated when an activity changes state:

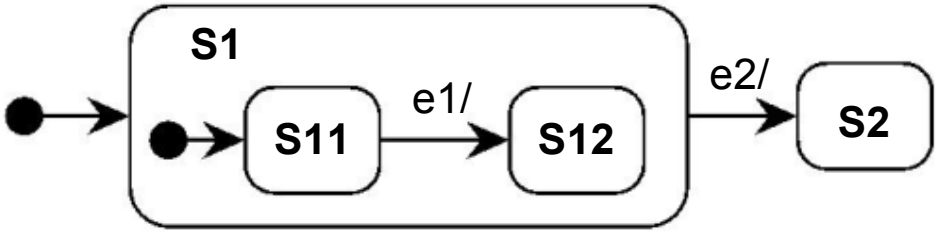
sp(A)	stopped(A)
st(A)	started(A)
sd(A)	suspended(A)
rs(A)	resumed(A)

Interface between control activity and sibling activities

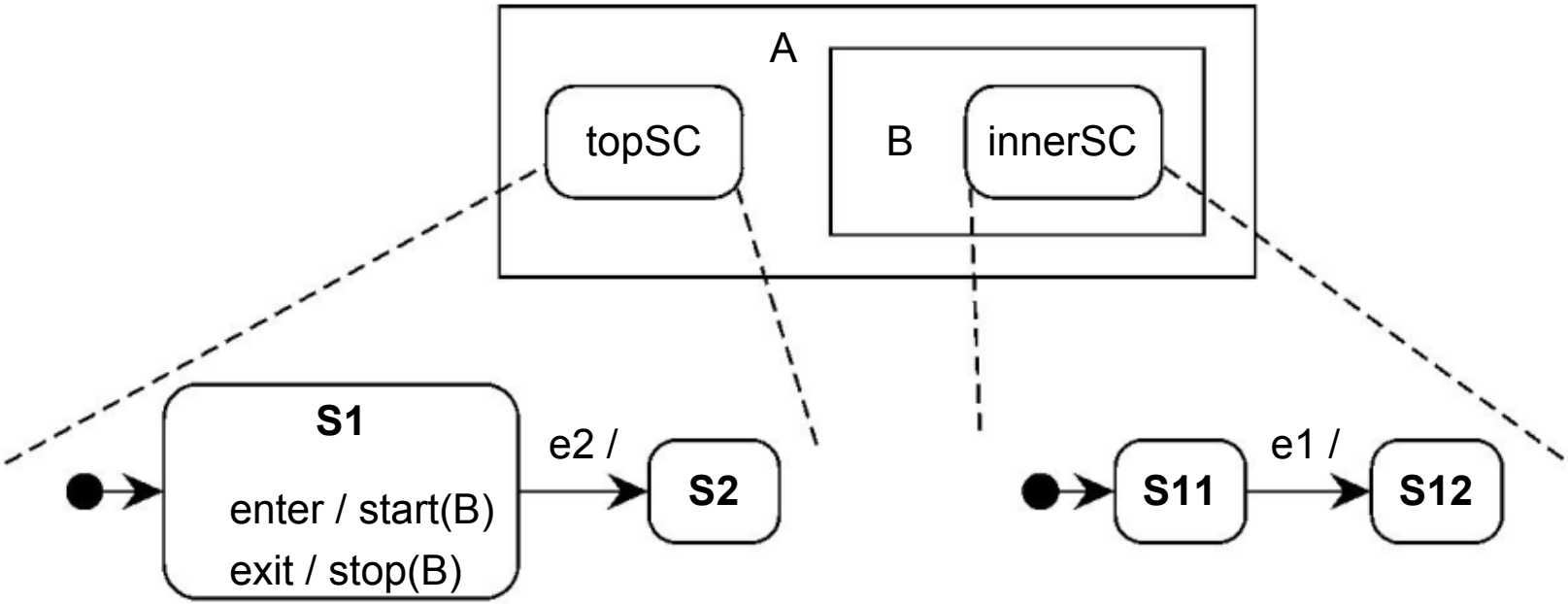


This interface is always present, but it is not shown in an activity chart.

Hierarchy in statecharts and in activity charts (1)

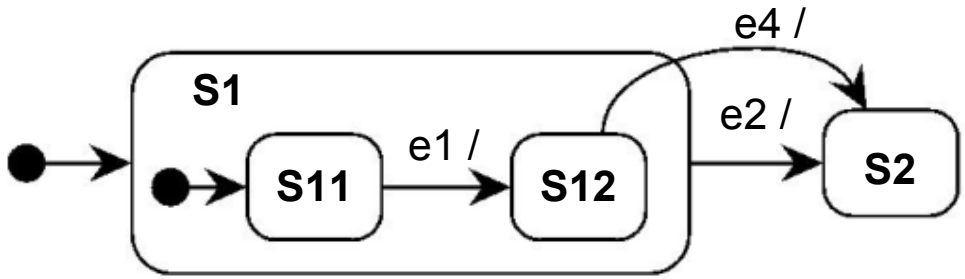


Moving the activity to another statechart leads to an equivalent model (having the same step semantics):

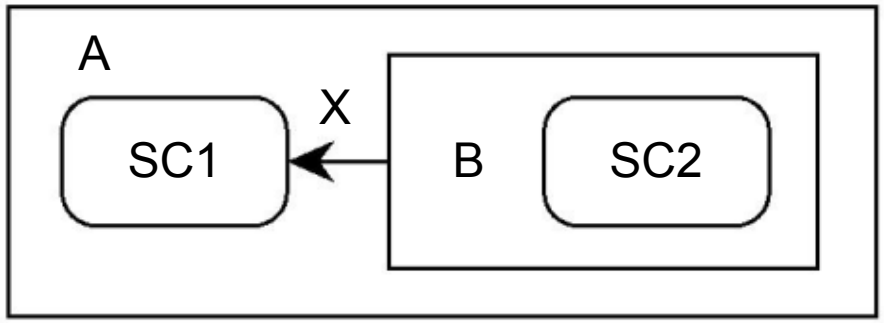


Hierarchy in statecharts and in activity charts (2)

This can be done too if there is an outward transition:

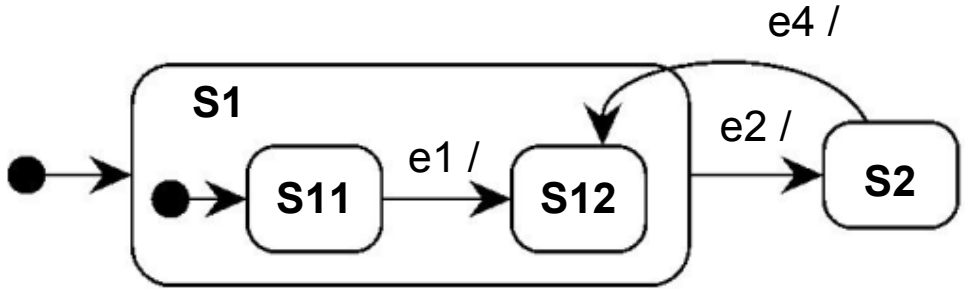


can be replaced by the equivalent model (same step semantics):

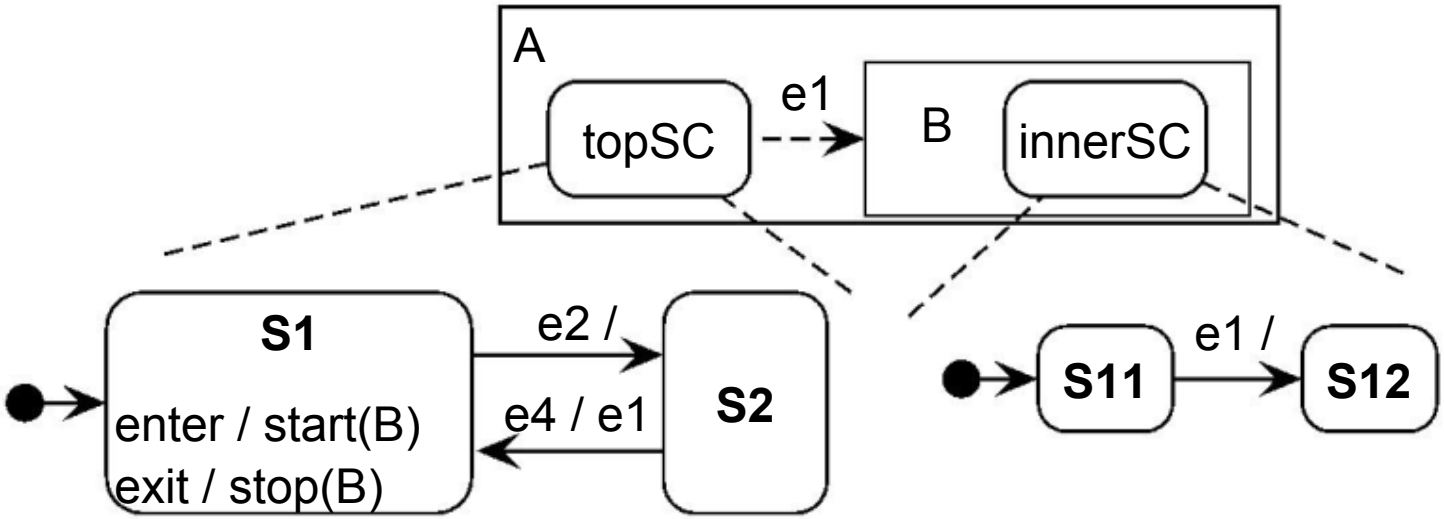


Hierarchy in statecharts and in activity charts (3)

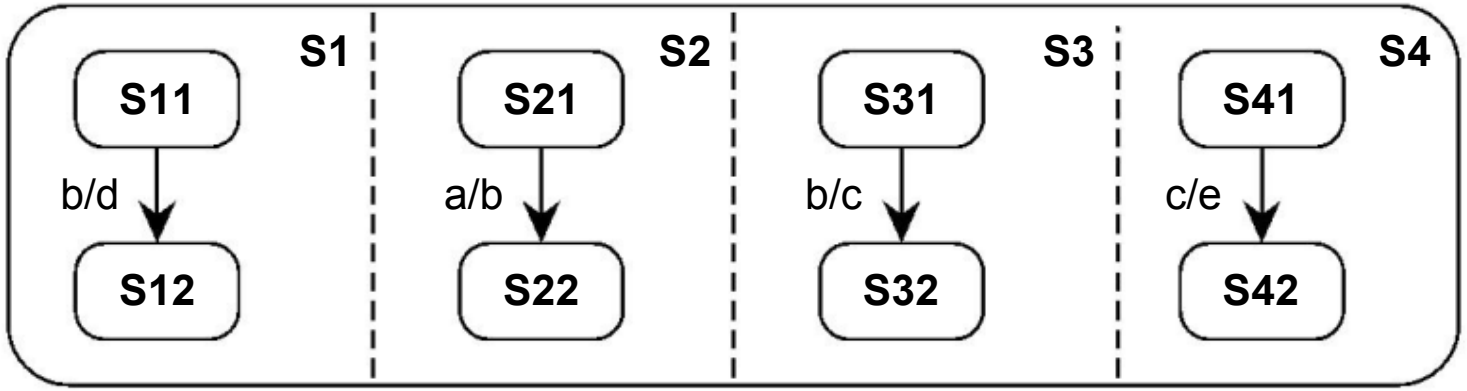
When there is an inward transition this does not work:



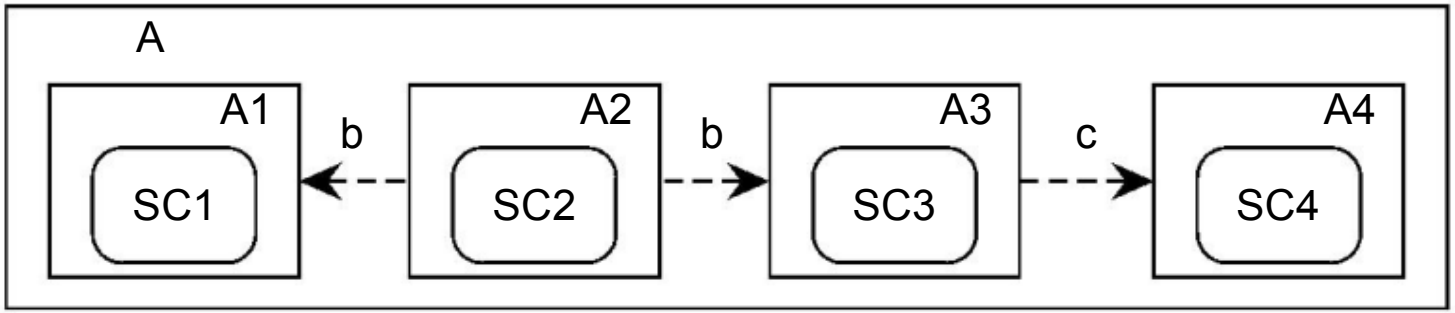
is not equivalent to this model (different step and superstep semantics):

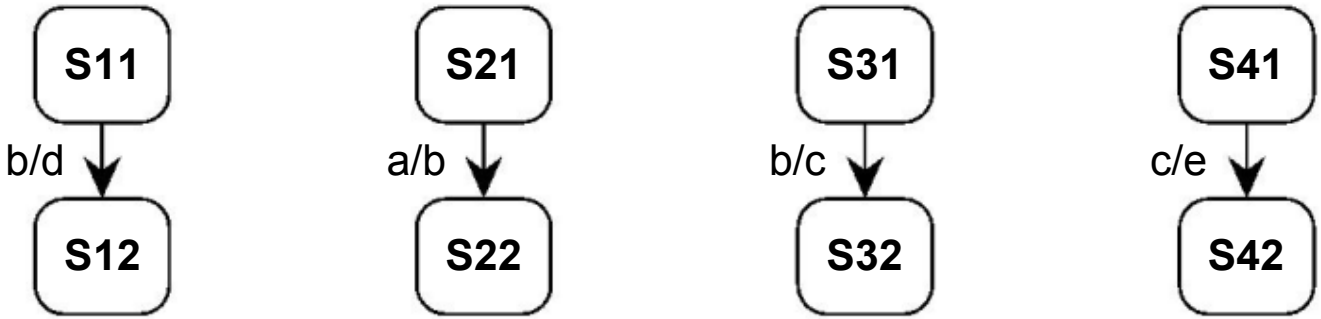


Parallelism statecharts and in activity charts (1)



Parallelism statecharts and in activity charts (2)





Execution semantics (1): Choices made

- *Channel capacity.* Flows have capacity one.
- *Time-continuous flows.* All flows are time-continuous. They behave as data stores.
- *Input buffer.* Each activity has an input buffer, which is a set.
- *Priority.* If a state transition conflicts with a state reaction, then the transition has priority. If two transitions are in conflict, then the highest-level one has priority.
- *Step semantics.* In a step, the system responds to all events that occurred since the start of the previous step.
- *Perfect technology.* Steps do not take time.
- *Breadth-first.* Activities are executed breadth-first.

Execution semantics (2): Execution state

- The status of the system, which consists of the following items:
  - The current configuration of the statecharts;
  - The values of all variables;
  - The truth-values of all conditions;
  - The activation state of each activity;
  - A list of internal events generated during the previous step;
  - A list of outstanding timeout events;
  - A list of outstanding scheduled actions.
- The time currently indicated by the system clock;
- A list of external events that have occurred since the beginning of the previous step.

Execution semantics (3): Execution step

Time is T during the step.

1. Construct the set E of events to be responded to. First, set $E := 0$.
   (a) Collect all external events generated by the environment since the previous step and add them to E.
   (b) Collect all events generated in the previous step and add them to E.
   (c) Collect all events generated by the events in E, and add them to E (recursively).
   (d) Execute all scheduled actions whose time is due in (T, T+1].
   (e) Process the list of timeout events. For each tm(e, n) in the list,
   - if e ∈ E compute and record the time at which tm(e, n) is to be generated;
   - else if the time for tm(e, n) to occur falls in the interval (T, T+1], then generate the event e and deactivate the timeout event.

2. Construct from E a maximal step S to be executed. First, set $S := 0$. Then:
   (a) Compute the set En of enabled transitions and state reactions, i.e. those whose triggering event occurred and whose guard is true. Remove from En those transitions or reactions that are in conflict with a transition of higher priority in En.
   (b) Compute from En maximal nonconflicting sets S of transitions. Add to each set the state reactions that do not conflict with it. Each resulting set is called a *step*.
   (c) Select a step S to be executed.
3. Execute S.
   (a) Add scheduled actions in the step to the list of scheduled actions.
   (b) Perform all other actions in the step.
   (c) Update the information on the history of states.
   (d) Update the current configuration.

Execution semantics (4): Superstep execution

1. Construct the set E of events to be responded to as for a step.
2. Repeat the following until $E = \emptyset$:

   2.1 Construct from E a maximal system step S to be executed as for a step. (There are no internal events at the beginning of a superstep.)

   2.2 Execute S as in the step execution semantics.

   2.3 Reconstruct the set E of events to be responded to. First, set $E := \emptyset$. Then:

   (a) Collect all events generated on internal flows and add them to E.

   (b) Generate derived events from the events in E and add these to E (recursively).

Main points

- Activity charts are a syntactic variant of DFDs.
- Statecharts have an execution semantics in STM.
- The presence of hierarchy and parallelism in statecharts as well as in activity charts increases the number of specification options for the author of a specification.

Chapter 22. The Unified Modeling Language (UML)

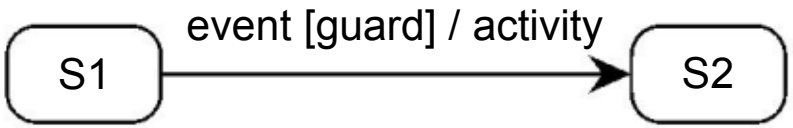
- Attempted unification of notations for object-oriented software design.
- Industrial standard defined by the OMG.
- Standard is still being updated.
- Different books present different versions!
- We treat only a light-weight version, expected to be consistent with future versions.

The UML contains eight notations

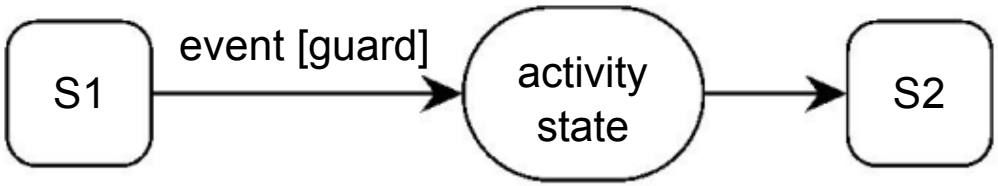
UML notations can be used in different methods in different ways. We will use them as follows. (Notations between brackets are not used by us.)

- **Activity diagrams.** User workflow.
- (Use case diagrams. System functions and context.)
- **Static structure diagrams.** Decomposition into software objects.
- **Statecharts.** Object life cycles.
- **Sequence diagrams.** Message-passing during a scenario.
- **Collaboration diagrams.** Message-passing during a scenario.
- (Component diagrams. Dependencies between executables.)
- (Deployment diagrams. Network of computing resources.)

Activity diagrams



- A transition in a statechart is instantaneous.
- If we want to represent that an action takes time, turn it into a **activity state:**

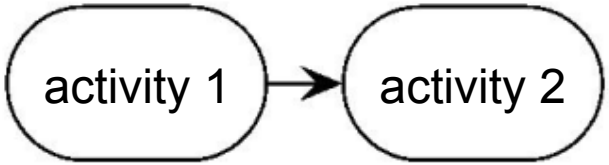


- Transitions are still instantaneous.

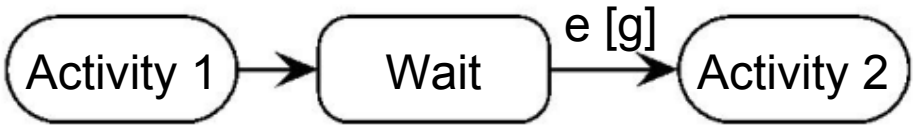
Workflow

- We will use activity diagrams to represent user workflows.
- Each individual workflow handles a case.
- Each individual workflow has local variables: state of the workflow, state of the case, list of events to be responded to.
- The variables are usually stored in a database.
- Guards are conditions on these variables.

Representing sequence

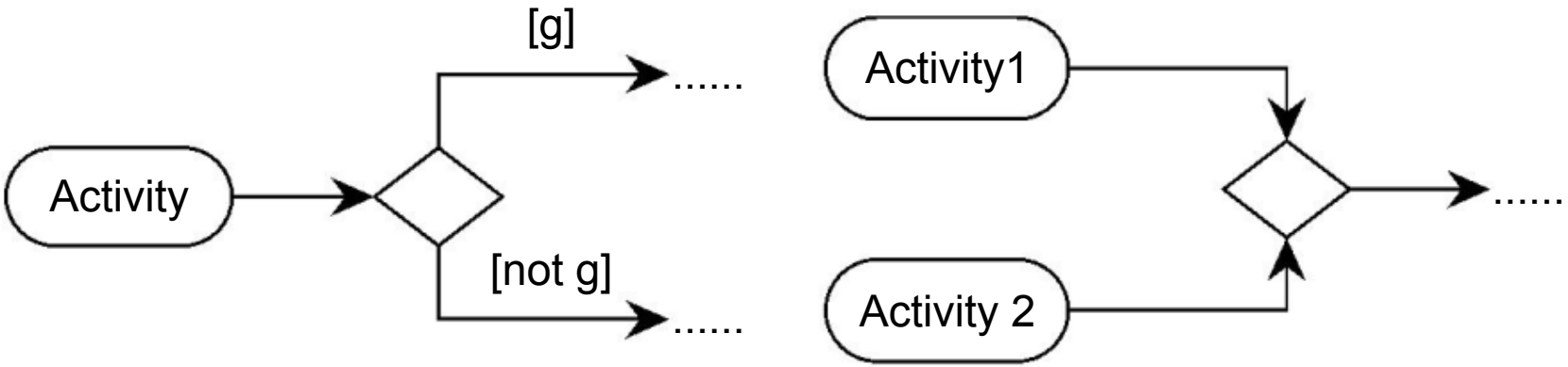


(k) When activity 1 terminates, activity 2 starts.



(l) When activity 1 terminates, the workflow waits for e to occur when g is true, before starting activity 2.

Representing choice

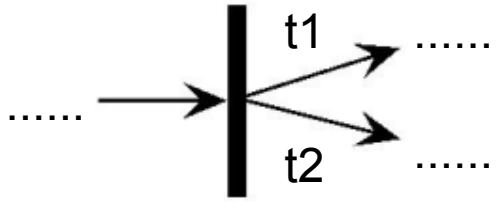


(m) Or-split.

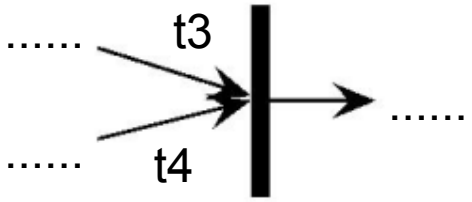
(n) Or-join.

Guards are tested on the workflow variables.

Representing parallelism

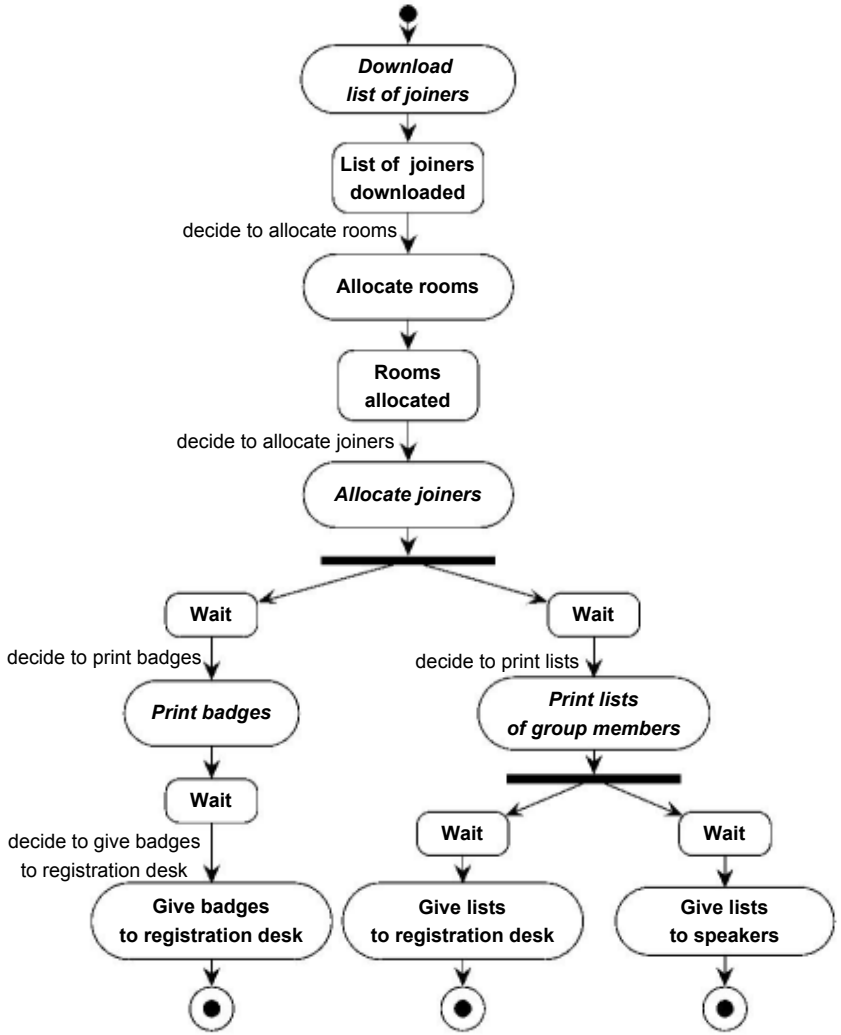


(o) And-split. t1 and t2 are executed simultaneously.



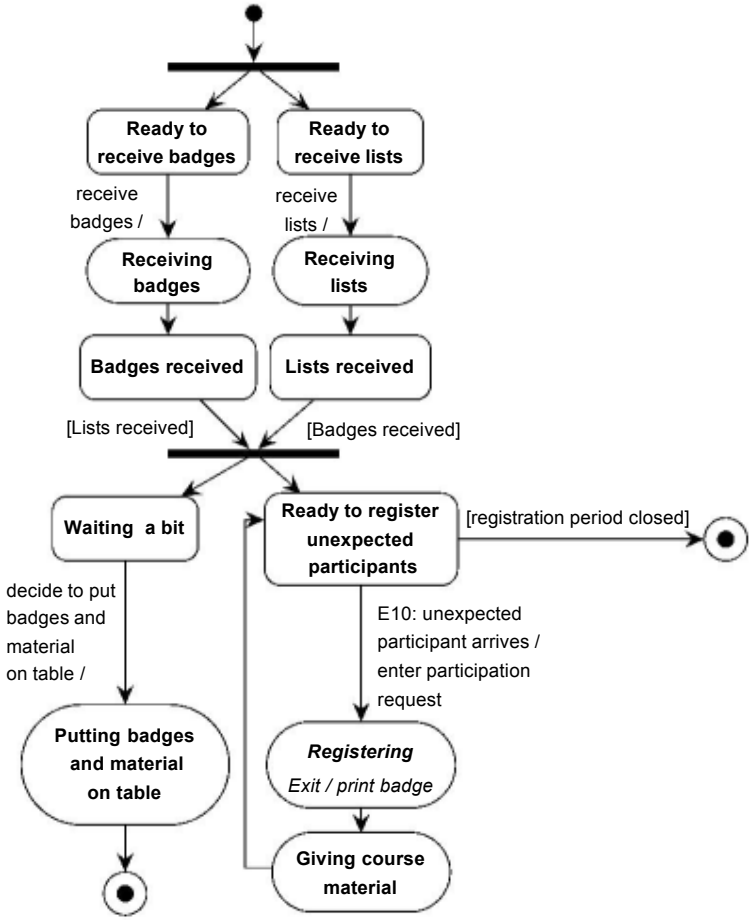
(p) And-join. t3 and t4 are executed simultaneously.

Workflow of the course coordinator



Activities in italics to be supported by TIS.

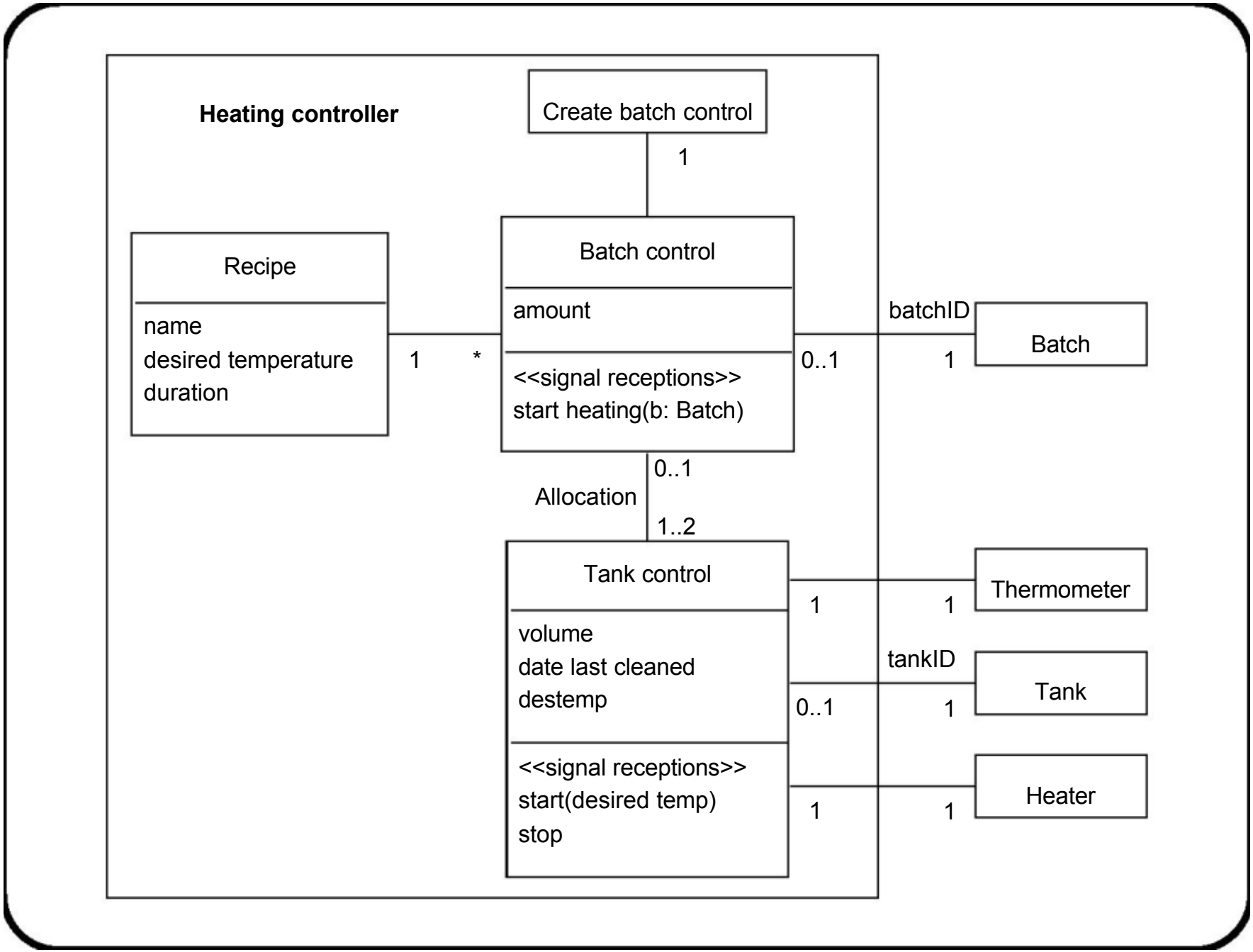
Workflow at the registration desk



More examples in appendix H (www.mkp.com/dmrs)

Static Structure Diagrams (SSDs)

- Also known as *class diagrams.*
- An **object** is a software entity with a fixed identity, a local state and an interface through which it offers services to its environment.
- An **object class** is an object type.
  - Class intension: All properties shared by all instances.
  - Extension: All possible instances.
  - Extent: Set of currently existing instances.


Heating controller
Create batch control
1
Recipe
name
desired temperature
duration
1
*
Batch control
amount
<<signal receptions>>
start heating(b: Batch)
0..1
batchID
1
Batch
0..1
Allocation
1..2
Tank control
volume
date last cleaned
destemp
<<signal receptions>>
start(desired temp)
stop
1
1
Thermometer
tankID
0..1
1
Tank
1
1
Heater

Meaning of an SSD

- Which classes of objects can exist in the software system,
- and how many of them can exist.
- Attributes are local variables of the object.
- Associations are access paths!
- Services can be operations or signal receptions.
  - **Operation** = computation performed by object.
  - **Signal** = named data structure that can be sent as a message to objects.

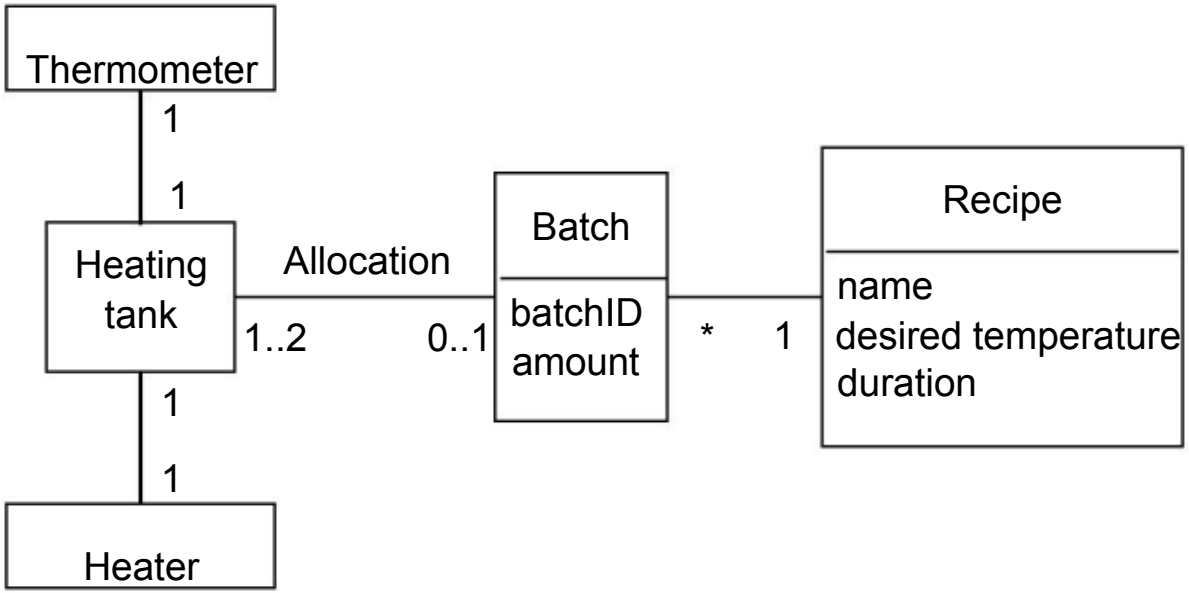
ERDs and SSDs

ERD	SSD
entity type	class
entity	object
relationship	association
tuple	link
association entity	association object
association entity type	association class
cardinality property	multiplicity property

The major differences are:

- ERD used to represent structure of subject domain.
- SSD used to represent structure of software objects.
- Objects offer services.

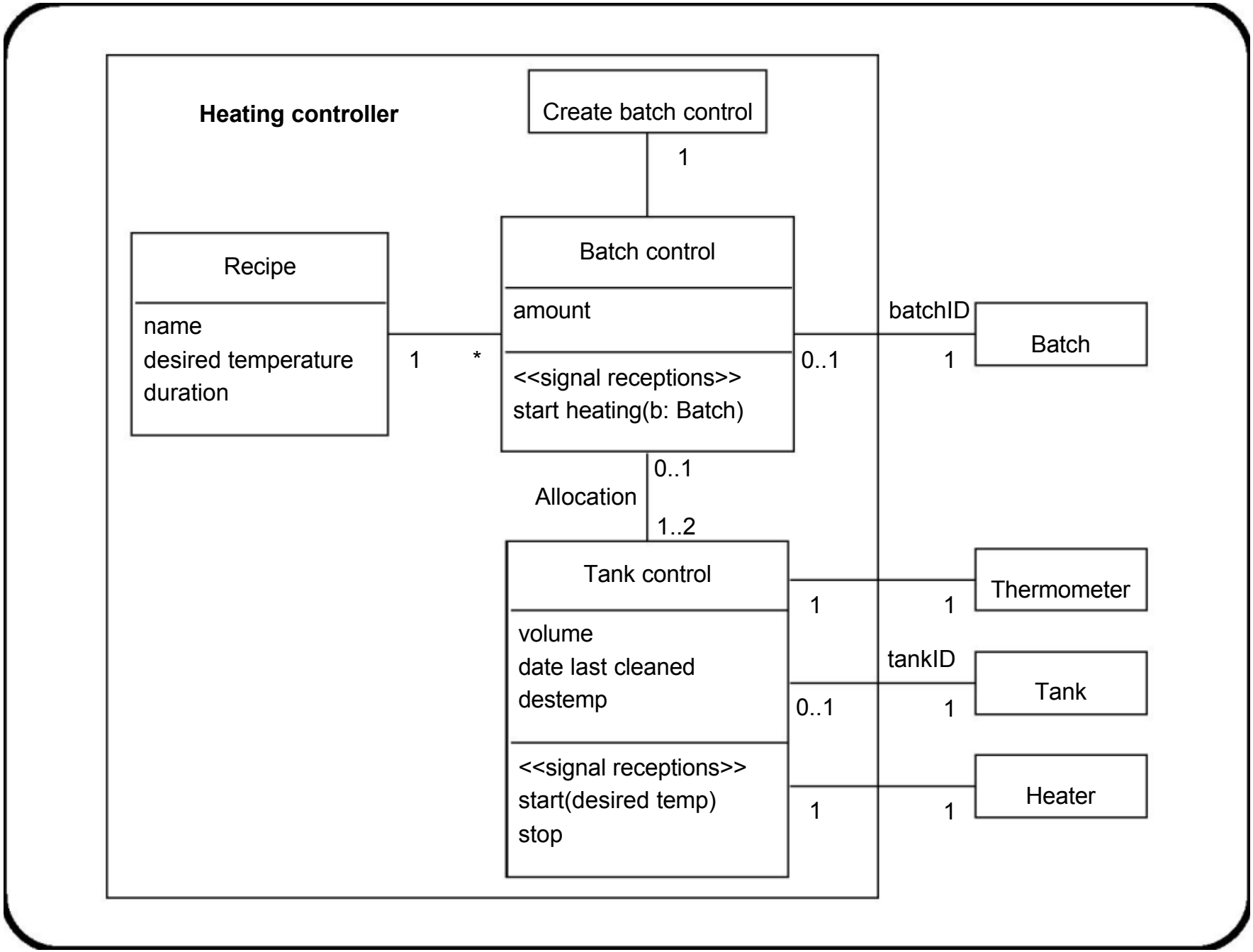
Subject domain of the heating controller



Guidelines used to design heating controller:

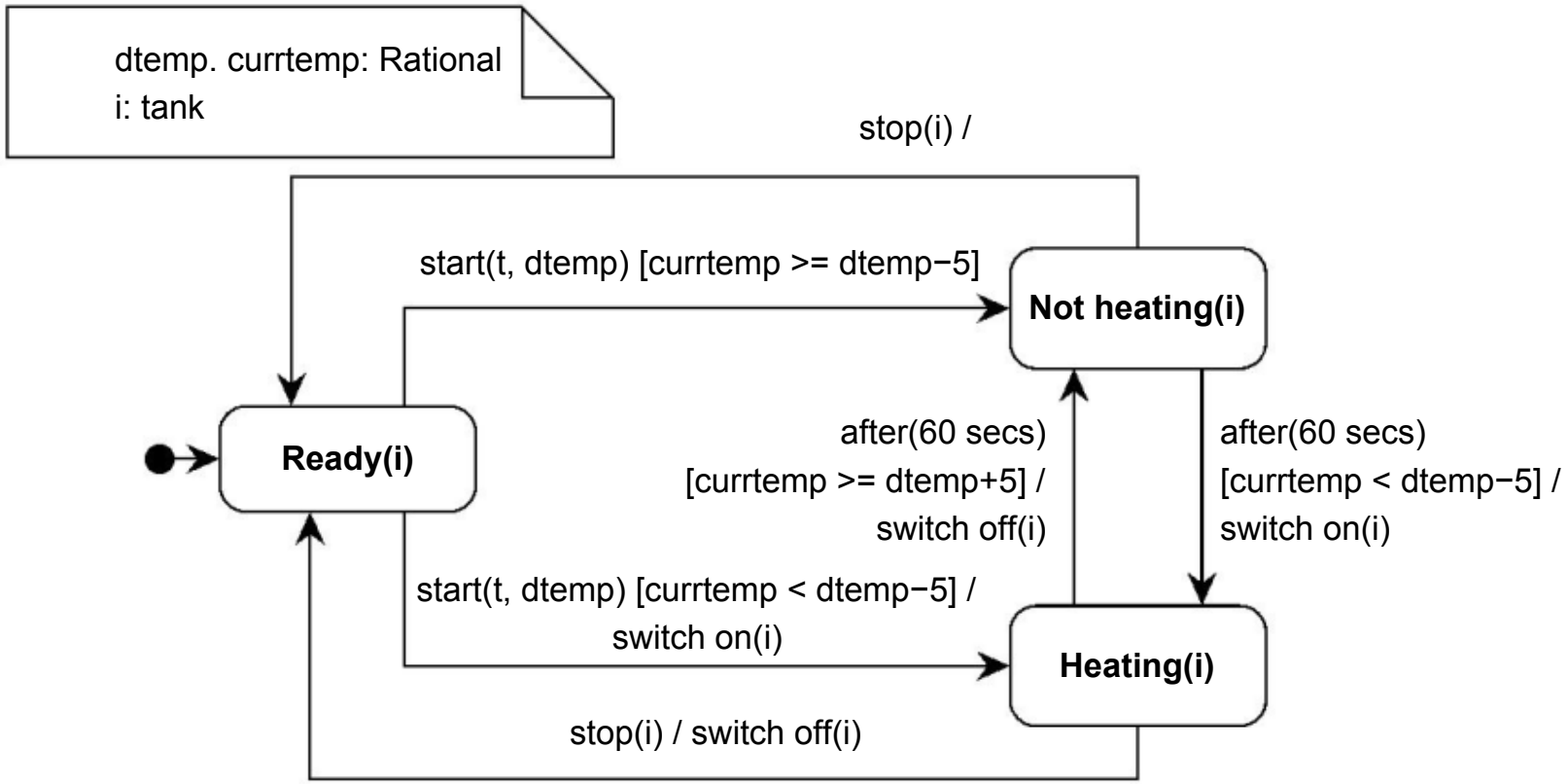
- Subject-orientation
- Functional decomposition
- Device-orientation

How are these guidelines used?


Heating controller
Create batch control
1
Recipe
name
desired temperature
duration
1
*
Batch control
amount
<<signal receptions>>
start heating(b: Batch)
0..1
batchID
1
Batch
0..1
Allocation
1..2
Tank control
volume
date last cleaned
destemp
<<signal receptions>>
start(desired temp)
stop
1
1
Thermometer
tankID
0..1
1
Tank
1
1
Heater

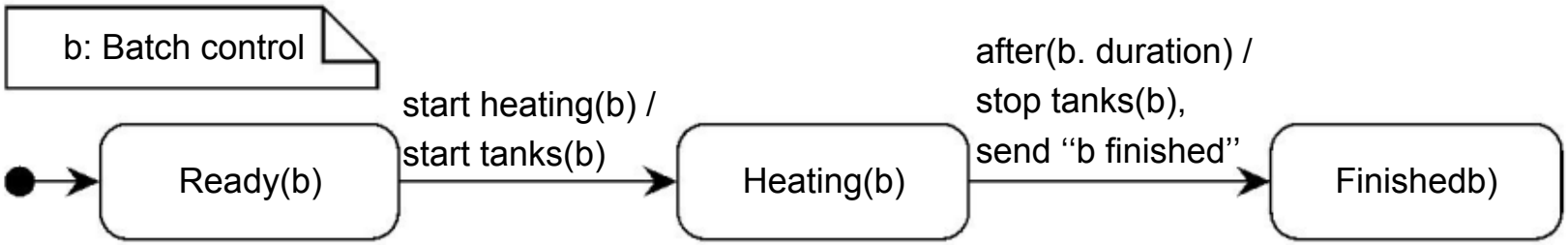
Statecharts in the UML (1)

Tank control



Statecharts in the UML (1)

Batch control

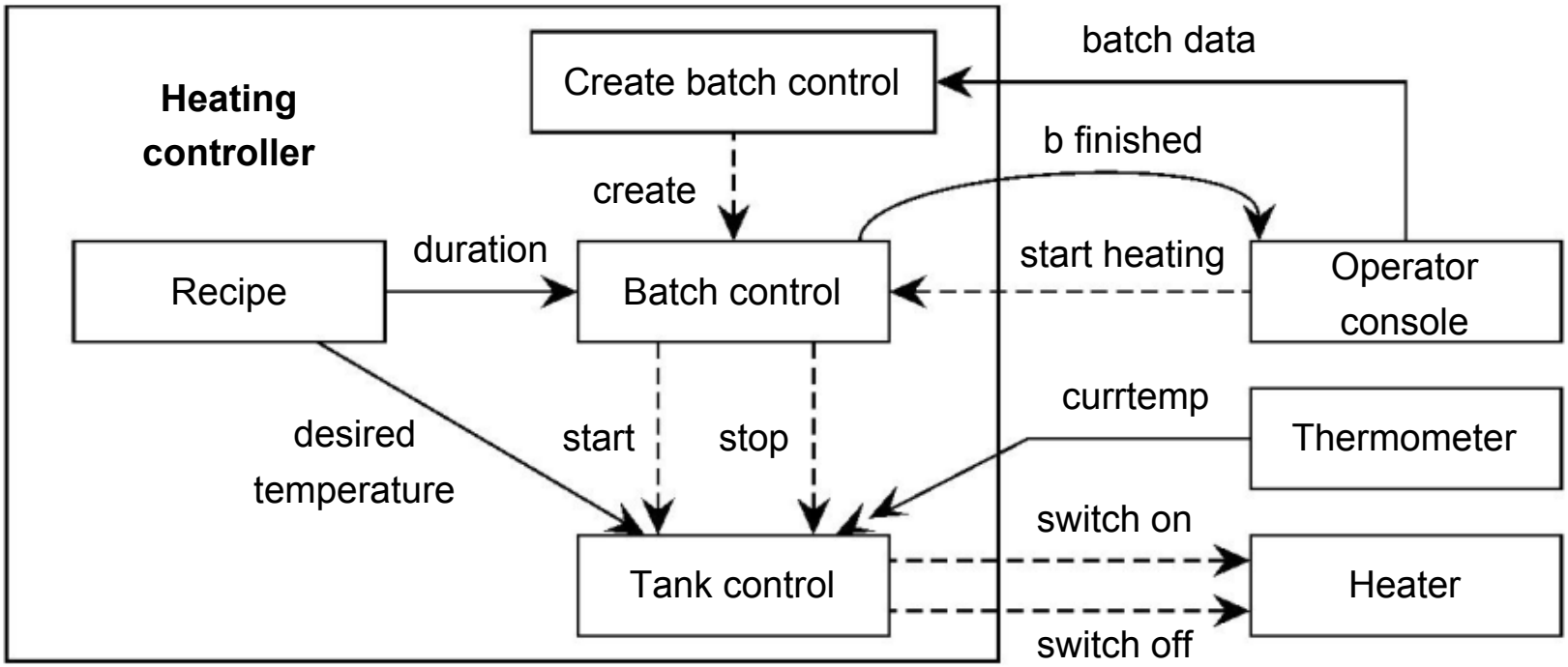


Dictionary:

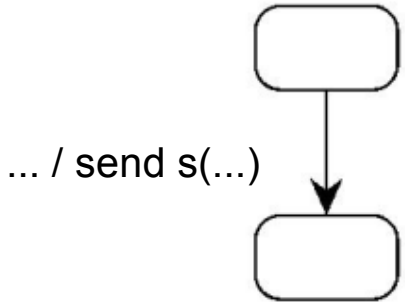
- **start tanks(b: Batch).**
  - For all tanks t in b.heating tank, send start(t, b.recipe .desired temperature).
- **stop tanks(b: Batch).**
  - For all tanks t in b.heating tank, send stop(t).

Coherence between SSD and statecharts (1)

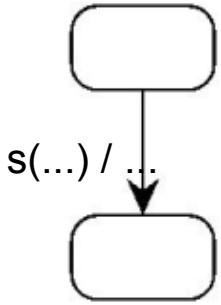
- Attributes, identifiers, interface data declared as local variables.
- Signals can trigger transitions.
- Actions can be operation calls or signal sending.
- Illustrate this coherence by an architecture diagram:



Coherence between SSD and statecharts (2)



Statechart of instances of A.

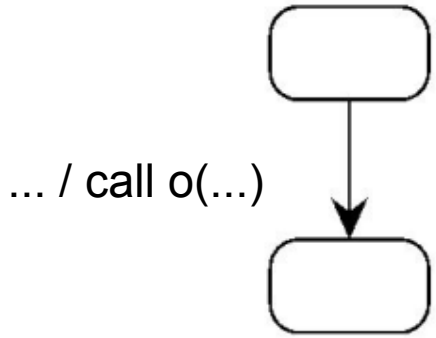


Statechart of instances of B.

Transitions can be triggered by s(...).

Instances of A can send messages to instances of B because there is an association from A to B.

Coherence between SSD and statecharts (3)



Statechart of instances of A.

Operation o must be defined for instances of B.

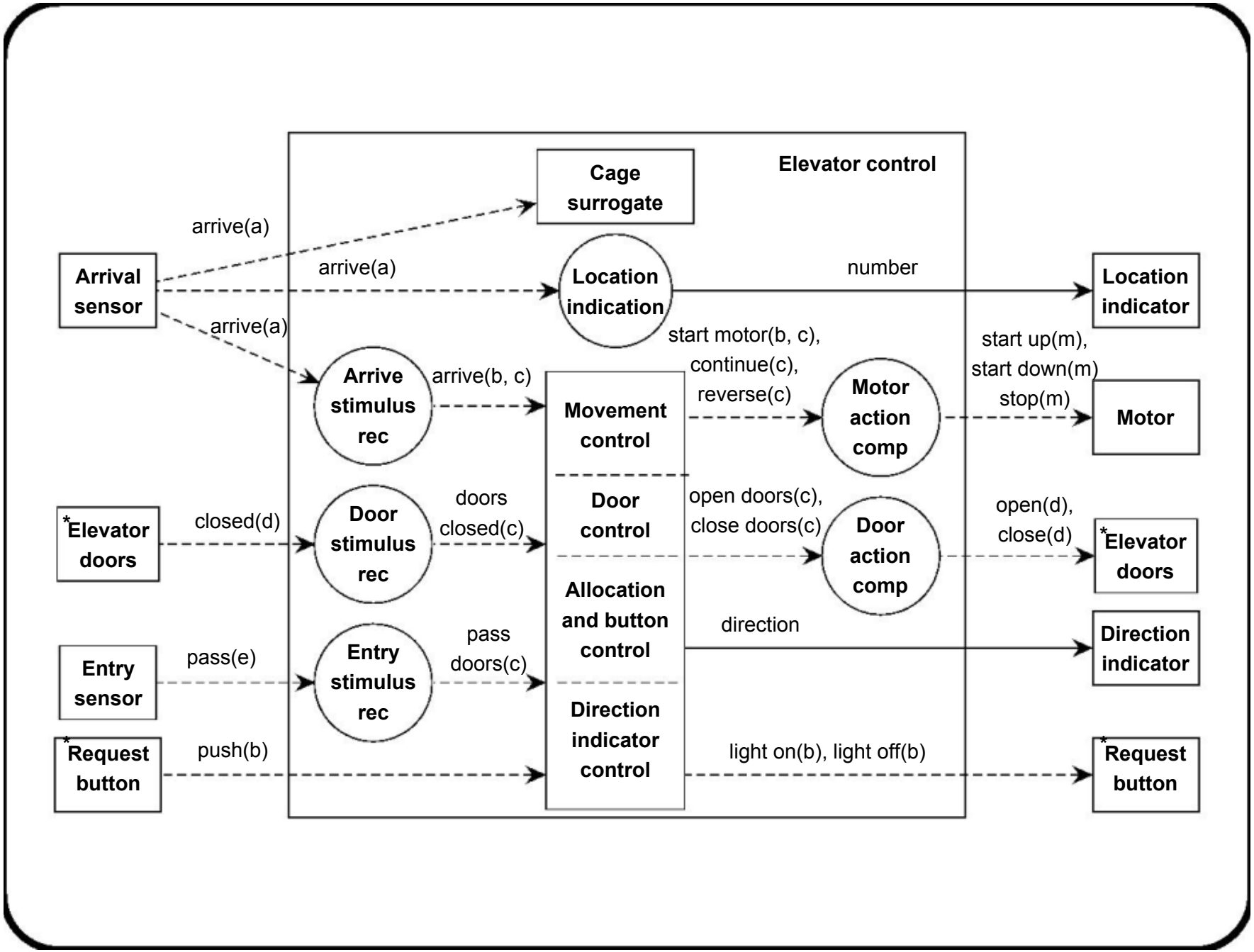
How to find an SSD

Considerations:

- A communication diagram shows communication channels among components.
- An SSD shows cardinality properties of components and access paths between classes.
- The SSD shows the access paths needed by each object to do its job.
- To find the access paths, we need to find out the job of each object first.
- To find the job of each object, we draw the communication diagram first.

Guidelines for finding an SSD

- √ Consider one stimulus-response pair.
- √ Draw a communication diagram of the stimulus-response process.
  - Use architecture guidelines to find components. See section 19.4.
- √ Draw an SSD of the classes and access paths needed by each component in the communication diagram.
  - Definition of operations tell us where each object must find its data.
  - Use subject domain ERD as inspiration to find multiplicity properties.

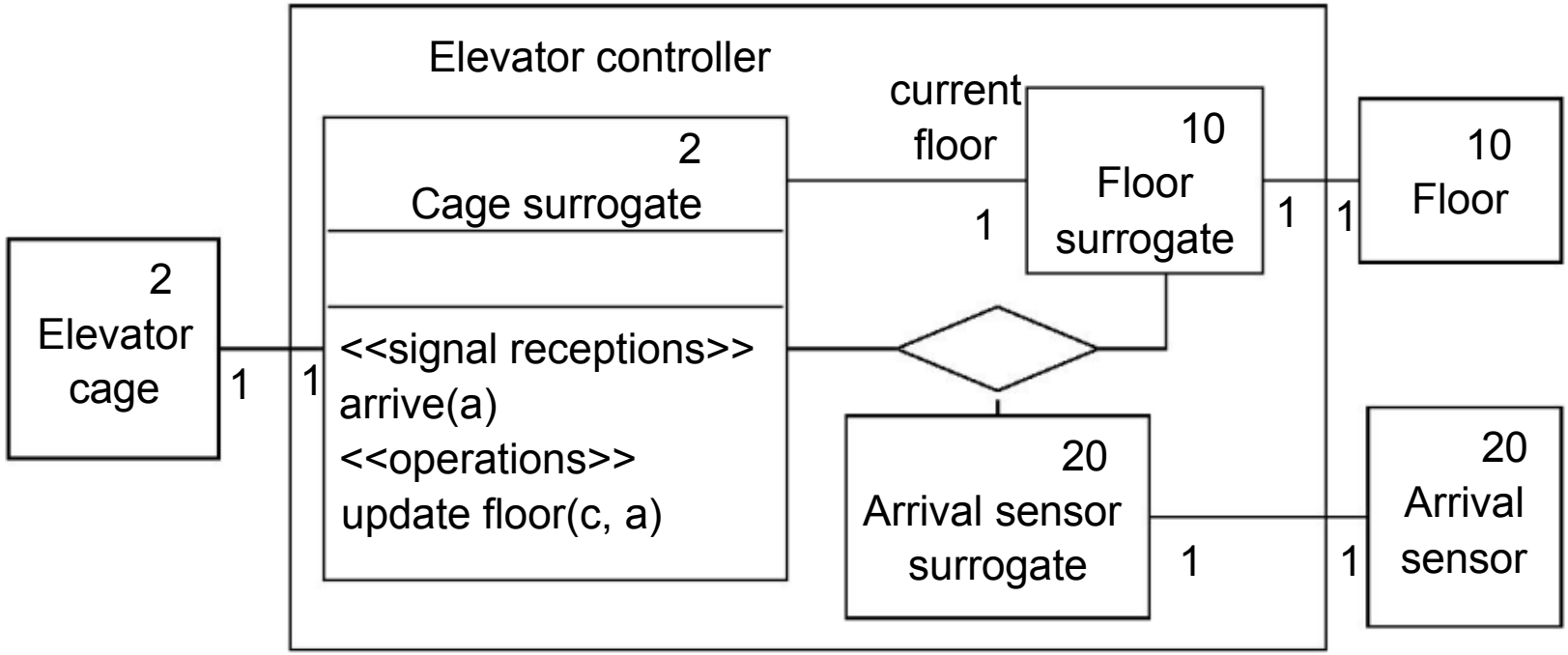

Elevator control
Cage surrogate
Arrival sensor
arrive(a)
arrive(a)
arrive(a)
Location indication
number
Location indicator
Arrive stimulus rec
arrive(b, c)
Movement control
start motor(b, c),
continue(c),
reverse(c)
Motor action comp
start up(m),
start down(m)
stop(m)
Motor
*Elevator doors
closed(d)
Door stimulus rec
doors
closed(c)
Door control
open doors(c),
close doors(c)
Door action comp
open(d),
close(d)
*Elevator doors
Allocation and button control
direction
Direction indicator
Entry sensor
pass(e)
Entry stimulus rec
pass
doors(c)
Direction indicator control
*Request button
push(b)
light on(b), light off(b)
*Request button

What information does a cage surrogate need? Here is its job:

c: Cage surrogate

arrive(a) / update floor(c, a), where update floor(c, a) = c.current floor := a.floor.

Here are the access paths needed:

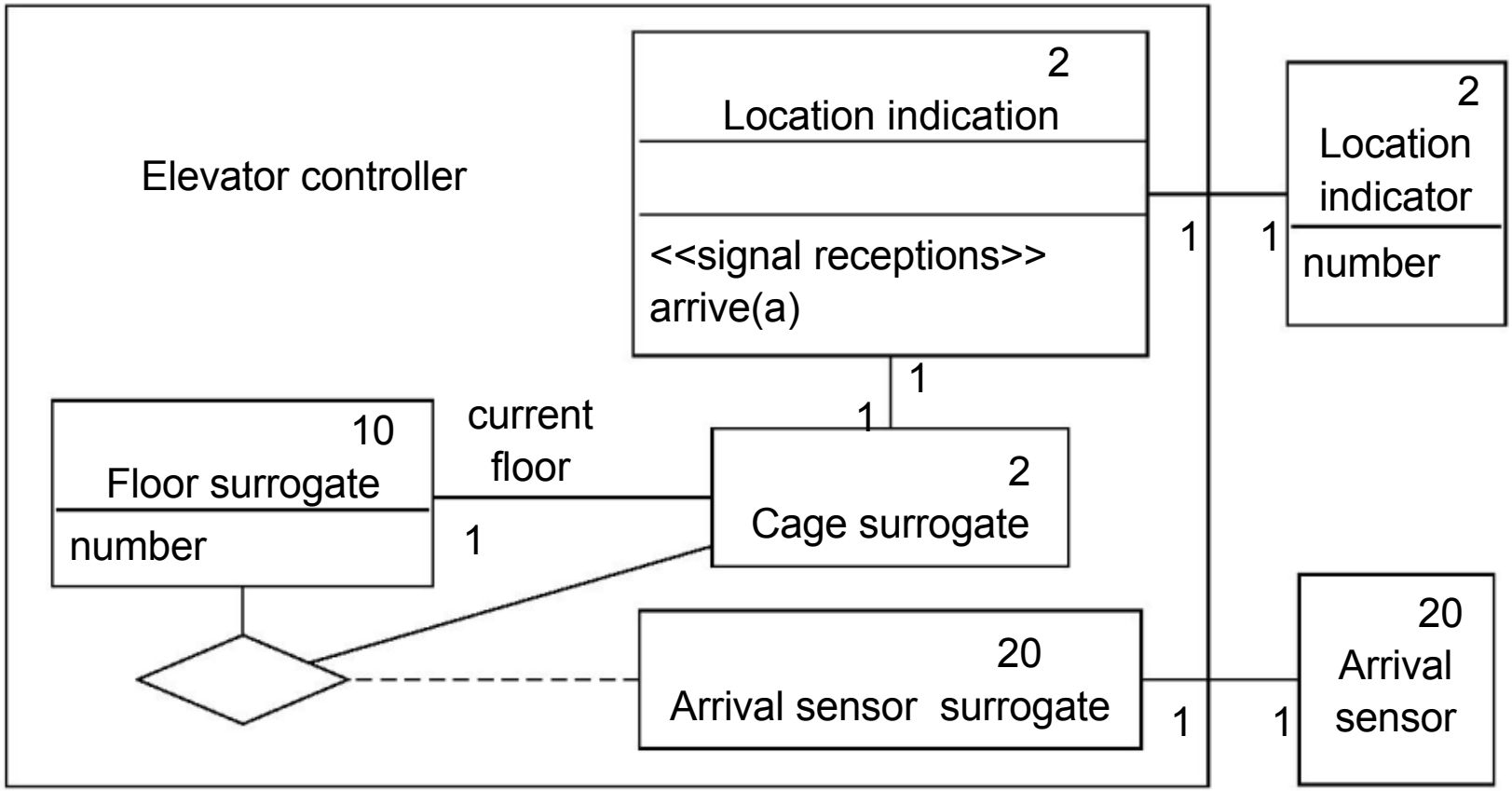


Here is the job of the location indicator:

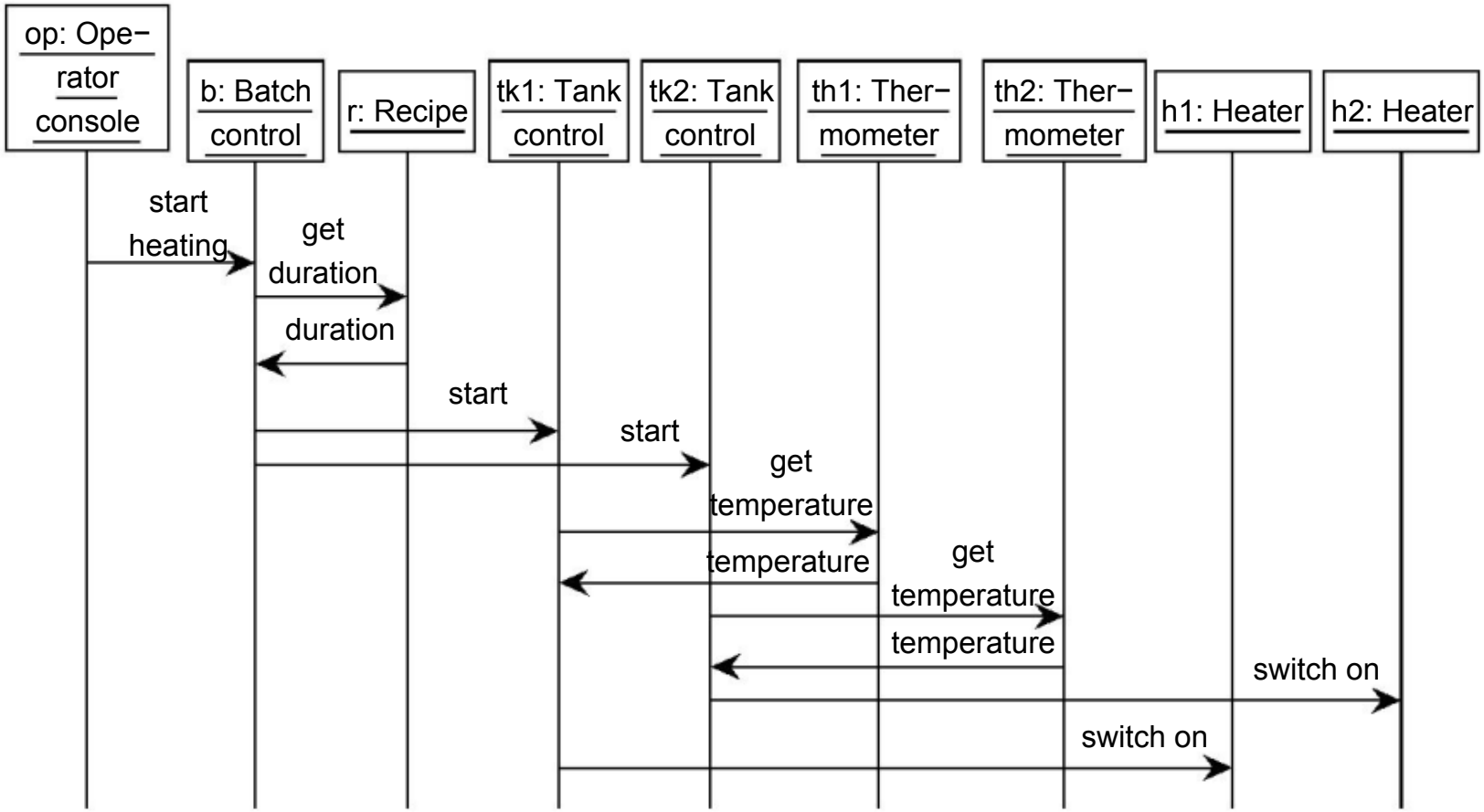
Location indication:

- When arrive(a),
- do set number(a.cage.location indicator, a.cage .current floor.number).

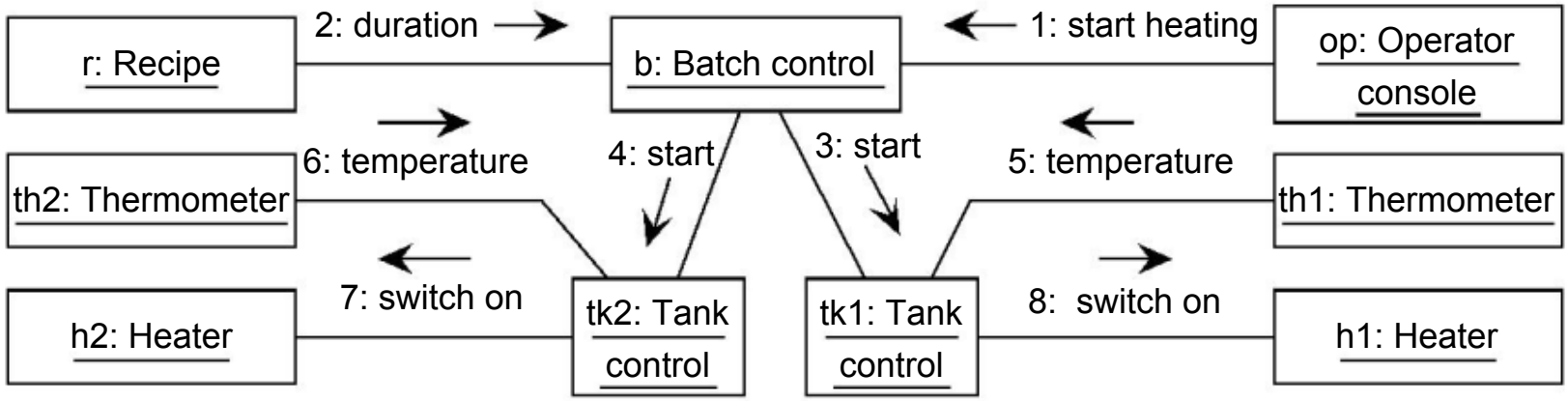
Here are the access paths through which it gets its information:



Sequence diagrams can be used to illustrate scenarios



Collaboration diagrams can be used to illustrate scenarios too



Possible uses of sequence and collaboration diagrams

- As illustration: The system must be able to execute this scenario.
- As specification: All executions of the system must contain this scenario.
- To specify patterns: The diagram represents roles that objects in the system play.

Main points

- Activity diagrams can be used to specify user workflow.
- Class diagrams represent decomposition of system into objects.
- Statecharts can be used to represent object life cycles.
- Use architecture diagrams to keep track of relationship between SSDs and statecharts.
- Find SSD by identifying access paths needed in stimulus-response processing.
- Illustrate executions by means of sequence or collaboration diagrams.

Chapter 23. Not Yet Another Method

The weight of professional boxers is classified according to the following scheme:

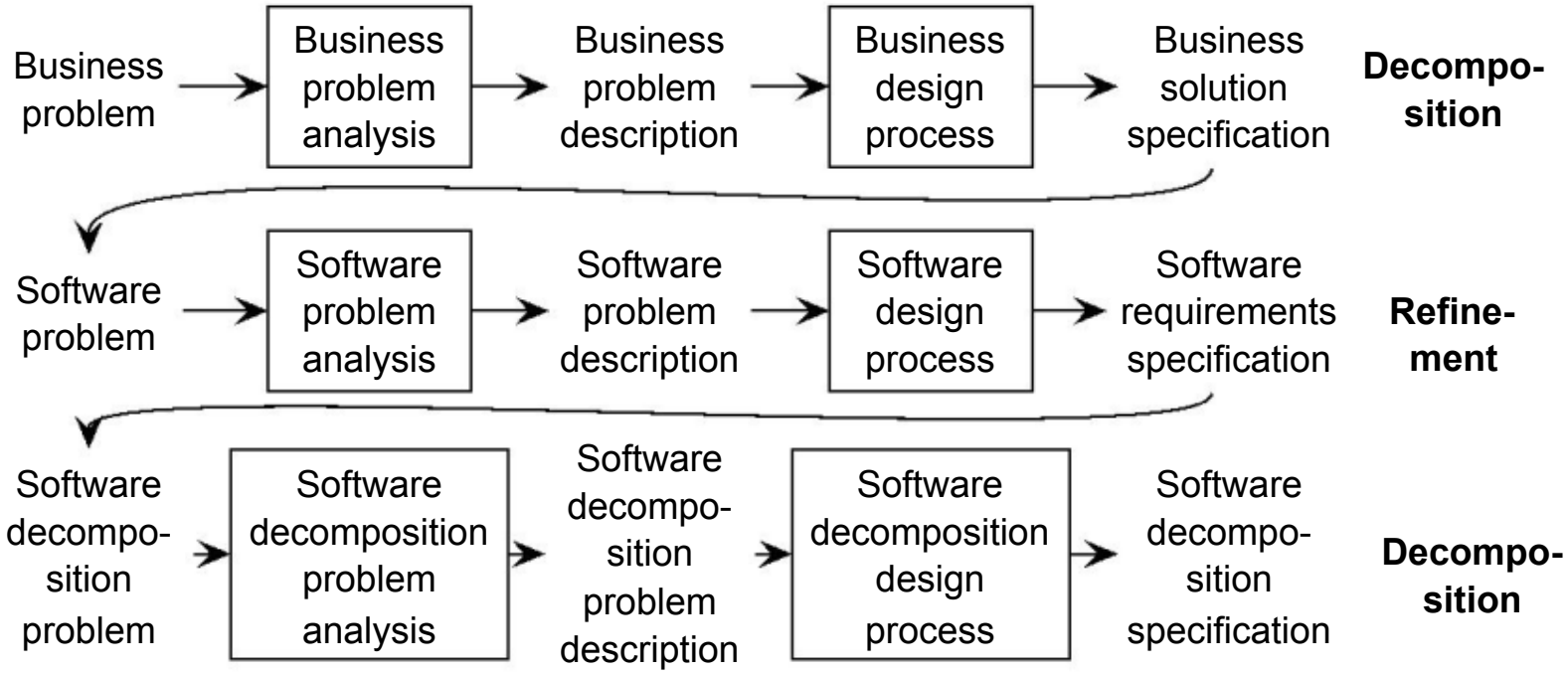
flyweight	$\leq$ 112 pounds
bantamweight	$\leq$ 118 pounds
featherweight	$\leq$ 126 pounds
lightweight	$\leq$ 135 pounds
welterweight	$\leq$ 147 pounds
middleweight	$\leq$ 160 pounds
heavyweight	$>$ 160 pounds

We can learn two things from this classification:

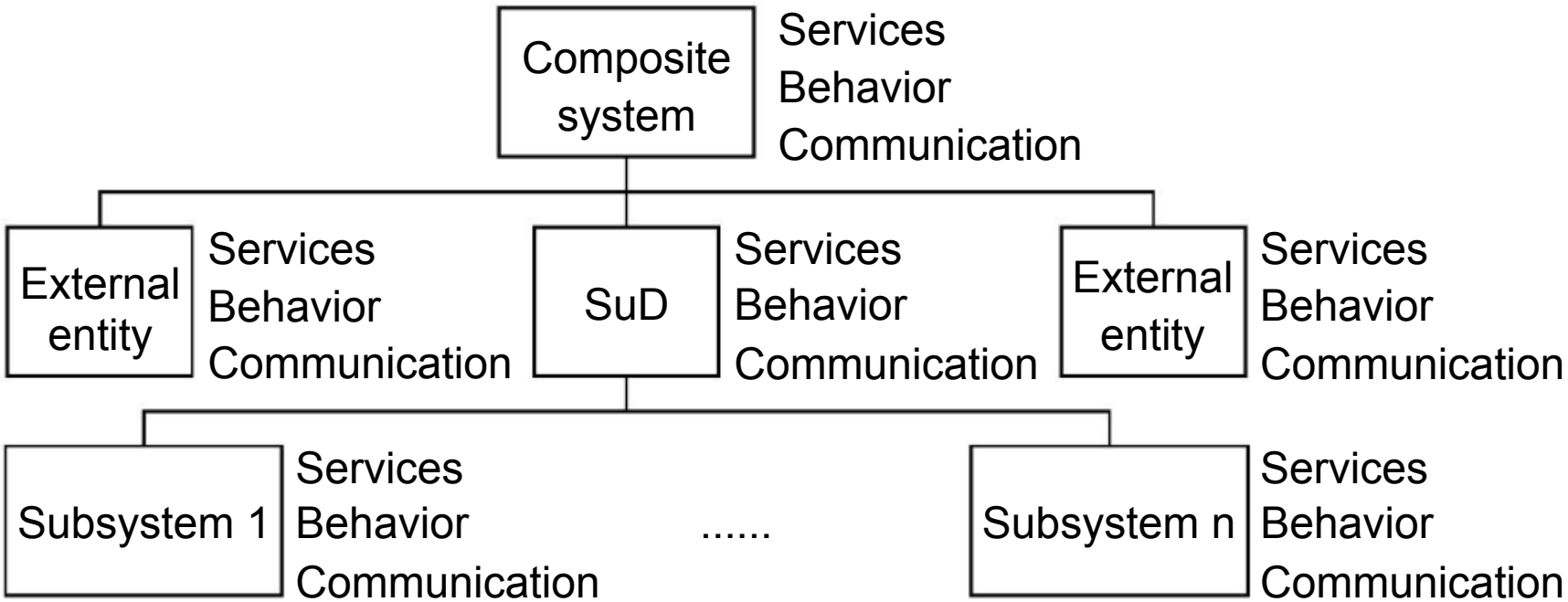
1. Lightweight is not the lightest weight.
2. Heavyweights can be as heavy as they want.

Keep it simple.

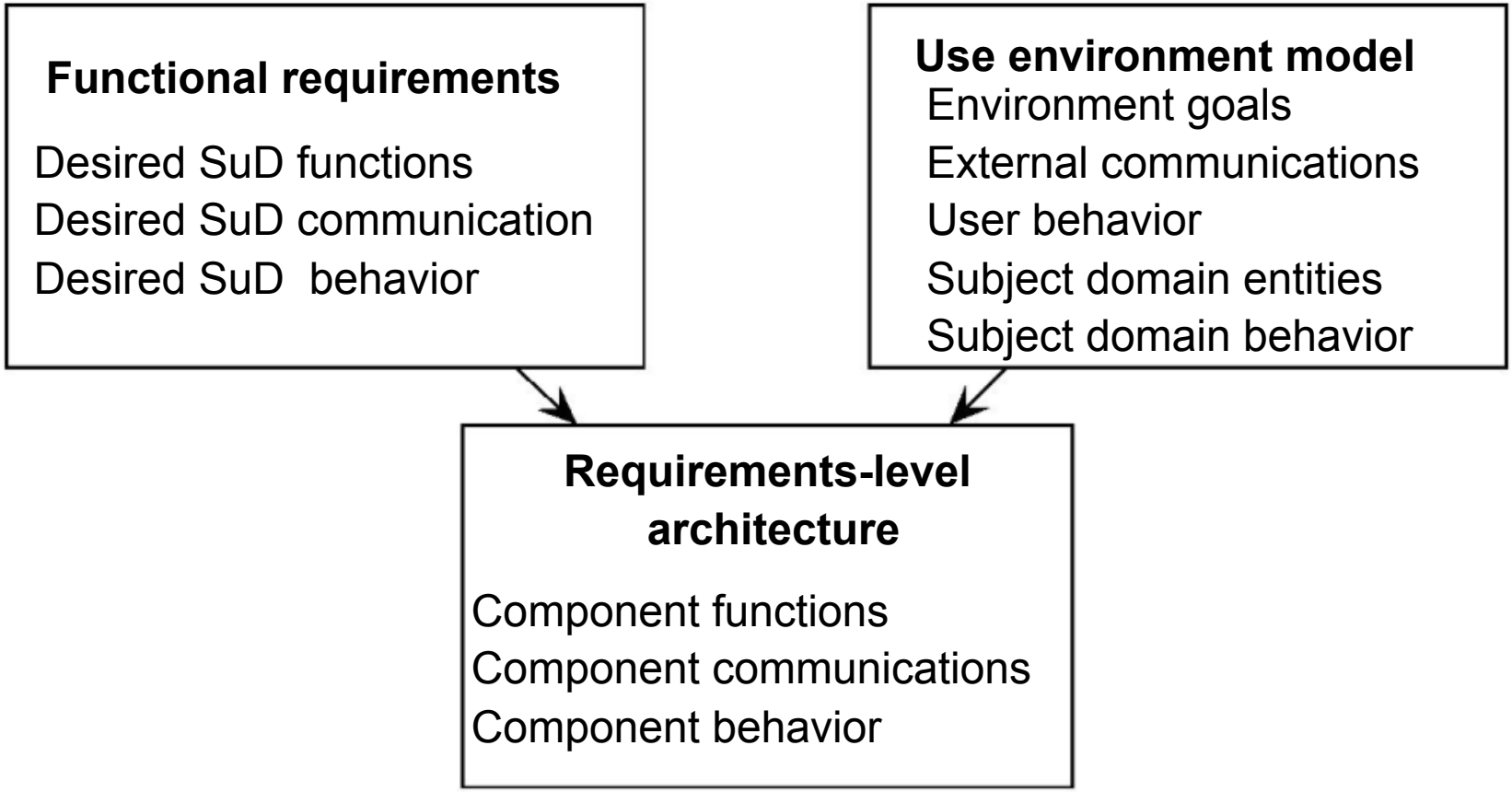
Problem-solving levels



The system hierarchy



Software design approach



- Requirements-level decomposition.
- Independent from changes in implementation platform.

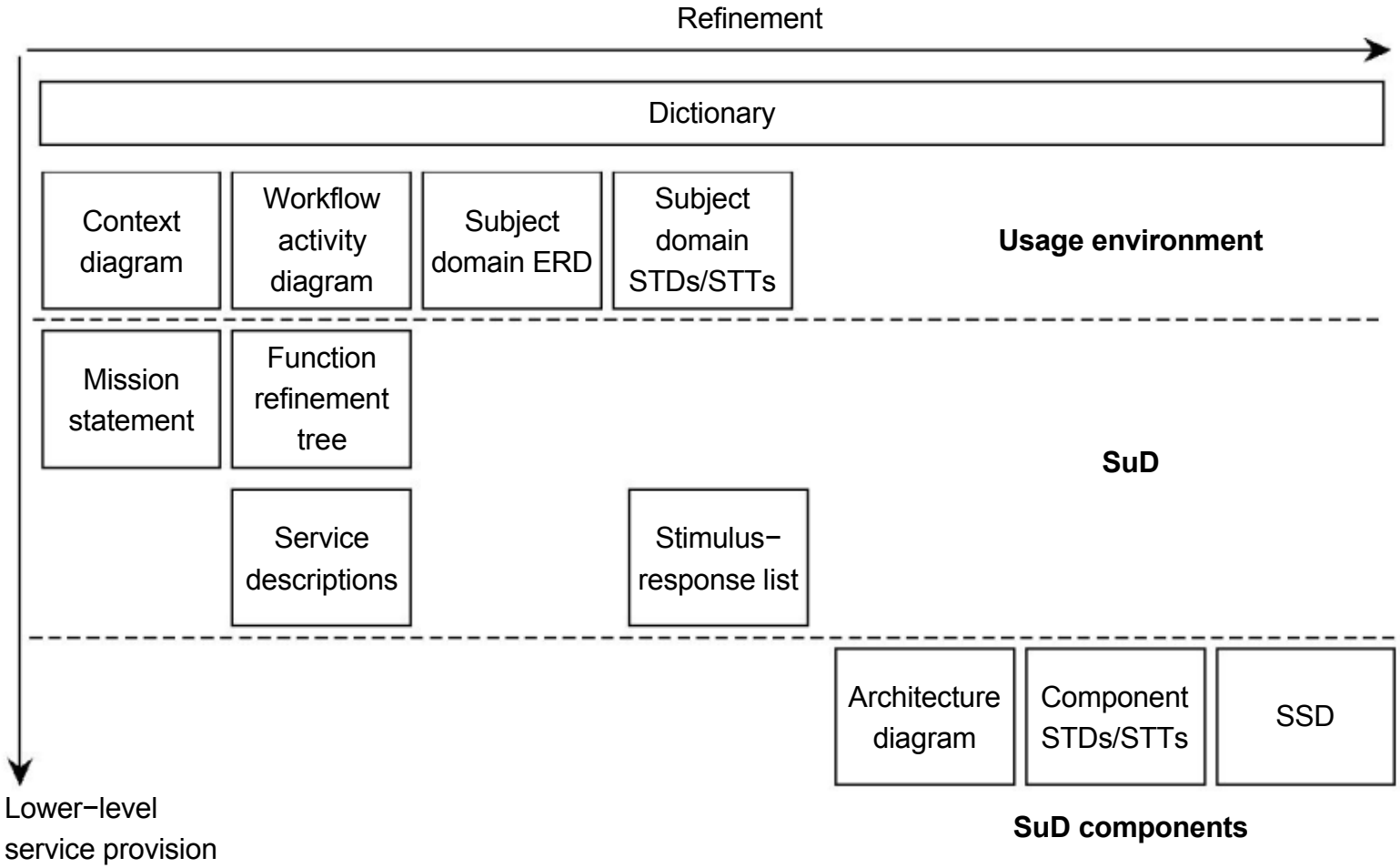
Notations treated in the book

	Functions	Behavior	Communication	Decomposition
Mission stmnt	X			
Funct. refinmnt tree	X			
Service descr.	X			
State trans. list		X		
State trans. table		X		
State trans. diagr.		X		
Activity diagram		X		
Data flow diagr.			X	
Arch. diagram			X	
ERD				X
SSD				X

Our use of notations

Use environment of SuD	• Context diagram • Subject domain ERD • Activity diagram of user workflow • STT or STD of desired subj. dom. behavior • STT or STD of assumed subj. dom. behavior
Functional specification of SuD	• Mission statement • Function refinement tree • Service descriptions • Stimulus-response descriptions
Decomposition of SuD	• Architecture diagram • STT or STD of component behavior • Static structure diagram

Ordering of design decisions



Often, in order to make a lower-level design model, the designer has to first make a more refined higher-level description.

- *Problem bounding.* Environment modeling.
- *Service description.* System design.
- *Defining key terms.* Environment modeling.
- *Identifying desired and assumed behavior.* System design.
- *Decomposition.* System design.

Engineering arguments

- Engineers predict product properties from product specification.
- Tinkerers fiddle around with the product and discover how it behaves.

Feed forward versus feedback loop.

$\Rightarrow$ We use product descriptions, among others, to produce engineering arguments. Ranges from very informal to very formal.

Using engineering knowledge:

- Products with this kind of decomposition usually have properties P;
- Since this product will have this kind of decomposition,
- It will probably have properties P.

Using throw-away prototyping:

- Since the prototype has properties P,
- And the prototype is similar to the final product,
- The final product probably has properties P.

Using model execution:

- Since the model execution has properties P,
- If the system implements this model exactly,
- Then the system will have properties P.

Using model checking:

- Since the state transition graph has properties P,
- If the system implements this graph correctly,
- the system will have properties P.

Using theorem-proving:

- Since the decomposition has been proved to have properties P,
- If the system correctly implements this decomposition,
- The system will have properties P.

Formality versus precision

- A description is **precise** if it expresses as briefly as possible what is intended.
- It is **formal** if it uses a language for which formal, meaning-preserving manipulation rules have been defined.

$\Rightarrow$

- Formal descriptions can be very imprecise: Statechart with superfluous transitions and states.
- Precise descriptions can be very informal: Function descriptions.

The attempt to be precise has priority over the attempt to be formal.

C.J. Smith, *Synonyms Discriminated.* G. Bell and Sons, Ltd., 1926.

- PRECISE denotes the quality of exact limitation, as

  distinguished from the vague, loose, doubtful, inaccurate; ... The idea of precision is that of casting aside the useless and superfluous.
- EXACTNESS is that kind of truth which consists in conformity

  to an external standard or measure, or has an internal correspondence with external requirement ... an exact amount is that which is required."
- ACCURACY, by contrast, refers to the attention spent upon a

  thing, and the exactness which may be expected from it. Accuracy is designed whereas exactness may be coincidental.
- CORRECTNESS, finally, applies to what is conformable to a

  moral standard as well as to truth generally, as "correct behavior".

C.J. Smith, *Synonyms Discriminated.* G. Bell and Sons, Ltd., 1926.

> "It is most desirable that men should be exact in duties and obligations, accurate in statements and representations, correct in conduct, and precise in the use of words."

Omit needless words

W. Strunk Jr. & E.B. White, *The Elements of Style.* Fourth edition. Allyn and Bacon 2000.

Rule 17

> Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the writer make all sentences short, or avoid all detail and treat subjects only in outline, but that every word tell.