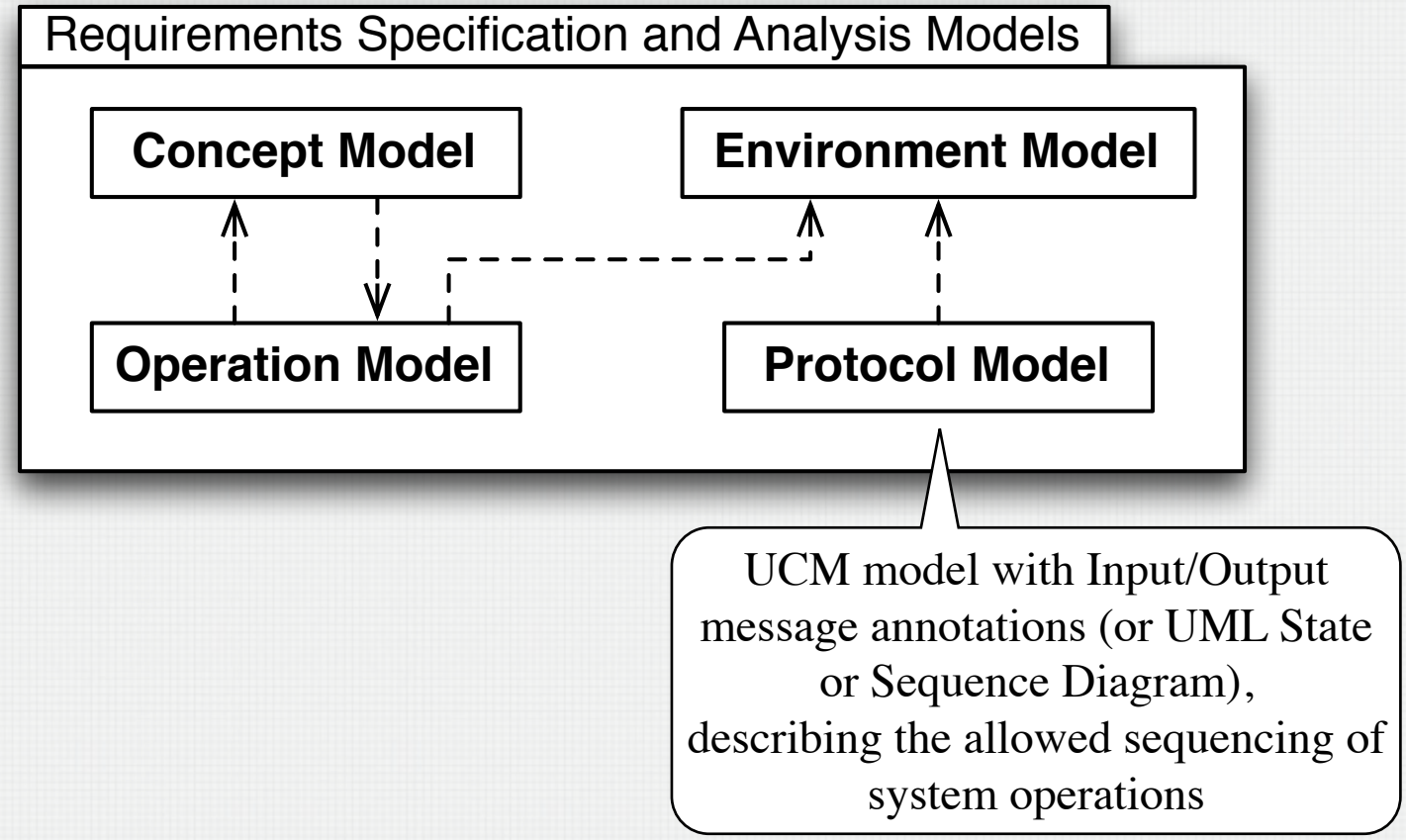
Protocol Model

Jörg Kienzle & Alfred Strohmeier

Behavioural Requirements Overview

- Protocol Model
  - User Requirements Notation
  - State diagrams
  - Sequence diagrams
- Checking Consistency of Requirements Models

Fondue Models: Requirements Spec.



Requirements Specification Phase

- Purpose
  - To produce a complete, consistent, and unambiguous description of
    - the problem domain and
    - the functional requirements of the system.
- Models are produced, which describe
  - Structural Models (see previous lecture)
  - Behaviour Models
    - Operation Model
      - Defines for each system operation the desired effect of its execution on the conceptual state
    - Protocol Model
      - Defines the system protocol, i.e. describes the allowed sequencing of system operations
  - The models concentrate on describing what a system does, rather than how it does it.

Protocol Model (1)

- The Protocol Model defines the allowable sequences of interactions that the system may have with its environment over its lifetime
- If at any point the system receives an event, either time-triggered or triggered by a message, that is not allowed according to the Protocol Model, then the system ignores the event and leaves the state of the system unchanged.
  - Note: A dependable system, instead of ignoring the message, should inform the environment about the “interaction error”
- The Protocol Model is depicted by one of the following diagrams
  - A UCM model with annotations that clearly identify when input messages occur in the flow
    - Output messages can optionally be shown as well
  - A state diagram (also called a statechart)
  - A sequence diagram (not explained in this lecture)

Protocol Model with UCMs

- The use case model already describes the interaction scenarios that the system needs to support
- Use case maps (UCMs) are a workflow notation that can depict any kind of workflow
  - Supports conditional and parallel execution, synchronization and timed synchronization
  - Hierarchical decomposition
  - Dynamic stubs
- We just need a way to designate which responsibilities correspond to input messages
  - Stereotypes can be used for that purposes

UCM Model with Annotations

- Responsibilities that represent input interactions are annotated with stereotype «in»
  - In jUCMNav, click on responsibility, then in the properties window on tab “metadata”, then “add”
    - Name: ST_Protocol, Value: in
- Responsibilities that represent output interactions are optionally annotated with stereotype «out»
  - Name: ST_Protocol, Value: out

From Use Cases to Protocol Model

- Each use case is mapped to a Use Case Map (UCM)
- Each interaction is mapped to a responsibility
  - Input interactions are tagged with stereotype <<in>>
  - Output interactions are tagged with stereotype <<out>>
  - Time-triggered events are mapped to timers
- The workflow of the scenario described in the use case is mapped to control flow elements in UCM
  - Ambiguities in the textual descriptions of the workflow are eliminated

RecycleItems Use Case

Use Case: RecycleItems
Scope: RecyclingMachine
Level: User-Goal
Intention in Context: The User wants to recycle bottles and cans in exchange for money.
Multiplicity: Only one User can recycle items at a given time.
Primary Actor: User
Secondary Actors: Recognizer, Display, FinishedButton, Printer
Main Success Scenario:

Step 1, 2 and 3 are repeated for each item the User wishes to recycle.

1. *User inserts a can or a bottle into the Recognizer*.
2. *Recognizer* informs *System* about the item that was recognized.
3. *System* acknowledges recognition to *User* by updating the refund total on the *Display*.
4. *User* informs *System* that s/he has no more items to process using the *FinishedButton*.
5. *System* prints receipt using *Printer*.

Extensions:

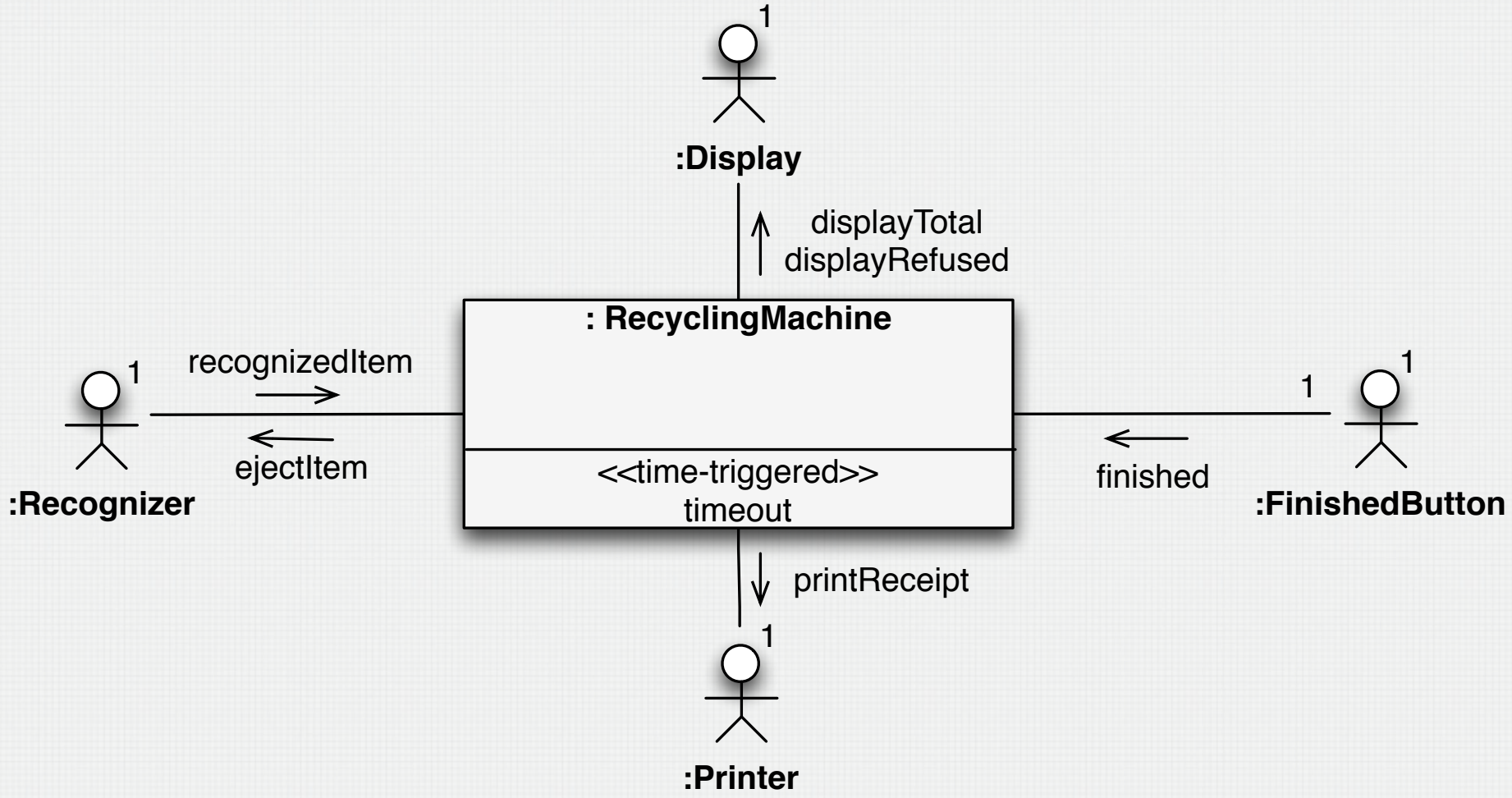
2a. Timeout occurs because user has not inserted any item into the recognizer for more than 2 minutes. Use case continues at step 5.

3a. System determines that the inserted item is not accepted at this store.

3a.1. *System* informs user that item is not accepted at this store using the *Display*.

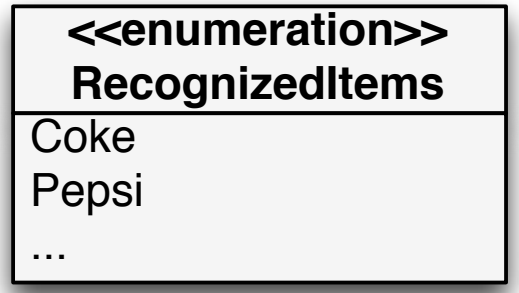
3a.2. *System* instructs the *Recognizer* to reject the inserted item. Use case continues with next item at step 1.

Recycling Machine Environment Model

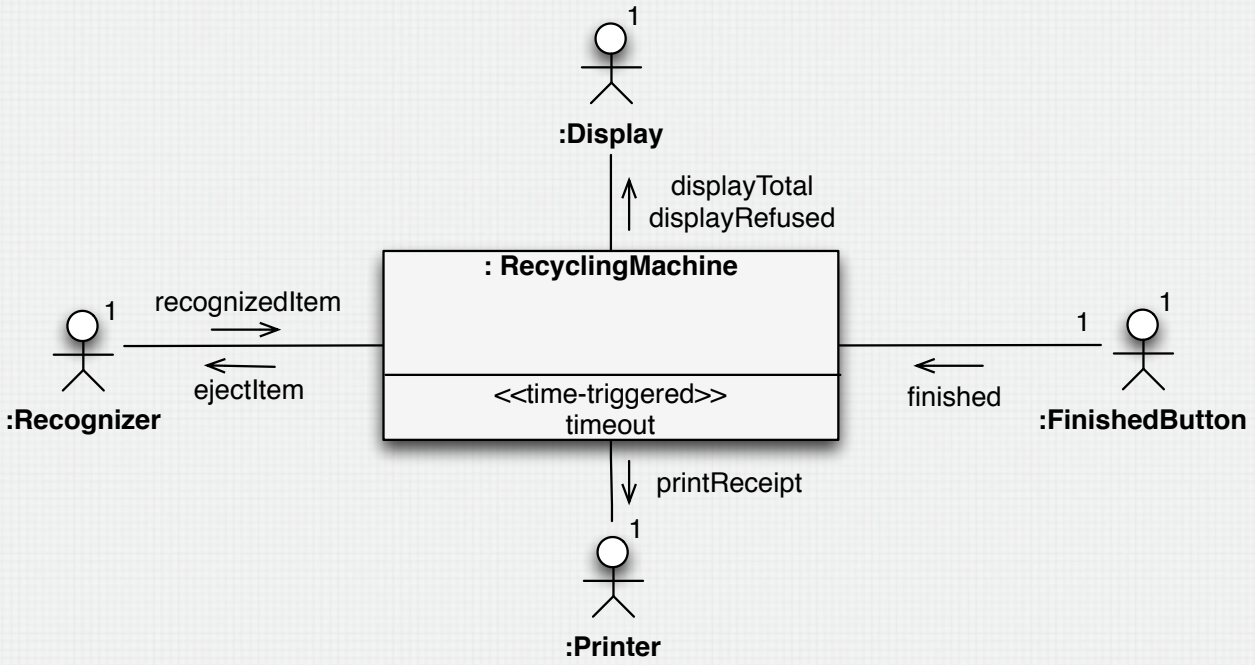


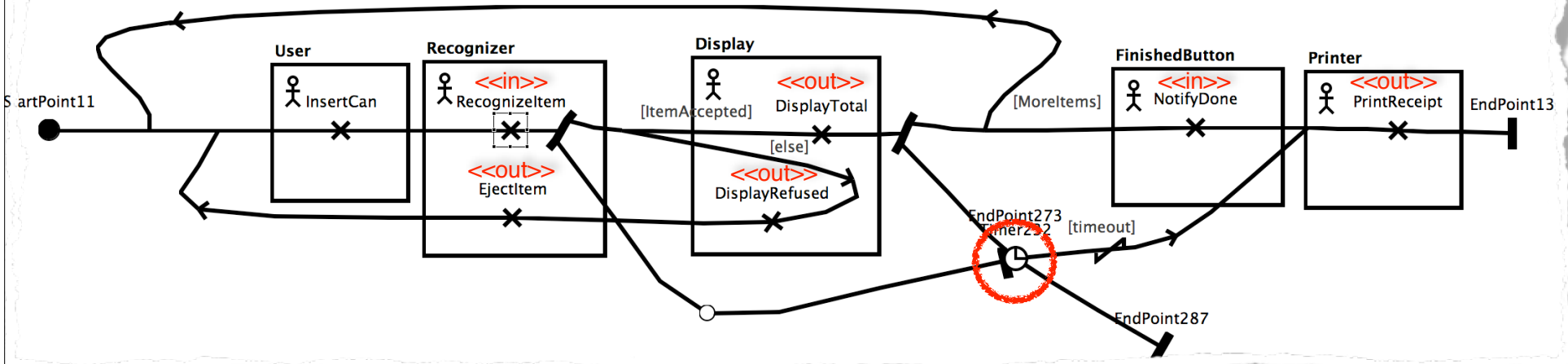
Recycling Machine Environment Model

- Input
  - RecognizedItem(item: RecognizedItems)
  - Finished
  - Timeout
- Output
  - EjectItem
  - DisplayTotal(amount: Positive)
  - DisplayRefused(reason: String)
  - PrintReceipt(amount: Positive)



Recycling Machine Protocol Model

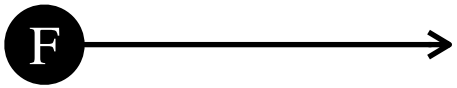




UCM - Failure Points

- Explicit approach with Failure Points
  - Indicates location of failure on scenario path
  - Failure condition
  - Sets failure variable (failure_name) to indicate which failure occurred

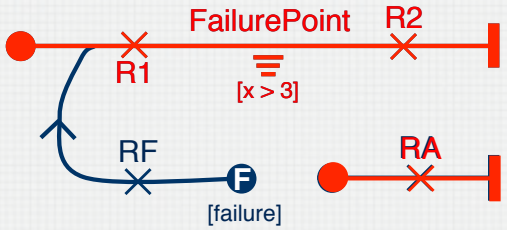
- Failure Start Point
  - Activated if guard (failure_name) evaluates to true



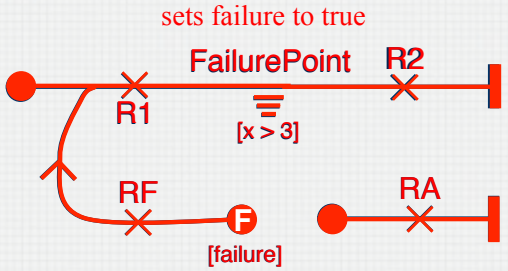
UCM - Failures Examples

- Example
  - RF:
    - Sets X to 0
    - Failure = false
    - Failure2 = false

Situation 1:
x = 1
No failure occurs,
workflows continue as normal

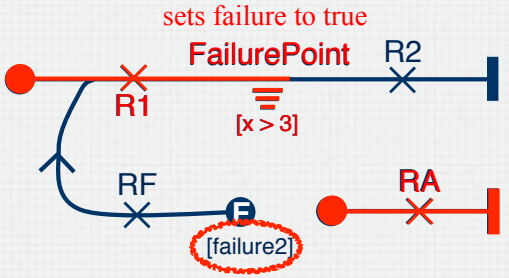


Situation 2:
x = 5
Failure occurs on first pass,
execution jumps to failure start point,
Second pass succeeds



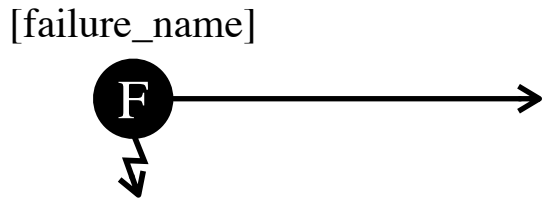
In all 3 cases, the execution of the path with RA is not affected!

Situation 3:
x = 5
Failure occurs on first pass,
but since no failure start point is defined
execution stops there

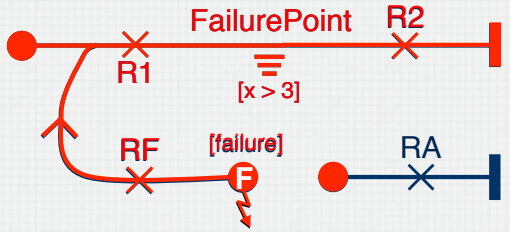


UCM - Abort Start Points

- Abort Start Points
  - Activated if guard (failure_name) evaluates to true
  - Aborts all other paths in the abort scope (all concurrent branches that are active on the same or lower level maps)

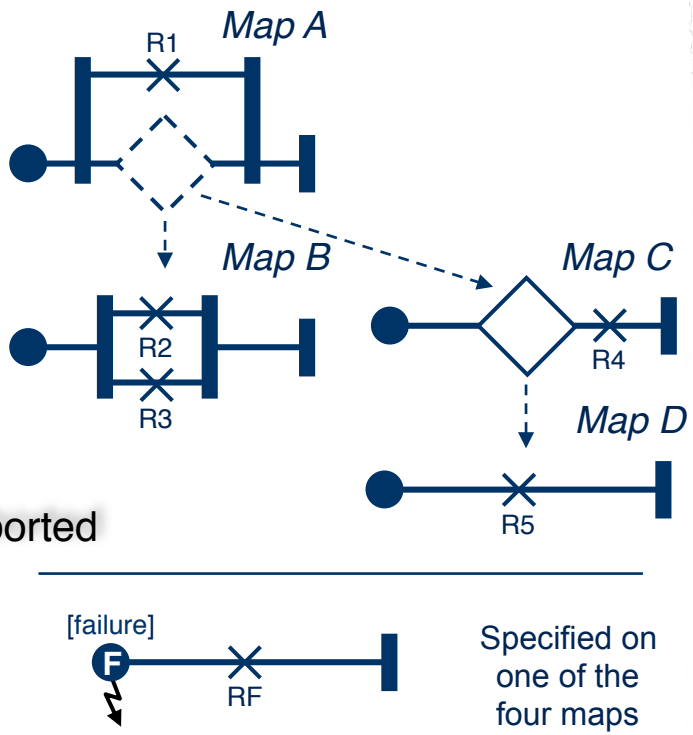


Situation 4:
x = 5
Failure occurs on first pass,
execution path containing RA is aborted,
execution continues at abort start point,
second pass successful



UCM - Implicit Failure Points

- Implicit approach without Failure Points
  - Location and failure variable defined by scenario definition
  - Greater flexibility in defining scope (path node, map, component reference, component/responsibility definition)
- Example (with map scope)
  - Failure occurs above map with the abort path
    - Result: not caught, scenario terminates with error
  - Failure occurs on or below map with abort path
    - Scenario ends with end point of abort path (unless end point is bound to out-path of a stub)
    - Abort path specified on map A
      - Result: all paths on all maps are aborted
    - Specified on B
      - Result: only the two concurrent paths on B are aborted
    - Specified on C
      - Result: all paths on C and D are aborted;
    - Specified on D
      - Result: only the path on D is aborted

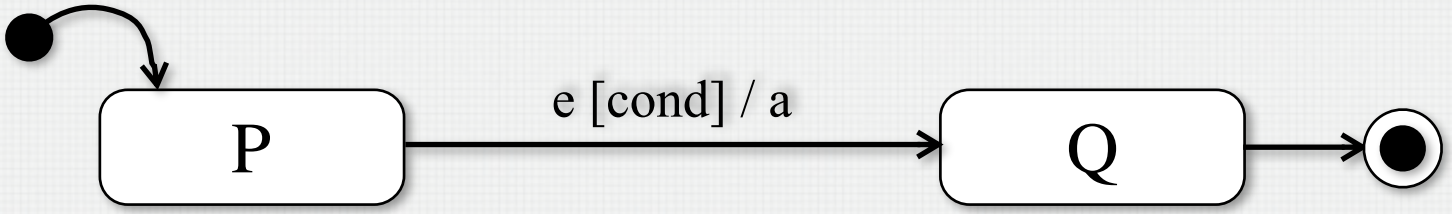


Protocol Model with State Diagrams

- In case the system protocol is rather “modal”, state diagrams can be used to model the order of input events
  - In UML 2.0 terminology these kind of state diagrams are called protocol state machines
- We will only use simplified state diagrams, because
  - We want to remain at a high level of abstraction during analysis;
  - A lot of information is already included in the Concept Model, the Environment Model and in the Operation Model;
  - We don’t want to duplicate information.
    - For example, it is not necessary to show guard conditions, because they are included in the operation schema as preconditions or if-then-else expressions.

General State Diagram

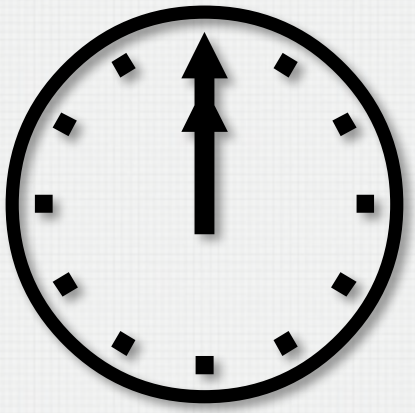
- Initially, the system starts in State P
- If the event e is received, and the guard cond is true, then action a is taken and the current state switches to State Q
- Immediately after than, the system terminates (reaches the final state)

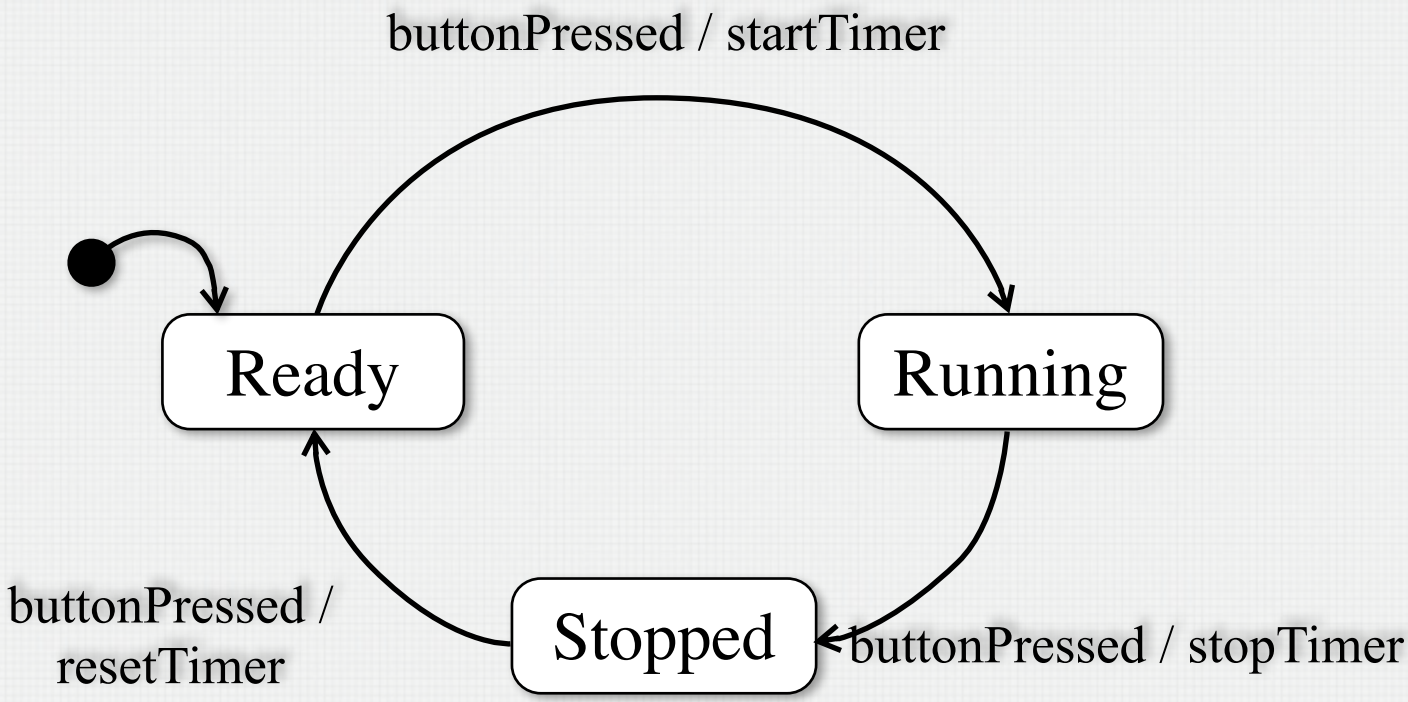


Protocol Model (3)

- In our case, the events designate input events. It is the input event that triggers the transition and the action that follows.
- An OperationName is provided as an Action, if the name is different from the input event that triggers the transition, otherwise it can be omitted
- We show only the event names, because the event parameters are declared with the declaration of the time-triggered event or of the message that triggers the event (and they are the same), and the source actor is documented in the operation schema.

Simple State Diagram Example



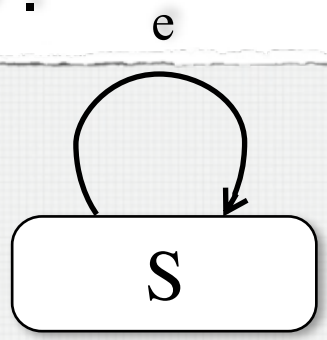


Sequencing

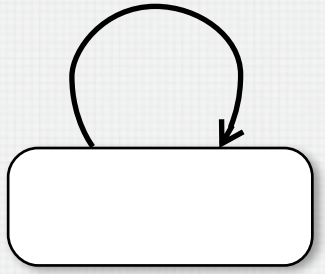
- Sequence: Event e1 leads to state S and is followed by event e2. The actions triggered by e1 and e2 have to occur in sequence.

Loops

- Repetition or loop: Event e occurs repeatedly leaving the system in the same state. Note that the concept of state here is at a high level with the meaning “the same events are acceptable”.

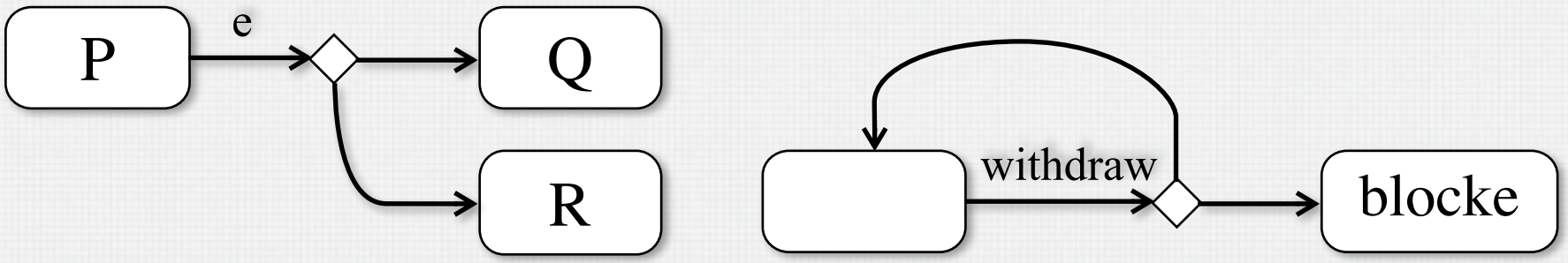


checkBalance



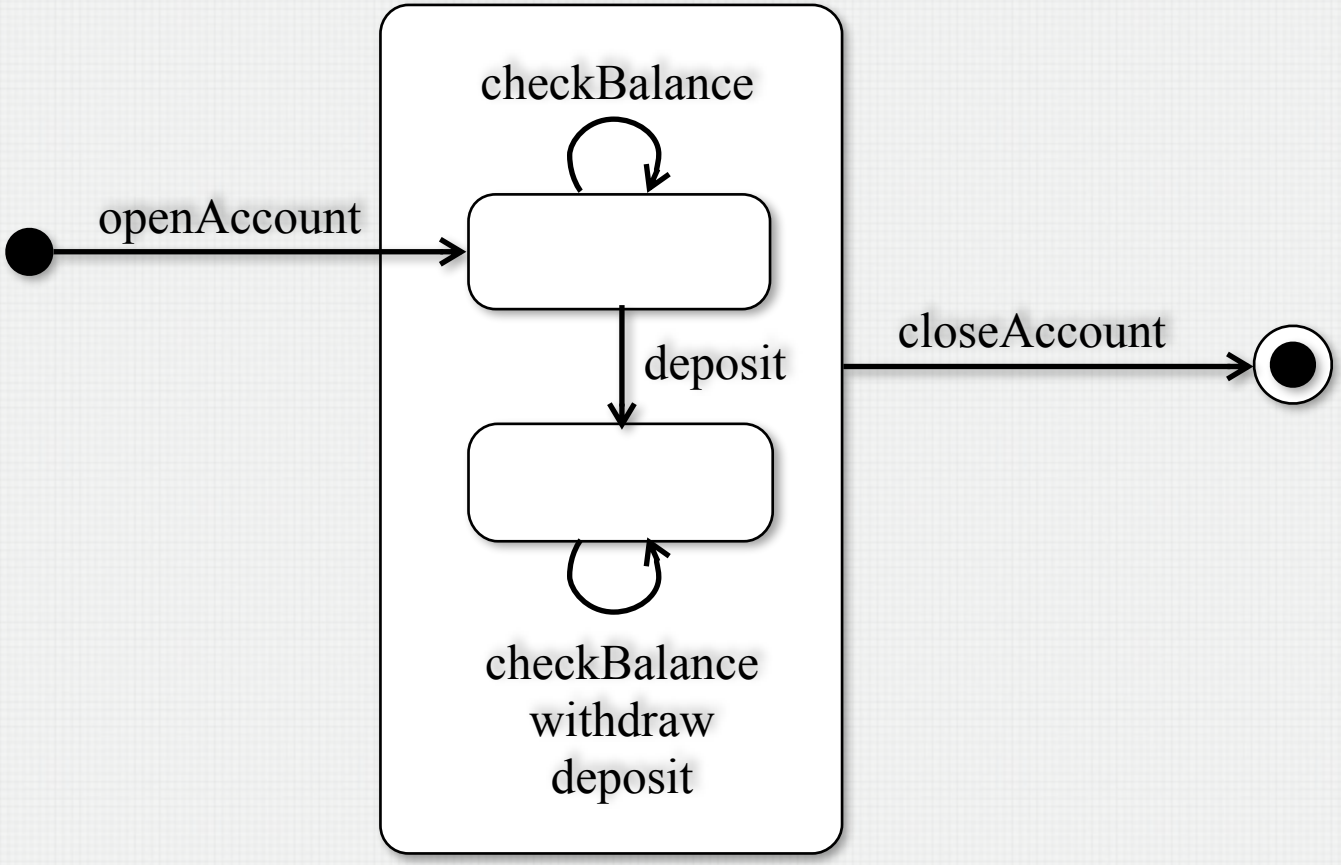
Alternative

- Alternative: Event e can lead to two different states, depending on the current state values.
  - In the example, depending on the available balance, withdrawal might be possible, or be rejected and result in a blocked account.



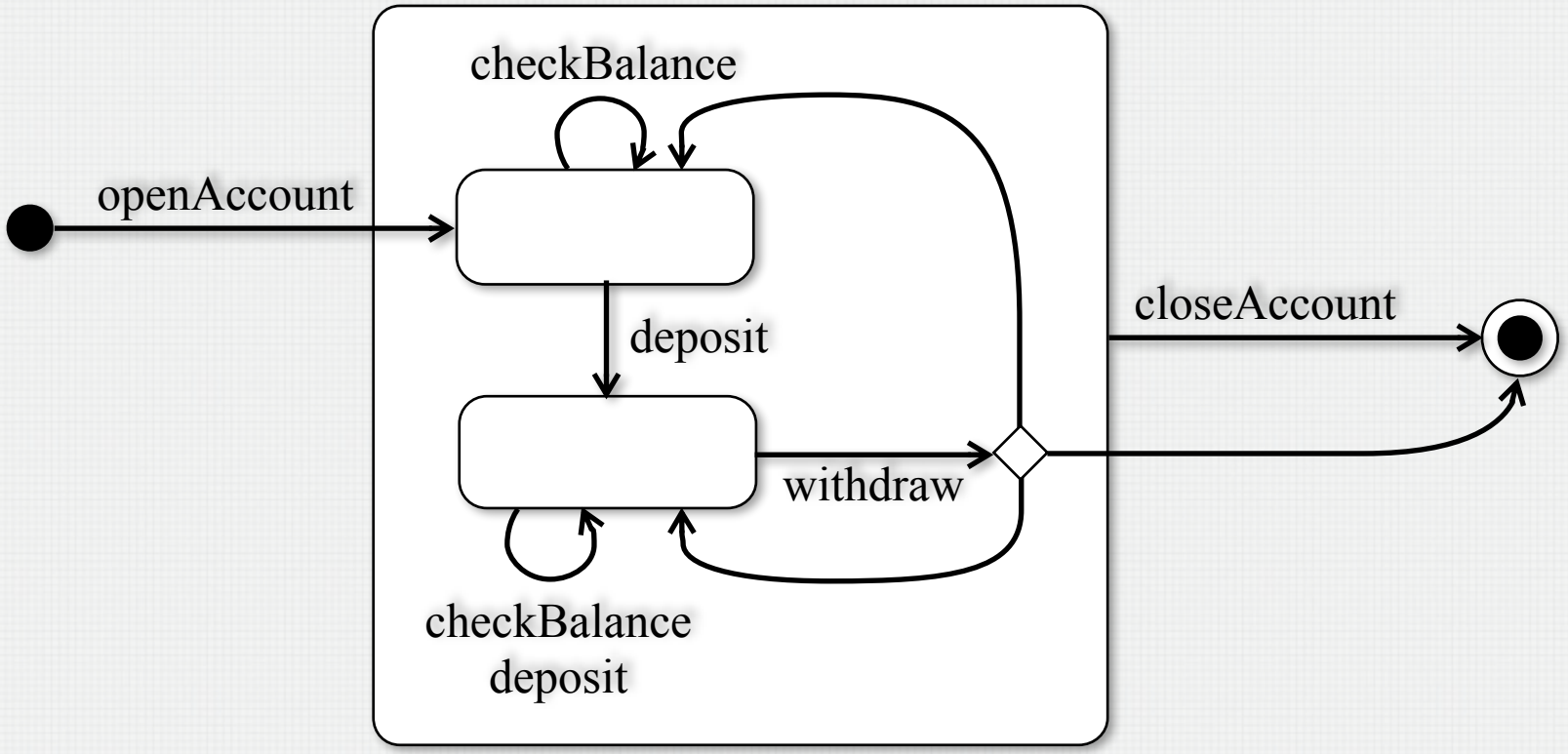
Hierarchy (1)

- Irrespective of what substate we are in, closeAccount moves the system to the final state



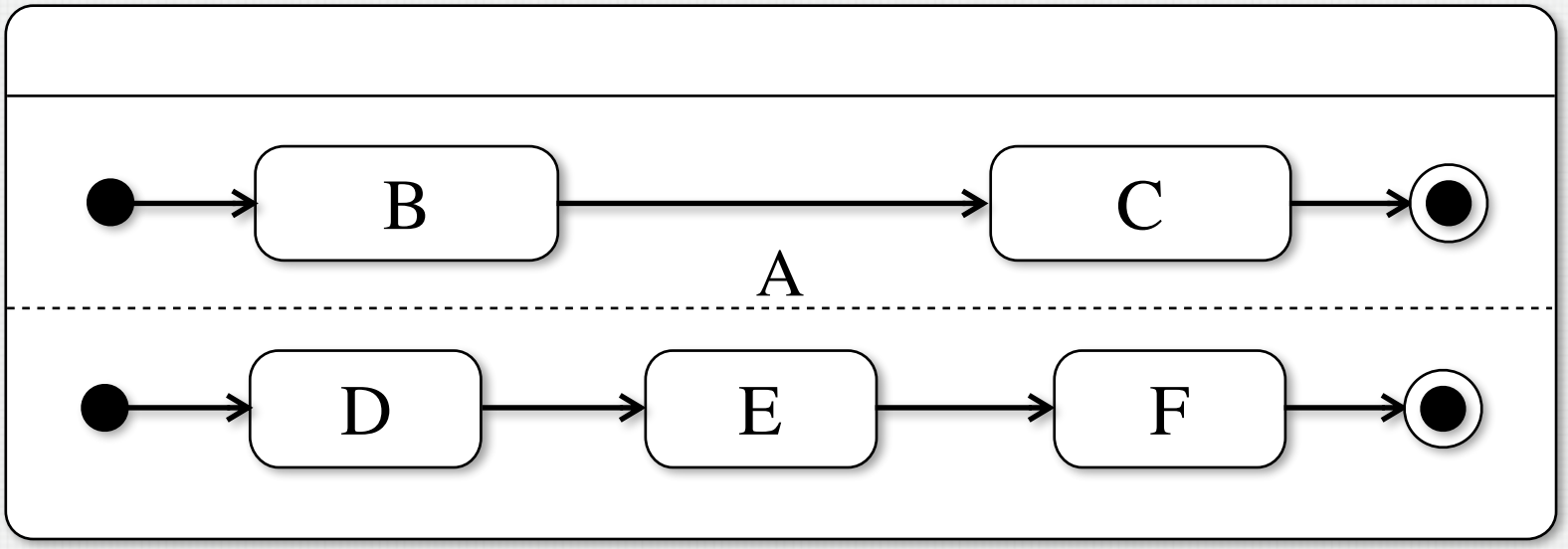
Hierarchy (2)

- Irrespective of what substate we are in, closeAccount moves the system to the final state

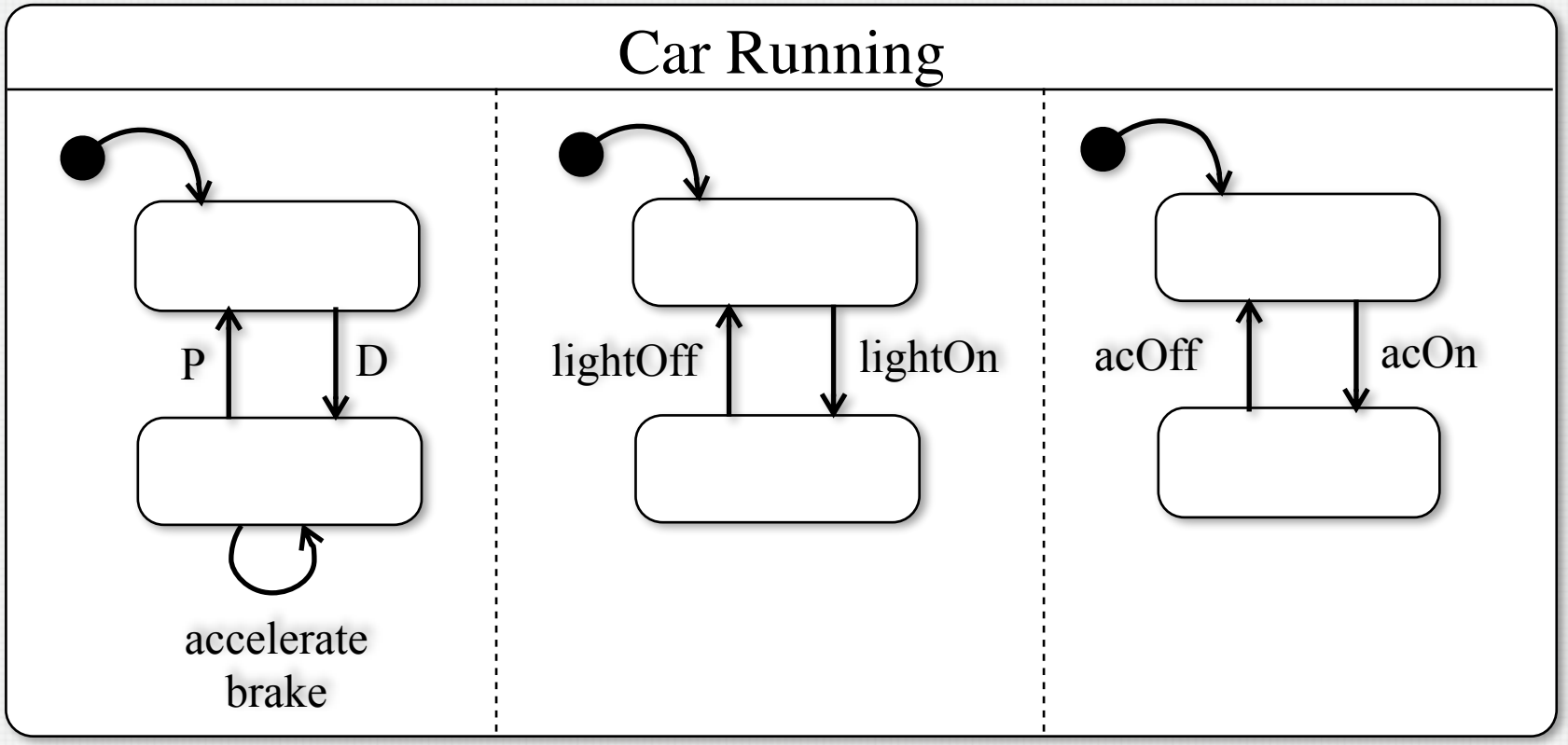


Concurrent / Orthogonal Substates

- Entering State A results in entering State B and State D
  - Shortcut for having States BD, CD, BE, CE, BF, CF
- Useful to model independent sequences

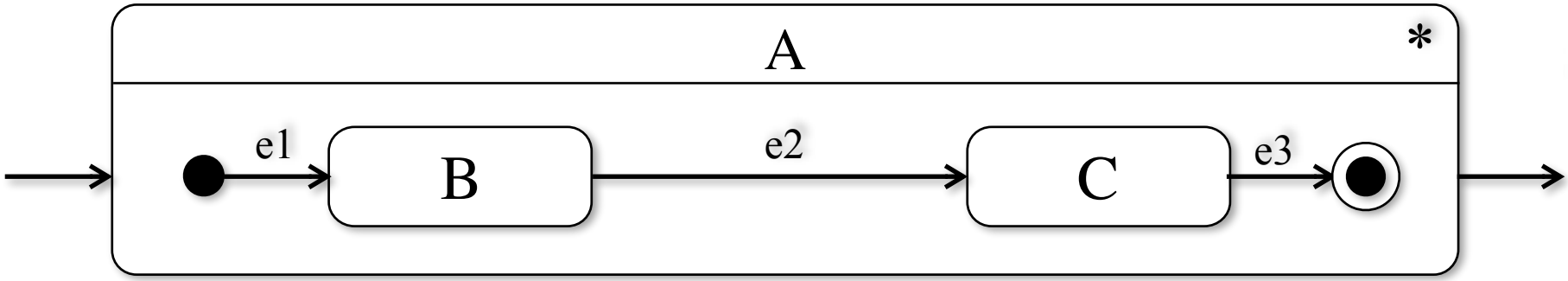


Concurrent Substates Example

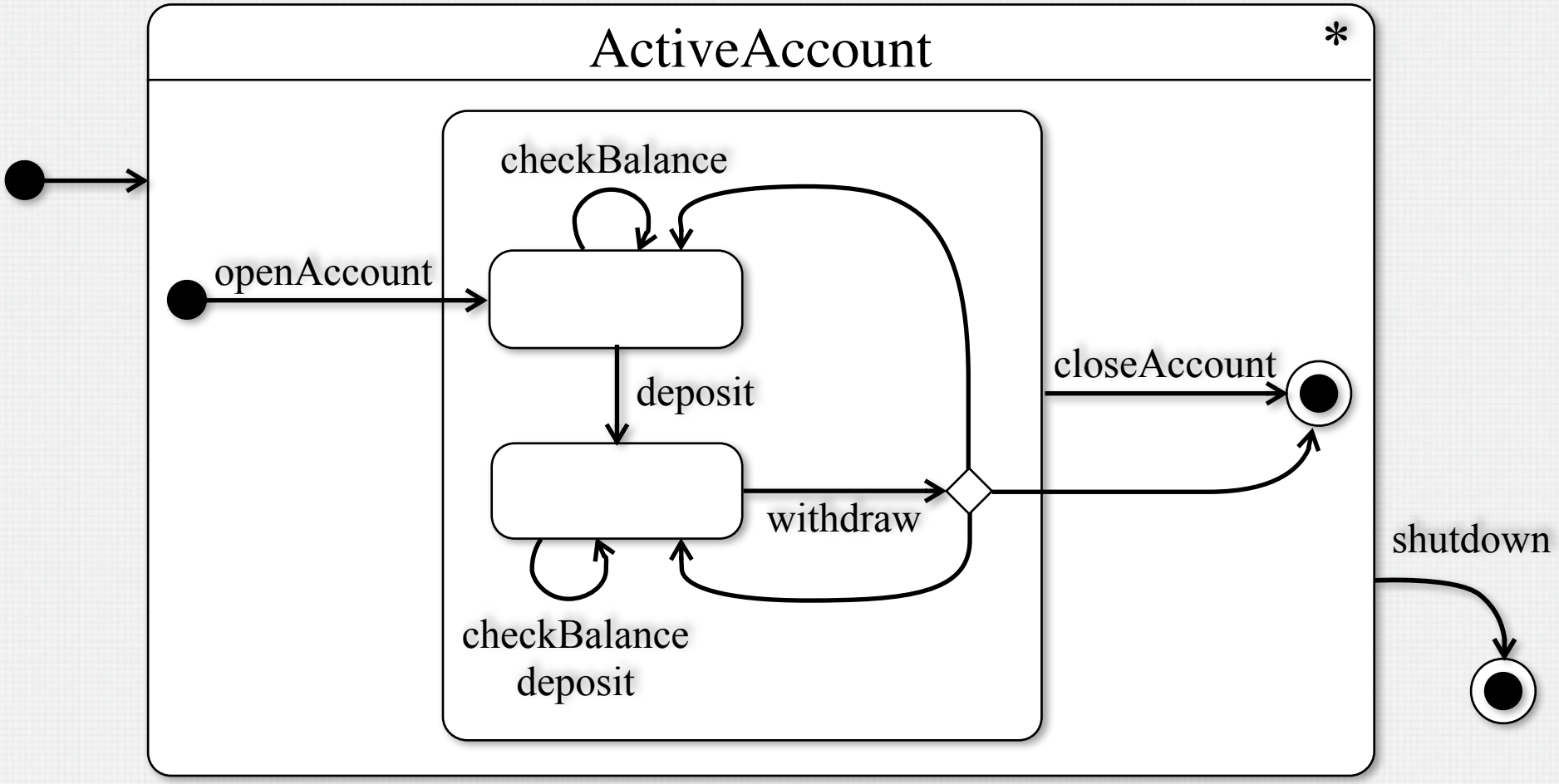


Autoconcurrent State

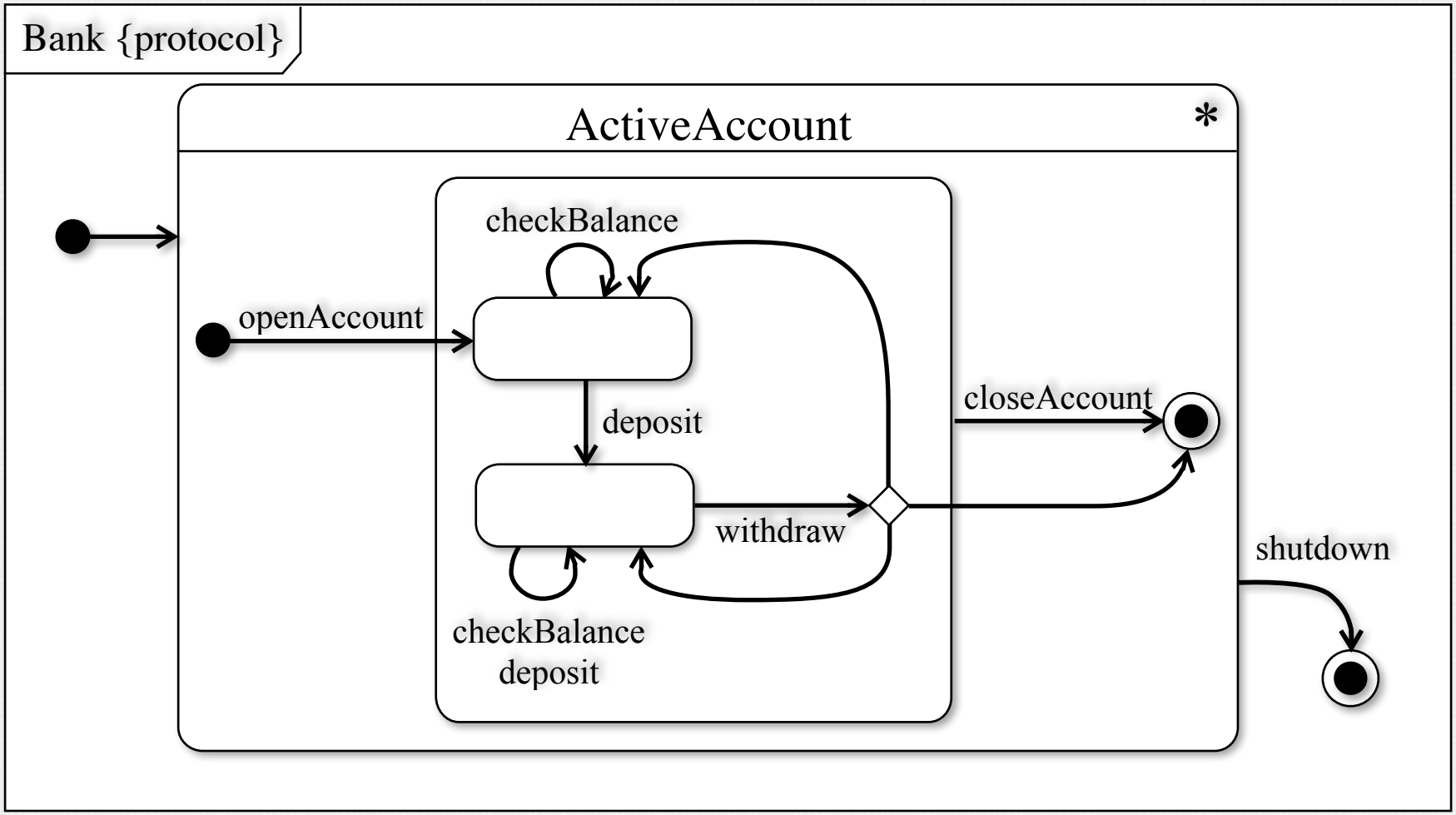
- As long as state A is active
  - Whenever e1 occurs, a new instance of state A is created, ready to process events
  - Typically, e1 will create a new instance of a class from the concept model, and all processing that follows (e2, e3) is related to that class instance



Autoconcurrent State Example



Example Protocol Model



Protocol Model and Operation Model (1)

- The behaviour of a system is defined by the Protocol Model and the Operation Model taken together.
- The Protocol Model determines the acceptability of an event and therefore of the corresponding triggering message.
- The precondition in the Operation Schema determines if the effect of an event is well behaved.
- The Protocol Model takes precedence over the precondition, as shown by the following table:

	Precondition true	Precondition false
Protocol accepts	Operation invoked and effect defined	Operation invoked but effect undefined
Protocol rejects	Event ignored	Event ignored

Protocol Model and Operation Model (2)

- Rejecting/ignoring an input event means that the state of the system is unaffected.
  - However, analysis uses an abstract notion of state, and the implementation is free to respond to the erroneous event and its triggering message, for example, with a helpful error message.
- A system need not have a Protocol Model. All input events are then acceptable at any time.

Checking for Model Consistency

- The analysis models should be complete and consistent
  - A model is complete when it captures all the meaningful abstractions in the domain.
  - Models are consistent when they do not contradict each other.
    - A model can also be checked for internal consistency.

Requirements Specification Process (1)

- 1. Determine the system interface
  - 1.1For establishing the system interface, analyze the scenarios in the Use Case Model. For each scenario:
    - Find the actors who are involved, and
    - The services they need.
  - 1.2Develop the Environment Model: identify actors, output messages, and input messages (system operations).
  - 1.3 Produce the Concept Model by adding the boundary and actors to the Domain Model. Only actors having direct interaction with the system should be shown, and nothing else should appear outside of the boundary. Add roles to all association ends.

Requirements Specification Process (2)

- 2. Develop the Behavior Model
  - 2.1 Develop the Protocol Model
    - Generalize the scenarios of the Use Case Model and define system states.
    - Combine system states to form the Protocol Model.
  - 2.2 Develop the Operation Model
    - 2.2.1 For each system operation, develop the pre- and postconditions:
      - Describe each aspect of the result as a separate subclause of Post.
      - Use the Environment Model to find the messages that have to be output as a result.
      - Check that results do not allow unwanted values.
      - Add relevant Concept Model invariants to the pre- and postconditions.
      - Ensure that the pre- and postconditions are satisfiable.
    - 2.2.2 Derive Scope, Messages and New clauses from the postconditions (incrementally with 2.2.1)
    - 2.2.3 Complete the message (type) declarations in the Environment Model

Requirements Specification Process (3)

- 3. Check the Analysis Models
  - 3.1 Check for completeness against the requirements:
    - All possible scenarios stated in the use cases are covered by the Protocol Model.
    - All required system services can be mapped onto system operations.
    - All static information is captured by the Concept Model.
    - Any other information, e.g. technical definitions and invariant constraints, are documented.
    - To check for completeness, compare the Use Case Model and the Operation Model
      - Inspect the use cases and define the state change that each should cause. Then "execute" the use cases, using the operation schemas. Check that resulting state conforms to what was expected.

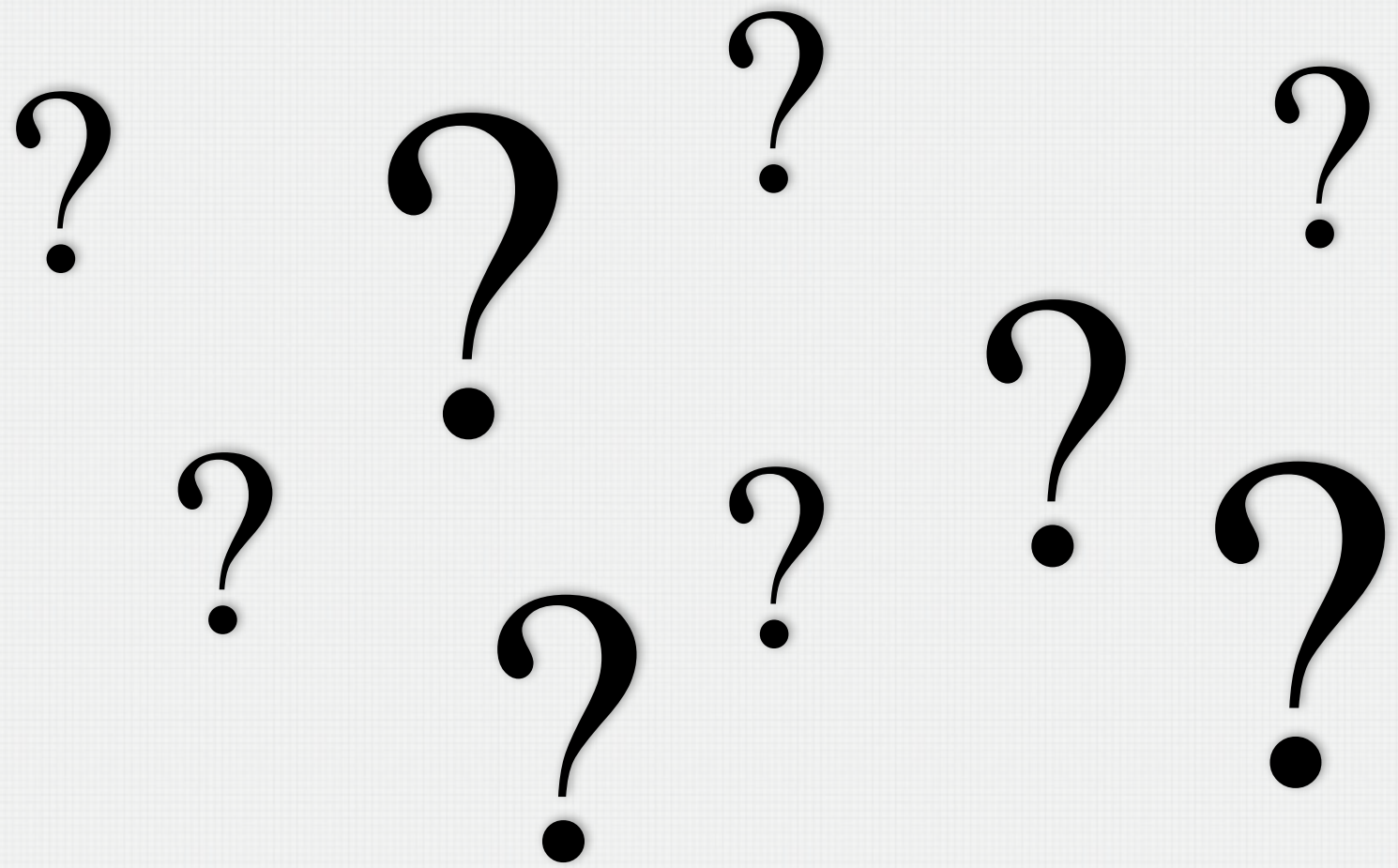
Requirements Specification Process (4)

- 3.2 Consistency between models:
  - Domain Model versus Concept Model:
    - All classes, relationships and attributes mentioned in the Domain Model appear in the Concept Model, or their absence can be justified and is documented.
  - Environment Model versus Concept Model:
    - The boundary of the Concept Model is consistent with the Environment Model.
  - Environment Model versus Protocol Model:
    - Every input message in the Environment Model appears in the Protocol Model as an event, and vice versa.
  - Concept Model versus Operation Model:
    - All classes, attributes and associations accessed in the Operation Model are part of the Concept Model, and there are no “useless” classes in the Concept Model.
    - The Operation Model must preserve Concept Model invariants.
  - Environment Model versus Operation Model:
    - An actor that appears in the Operation Model is part of the Environment Model.
    - All input messages in the Environment Model must trigger an operation modeled by an operation schema in the Operation Model, and all output messages in the Environment Model must be generated by a system operation.

Iterations

- It is usually necessary to go back and forth between the Environment Model, the Concept Model and the Operation Model to make them complete, consistent, but also as simple as possible, by eliminating the unused elements.

Questions?



Buy Drink Use Case (1)

Use Case: Buy Drink

Scope: Vending Machine

Level: User Goal

Intention in Context: The intention of the *Customer* is to buy a drink in exchange of money.

Multiplicity: There can always be only one *Customer* interacting with the system at a given time.

Primary Actor: Customer

Secondary Actors: Selector Button, Coin Slot, Shelf, Sensor, Money Box, Drink Light, Cancel Button, Display, Terminal

Precondition: The system is in service, filled with drinks and change, and the Money Box is not full.

Buy Drink Use Case (2)

Main Success Scenario:

Customer selects drink by pushing appropriate drink selector button.

1. *Button* notifies *System* of selected drink.
2. *System* displays the price of the selected drink on *Display*.

Customer inserts a coin into Coin Slot.

3. *Coin Slot* notifies *System*.
4. *System* recognizes the coin, and updates the remaining price on *Display*.

Steps 3 and 4 are repeated until the amount of inserted money reaches or exceeds the price of the drink.

5. *System* validates that there are sufficient funds for the selection and notifies *Shelf* to start dispensing the drink.
6. *Sensor* informs *System* that the drink has been dispensed.
7. *System* asks *Money Box* to collect the specified amount of money and, if necessary, provide the change.

Customer collects the drink and optionally the change.

Buy Drink Use Case (3)

Extensions:

2a. *System* ascertains that the selected drink is not available and flashes *Drink Lights*; use case ends in failure.

4a. *System* fails to identify the coin; *System* asks *Money Box* to eject coin; use case continues at step 3.

(3-4)a. *Customer* informs *System* to abort the sale by hitting the *Cancel* button;

(3-4)a.1 *System* asks *Money Box* to eject coins; use case ends in success.

(3-4)b. *System* times out.

(3-4)b.1 *System* asks *Money Box* to eject the inserted coins; use case ends in failure.

5a. *System* ascertains that the inserted money exceeds the price for the drink and that there is not enough change;

5a.1 *System* asks *Money Box* to eject inserted coins.

5a.2 *System* displays “no change” on *Display*; use case ends in failure.

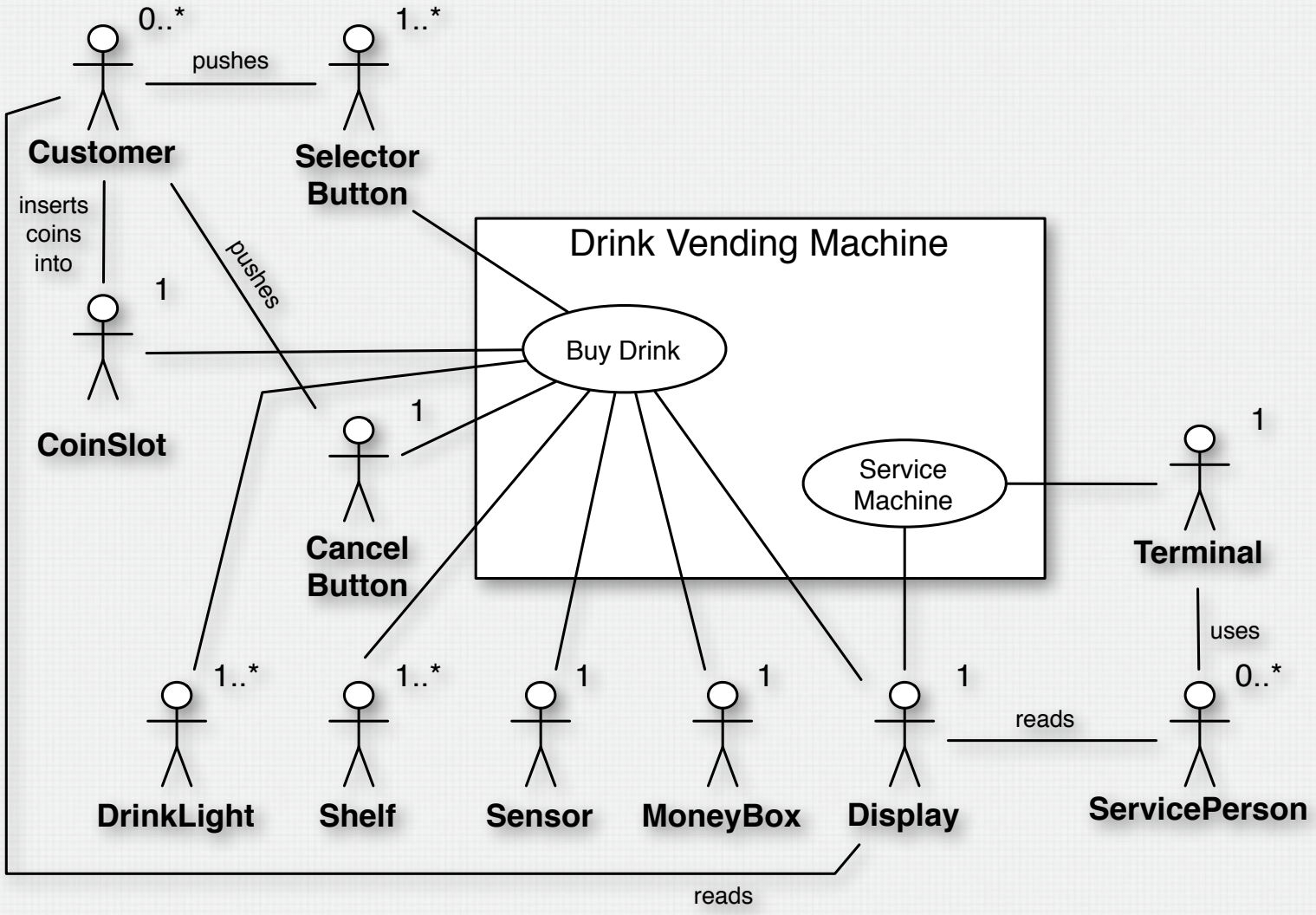
7a ll. The *Money Box* is full.

7a ll.1 *System* displays “no service” on *Display* and goes out of service; use case ends successfully.

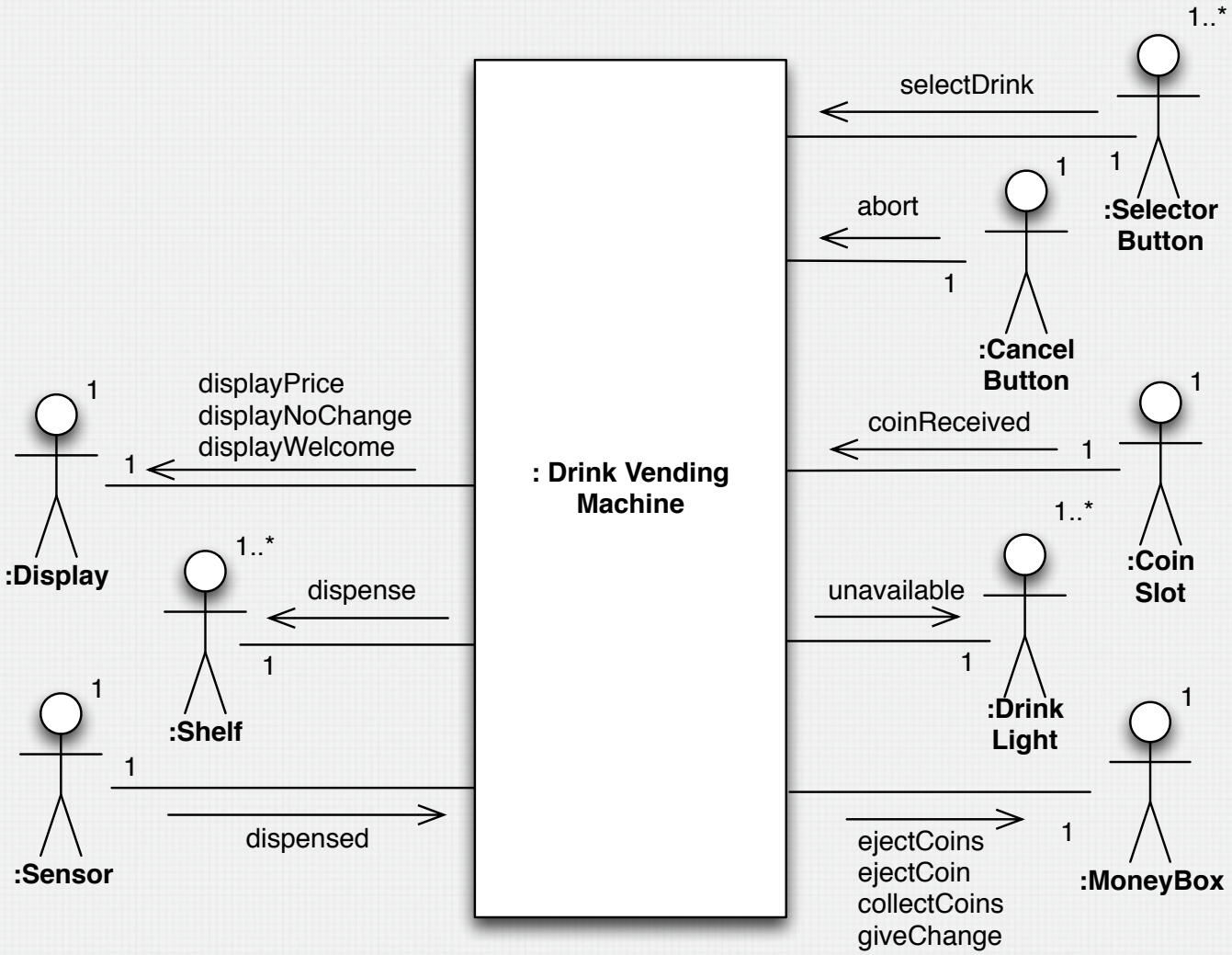
7b ll. The delivered drink was the last one of that kind.

7b ll.1 *System* turns on the appropriate *Drink Light*; use case ends successfully.

Vending Machine Use Case Diagram



Drink Vending Machine Environment Model



Drink Vending Machine Questions

1. Create a URN model for the *BuyDrink* use case.
   - If you decide to group several basic interaction steps into one URN responsibility, please use the description field of the responsibility to document which use case steps it represents

Clinical Lab System Question

- The task is to develop a computerized data management system for a clinical test analyzer. An analyzer can carry out tests on body fluids such as blood, urine, and swab specimens. An analyzer is capable of carrying out tests on several samples simultaneously.
- The technician enters a batch of samples from a single patient by first entering the patient's identification and then indicating, one at a time, the tests that need to be performed on the samples. By a "batch end" message, s/he informs the system that there are no more samples for the current patient. When all the tests for a patient have been performed by the analyzer, they are collected together into a patient report, which is sent to the technician.

Clinical Lab System Question (2)

- The system can perform test requests for more than one patient at a time. The technician may ask for a report reflecting the current status of a patient's tests before they are all completed. The tests for a patient may also be aborted, in which case a patient report containing just the test results collected so far is generated and all further tests on samples from the same patient are ignored.
- Environment Model
  - Show by a Environment Model the interaction between the technician, the system and the analyzer.
  - Provide message declarations.
  - Write down some possible/forbidden message sequences; show both input and output messages. (Can also be answered based on the Protocol Model.)
- Protocol Model
  - Devise a Protocol Model for the clinical lab system