Chap. 11
Specifying Control

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill

Topics Covered

- States and Events
- Notations
- Preparing a State Machine
- Protocol and Behavioural State Machines
- Consistency Checking

Interaction diagrams vs. State machines

- Interaction (communication or sequence) diagrams
  - Capture responses of all objects that are involved in a single use case or other interaction
- State machine
  - Capture all possible responses of a single object to all use cases in which it is involved

State

- The current state of an object is determined by the current value of the object's attributes and the links that it has with other objects.

- Some attributes have no impact on its state
  - `StaffName` or `StaffNo` in `StaffMember` class
- Others have some impact on its state
  - `StartDate` in `StaffMember` class

State

- A state describes a particular condition that a modelled element (e.g. object) may occupy for a period of time while it awaits some event (trigger).
- Trigger
  - An event that can cause a state change of the object
- Transition
  - Movement from one state to another
- Conceptually, an object remains in a state for an interval of time.

Types of Trigger

- A *change trigger*
  - occurs when a condition becomes true.
- A *call trigger*
  - occurs when an object receives a call for one of its operations either from another object or from itself.
- A *signal trigger*
  - occurs when an object receives a signal (an asynchronous communication).
- An *relative-time trigger*
  - is caused by the passage of a designated period of time after a specified event.

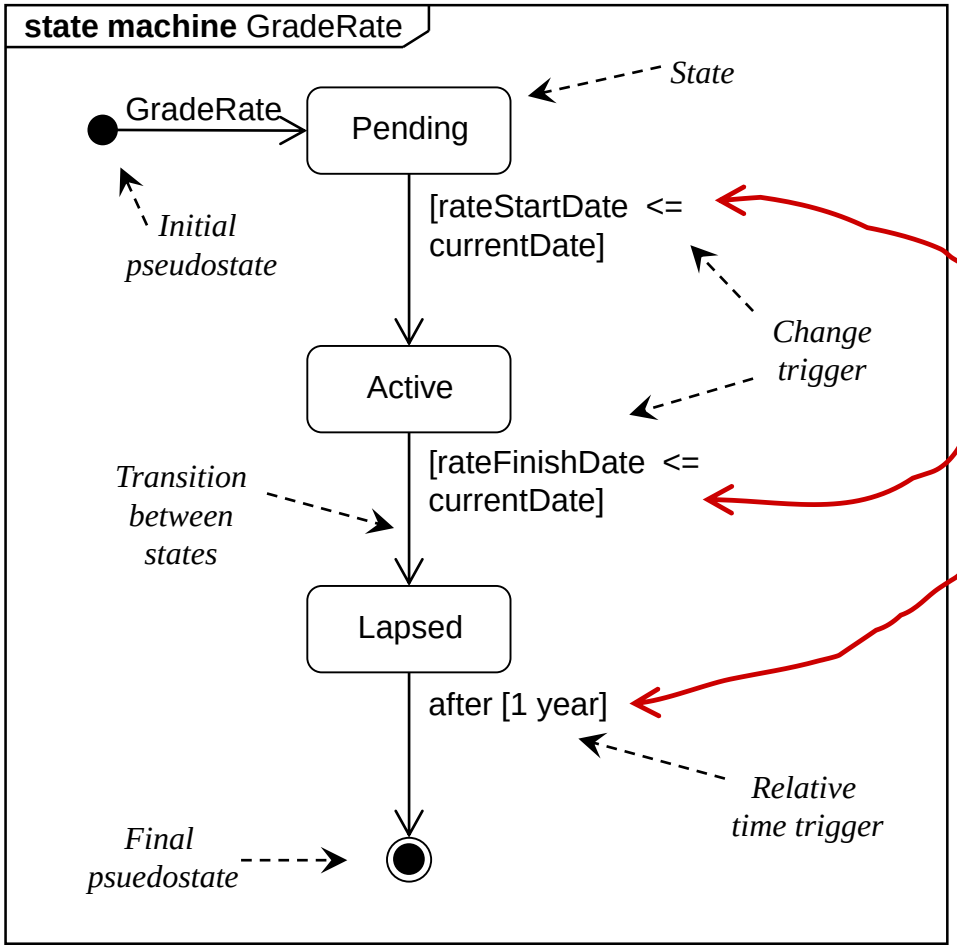
State machine

- The current state of a `GradeRate` object can be determined by the two attributes `rateStartDate` and `rateFinishDate`.

- An enumerated state variable may be used to hold the object state, possible values would be Pending, Active or Lapsed.

State machine

state machine for the class `GradeRate`.



Movement from one state to another is dependent upon events that occur with the passage of time.

State machine

- Transition string

  `trigger-signature '[' constraint ']' '/' activity-expression`

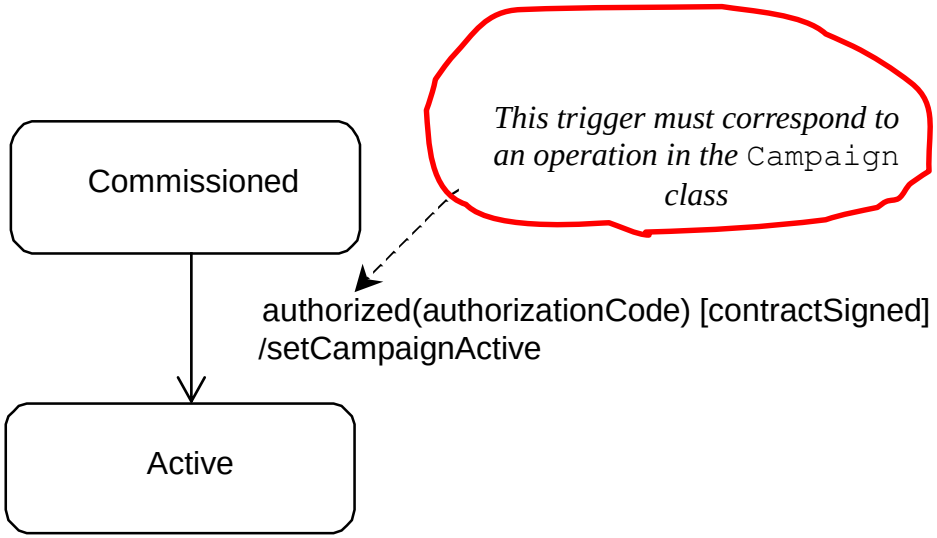
- `trigger-signature` =

  `event-name '(' parameter-list ')'`

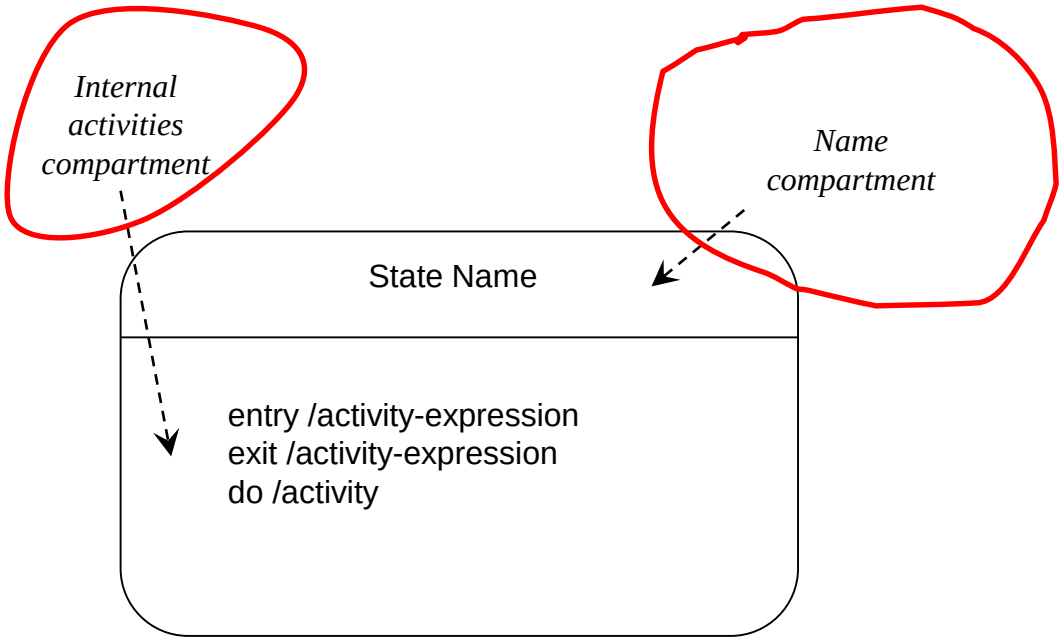
- `constraint` (guard condition) =
  - boolean expression that is evaluated at the time the trigger fires.
- `activity-expression`
  - executed when a trigger causes the transition to fire
- `parameter-list`

  `parameter-name ':' type-expression`

Transition string

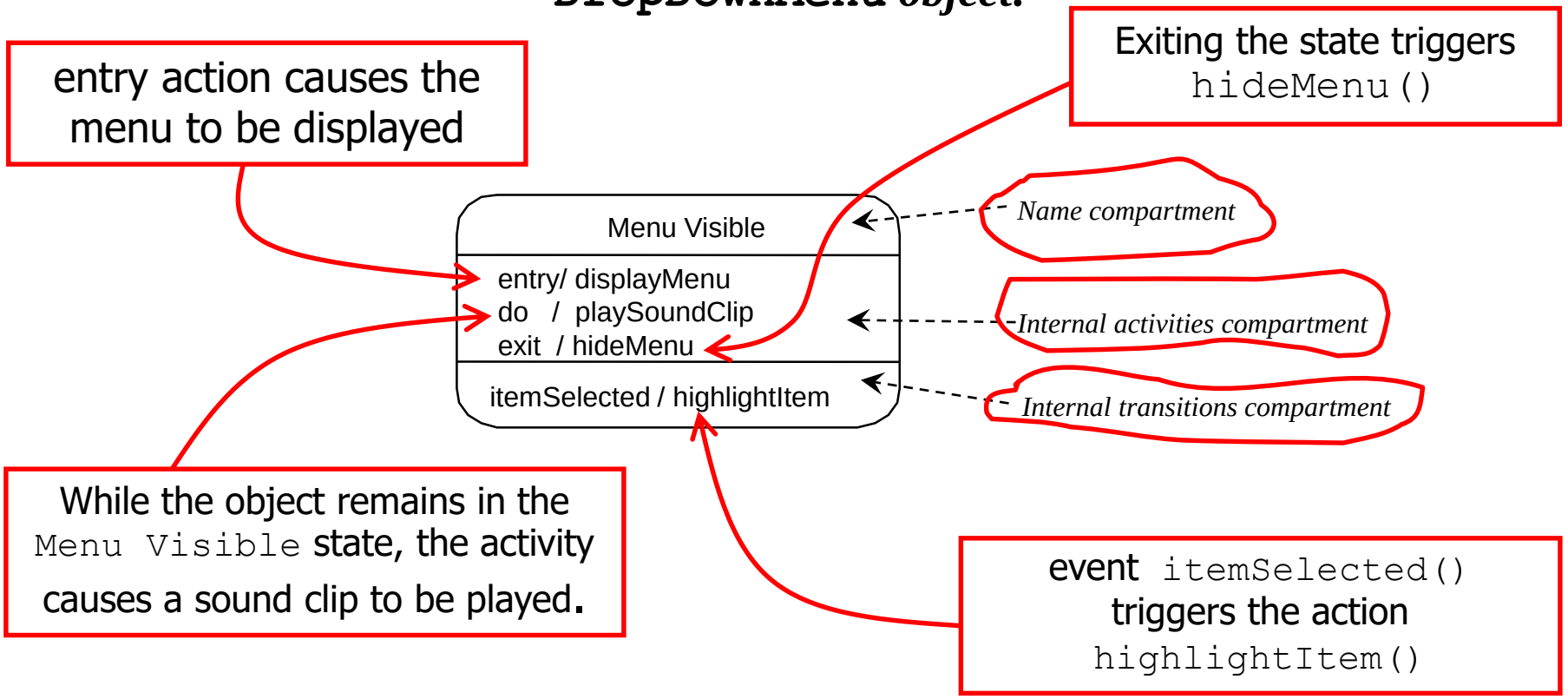


Internal Activities



'Menu Visible' State

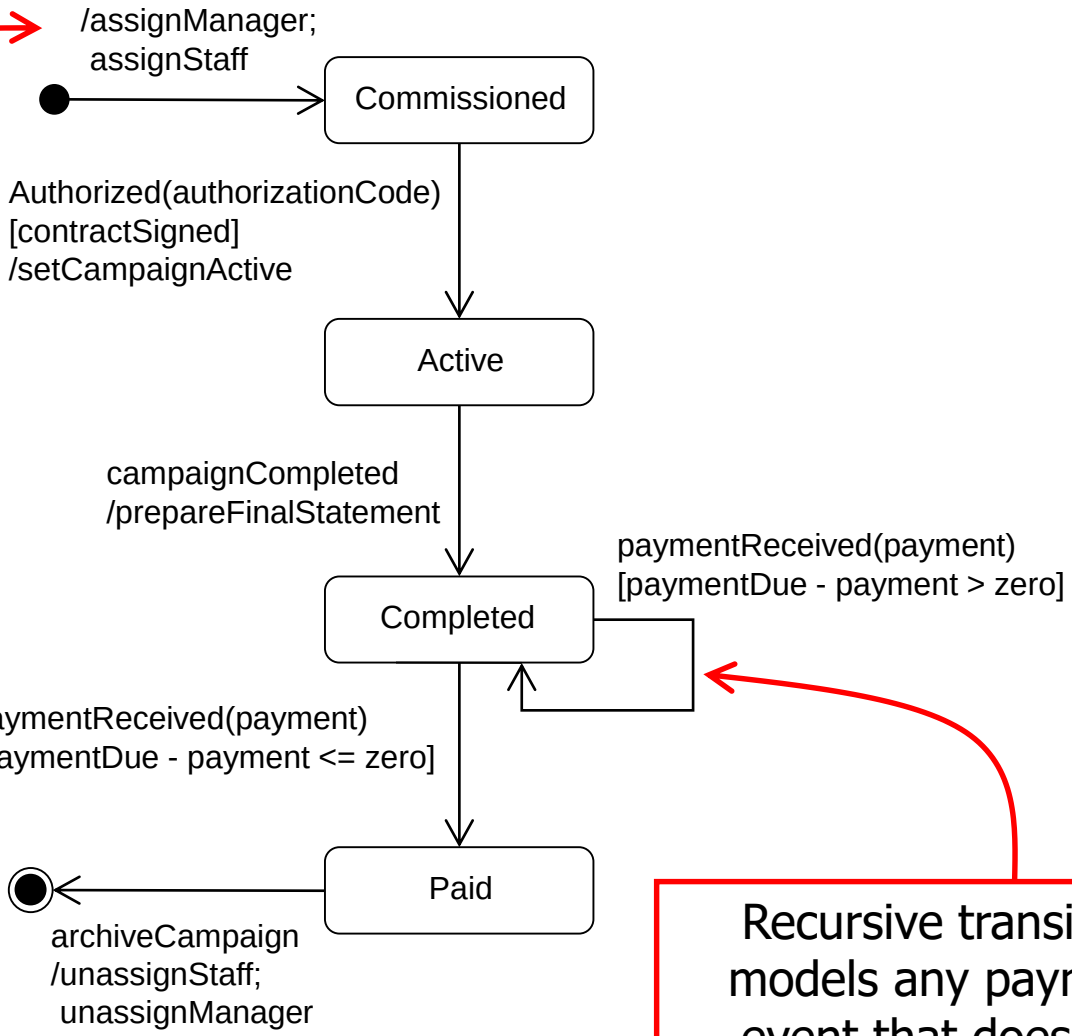
`Menu Visible` *state for a* `DropDownMenu` *object.*



Action-expression assigning manager and staff on object creation

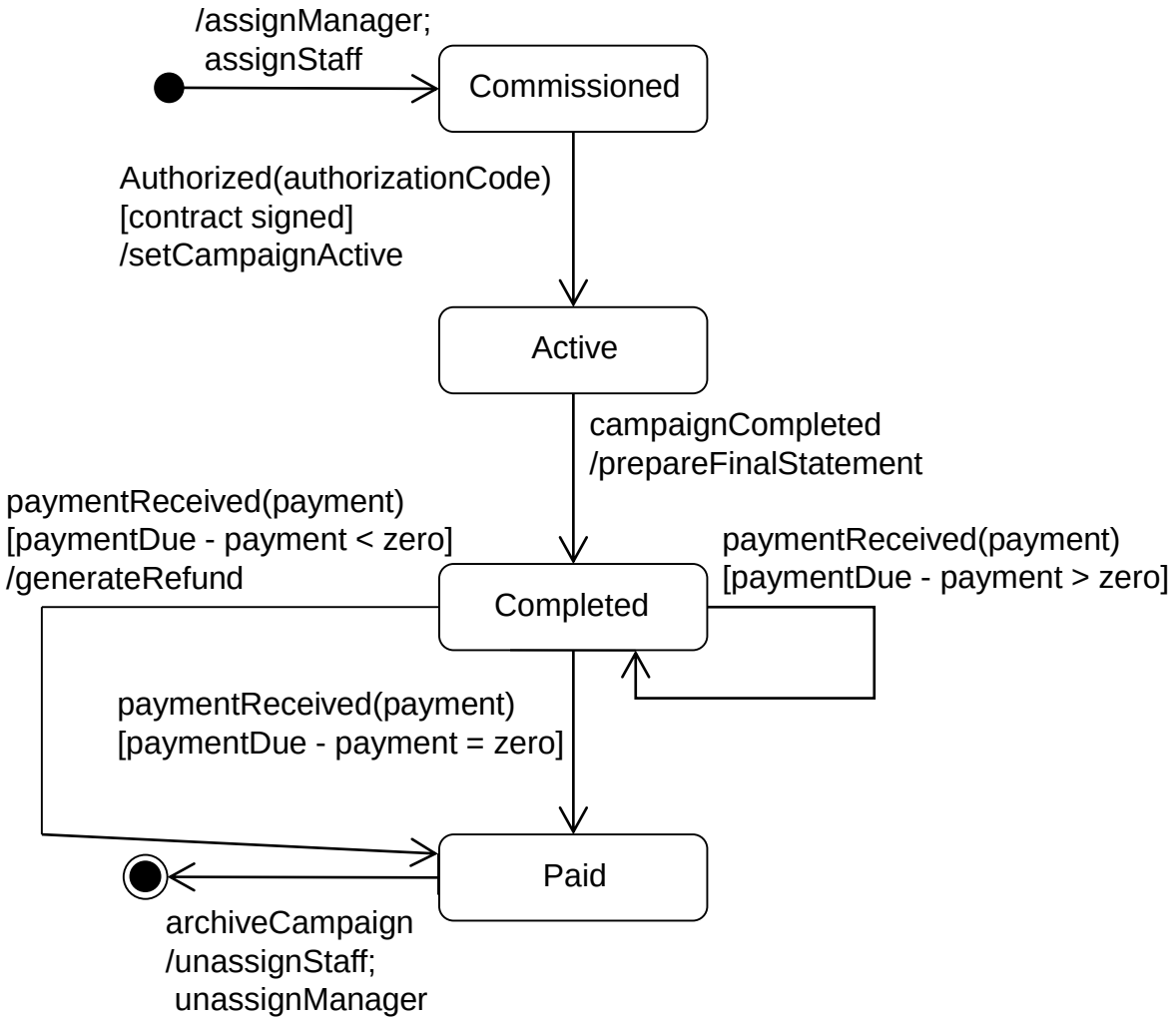
state machine f or the class `Cam paign`.

Guard condition ensuring complete payment before entering `Paid`



Recursive transition models any payment event that does not reduce the amount due to zero or beyond.

A revised state machine for the class `Campaign`



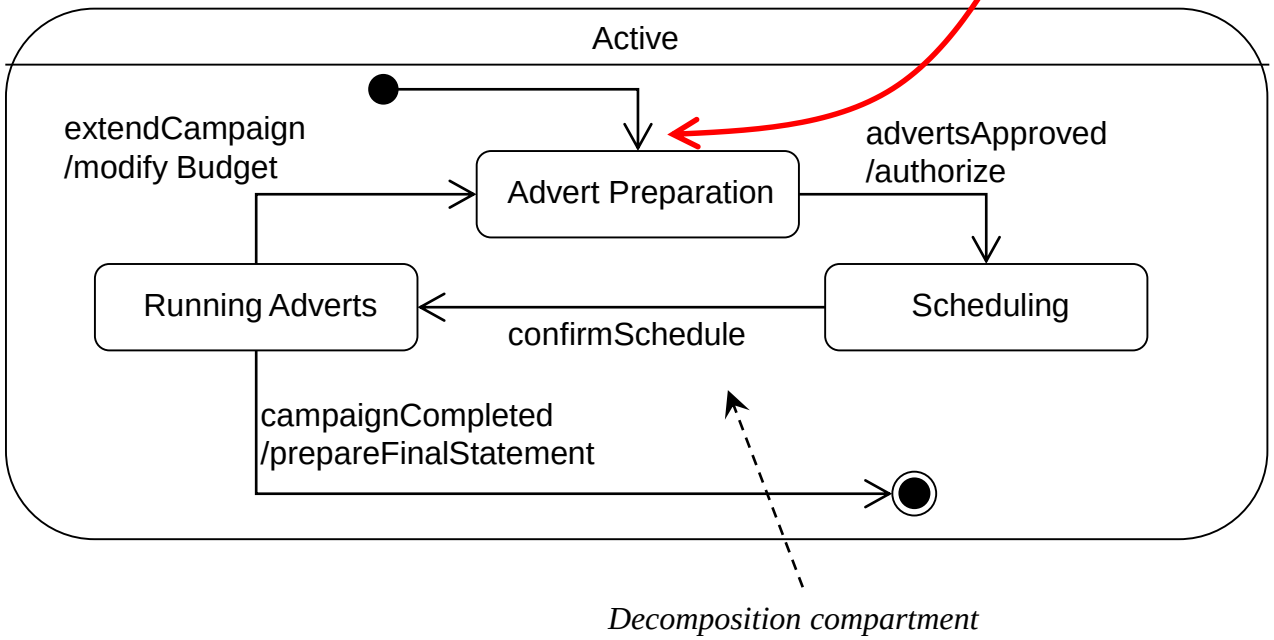
If the user requirements were to change, so that an overpayment is now to result in the automatic generation of a refund, a new transition is added.

Composite states

When state behavior is complex, objects can be represented at different levels to reflect any hierarchy of states

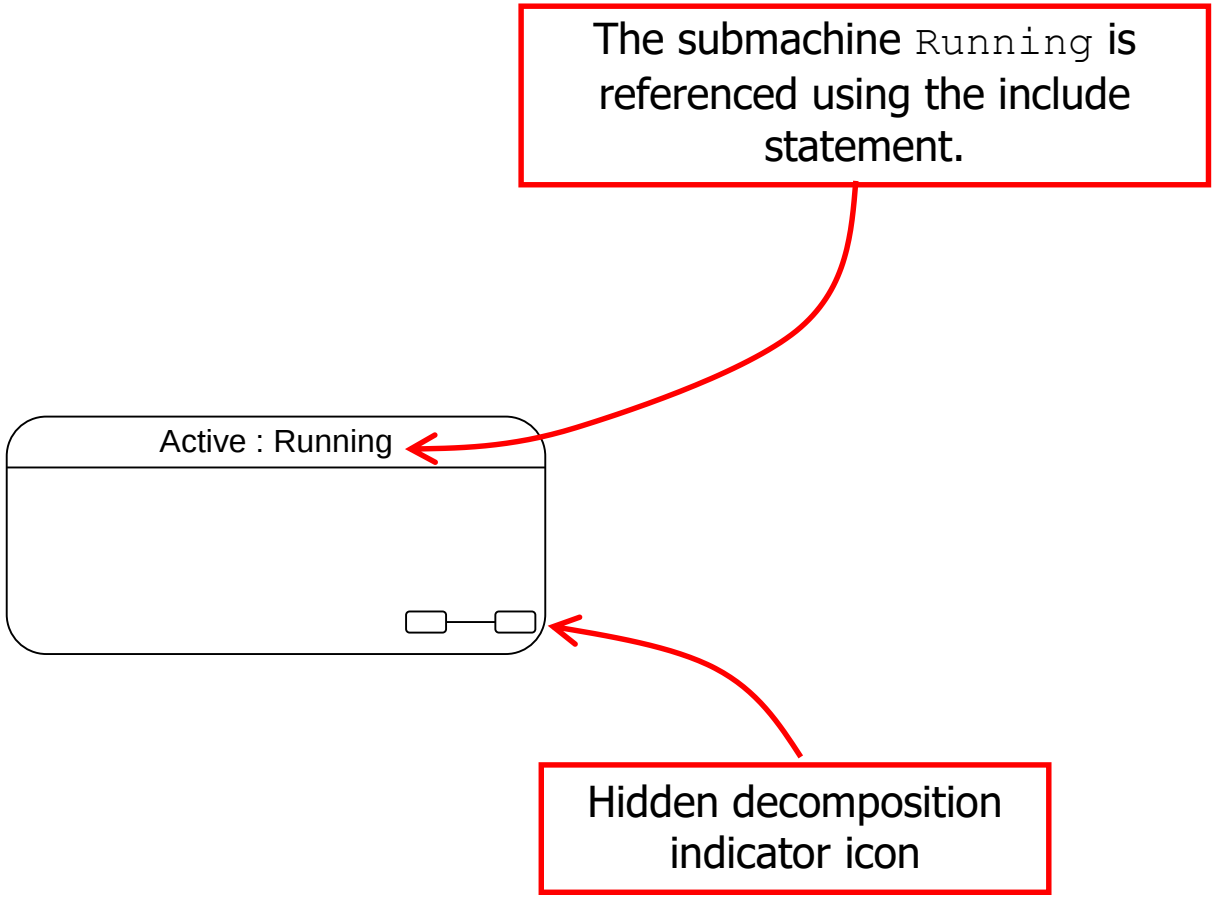
The transition from the initial pseudostate symbol should not be labelled with an event but may be labelled with an action, though it is not required in this example

The `Active` state of `Campaign` showing nested substates.



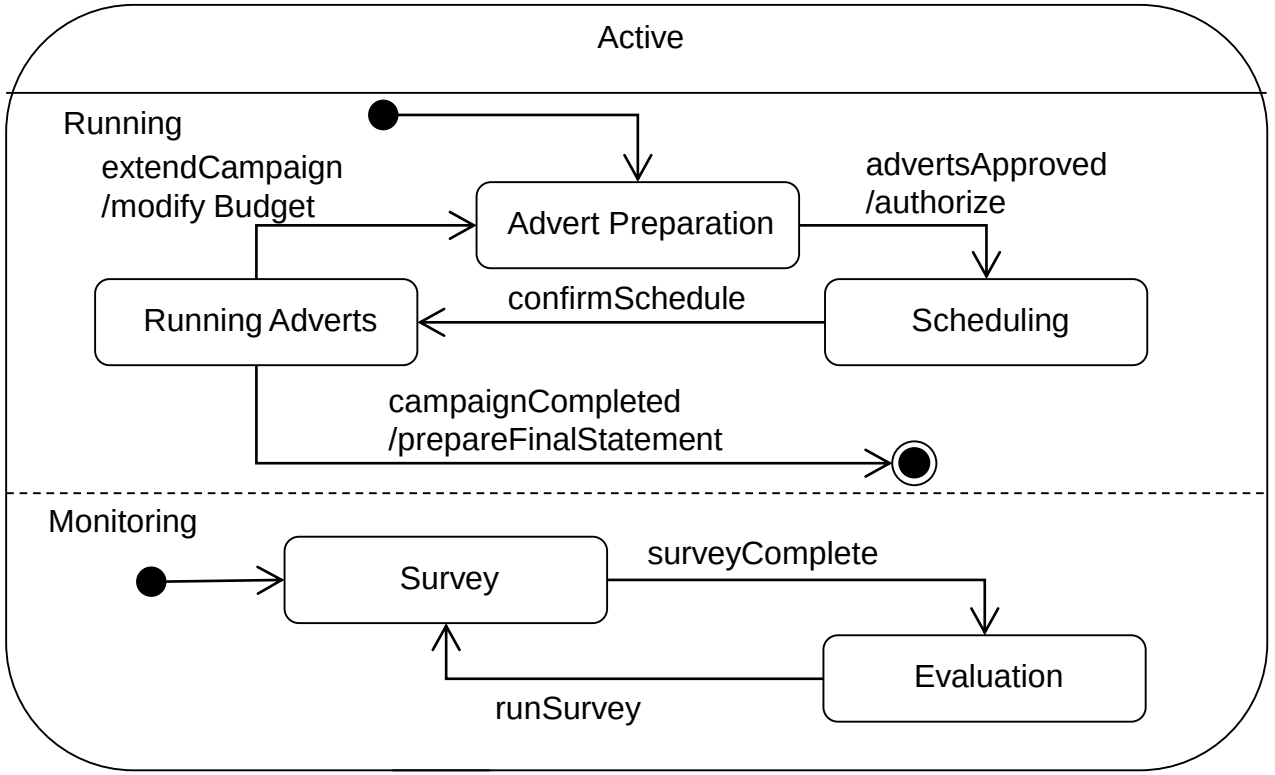
Composite states

The `Active` state of `Campaign` with the detail hidden.



Concurrent states

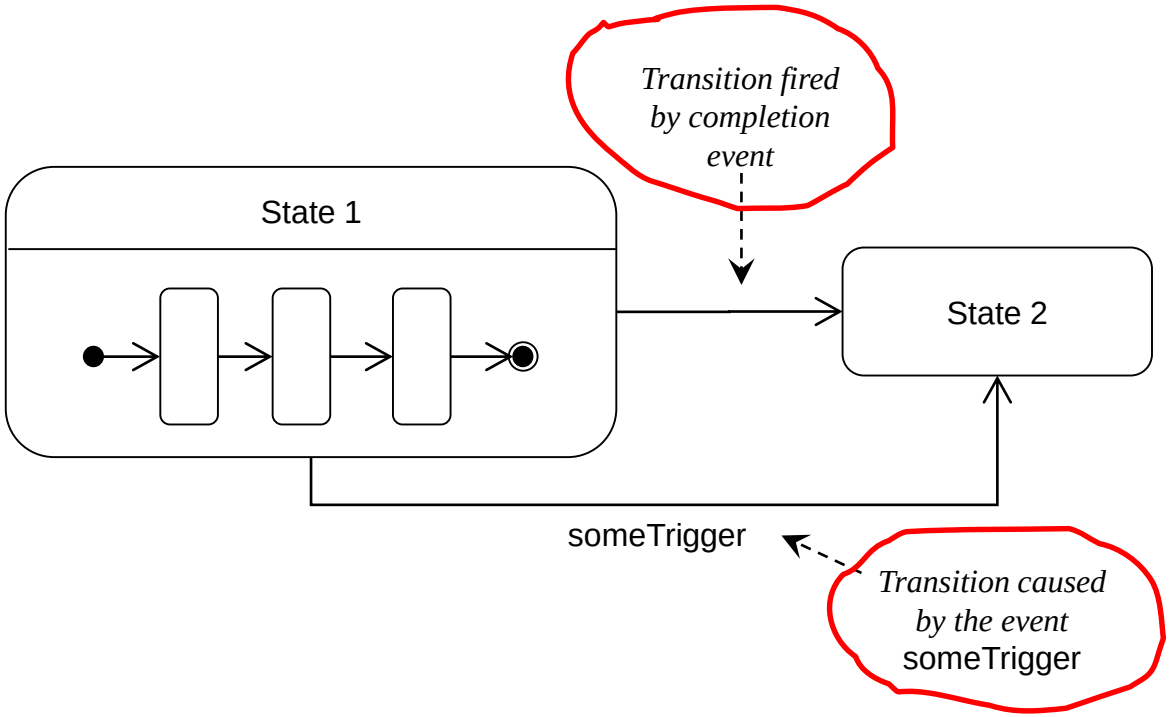
Distinct substates can be entered and exited independently of each other



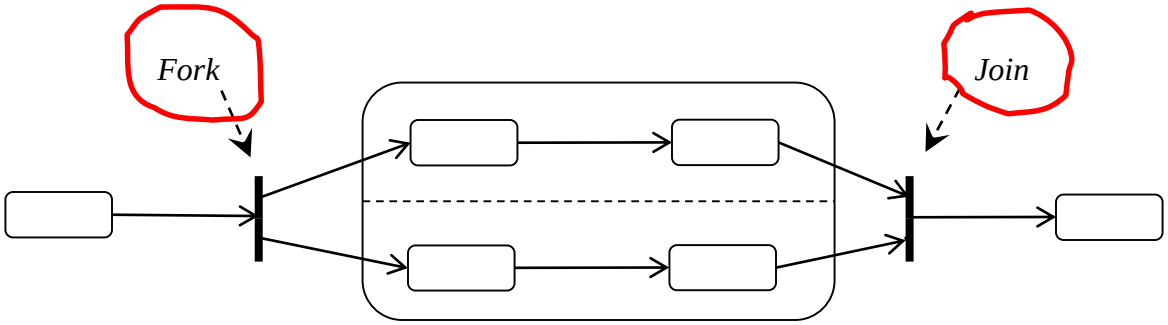
Concurrent States

- A transition to the `Active` state means that the `Campaign` object simultaneously enters the `Advert Preparation` and `Survey` states.
- Once the composite state is entered a transition may occur within either concurrent region without having any effect on the state in the other concurrent region.

Completion Event



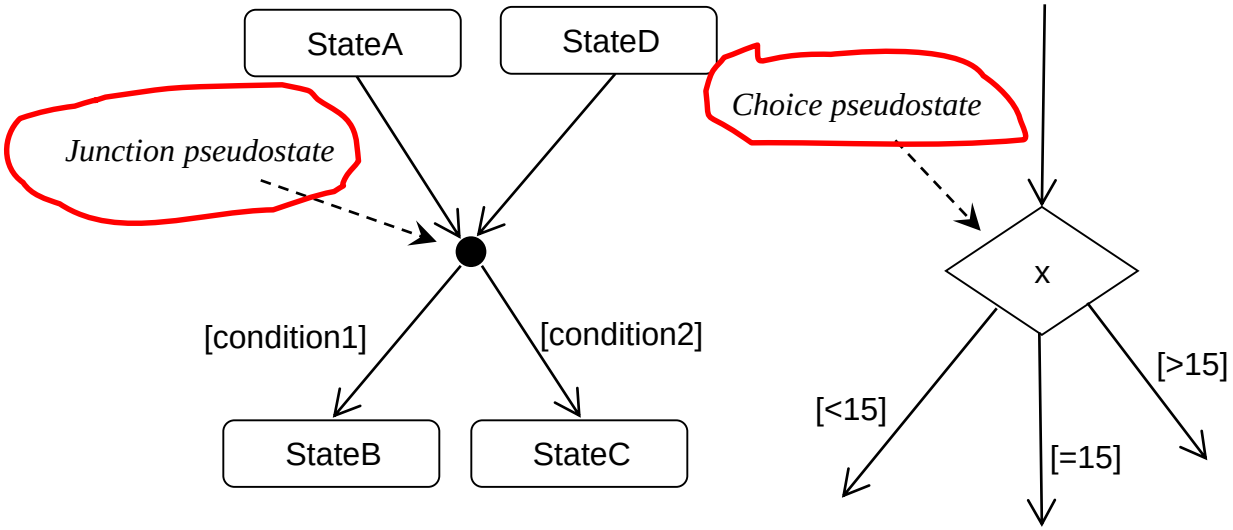
Synchronized Concurrent Threads.



- **Explicitly showing how an event causes specific concurrent substates to be entered.**

- **Shows that the composite state is not exited until both concurrent nested state machines are exited.**

Junction & Choice Pseudostates



Preparing state machines

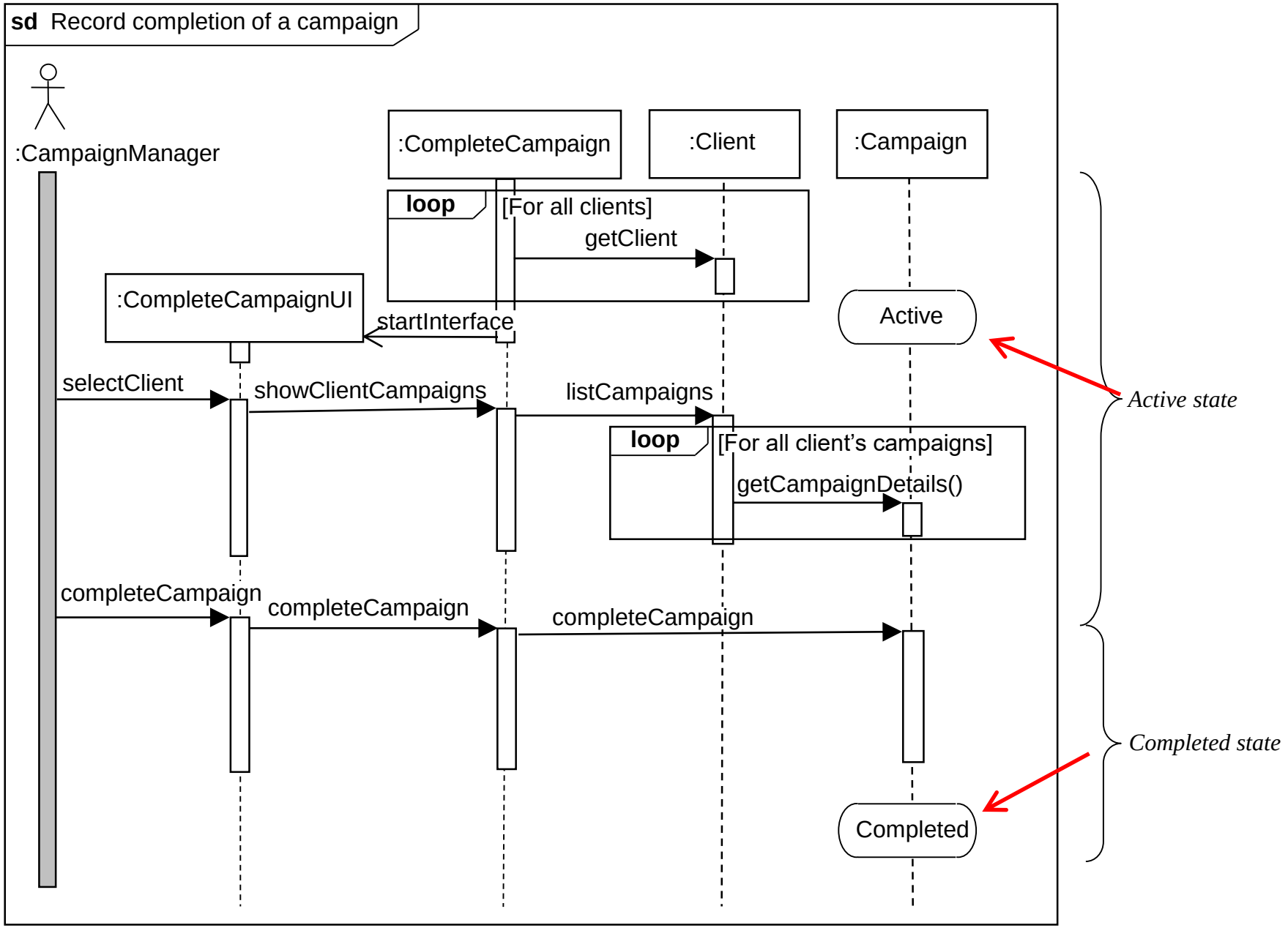
1. Examine all interaction diagrams that involve each class that has heavy messaging.
2. Identify the incoming messages on each interaction diagram that may correspond to events. Also identify the possible resulting states.
3. Document these events and states on a state machine.
4. Elaborate the state machine as necessary to cater for additional interactions as these become evident, and add any exceptions.

Preparing state machines

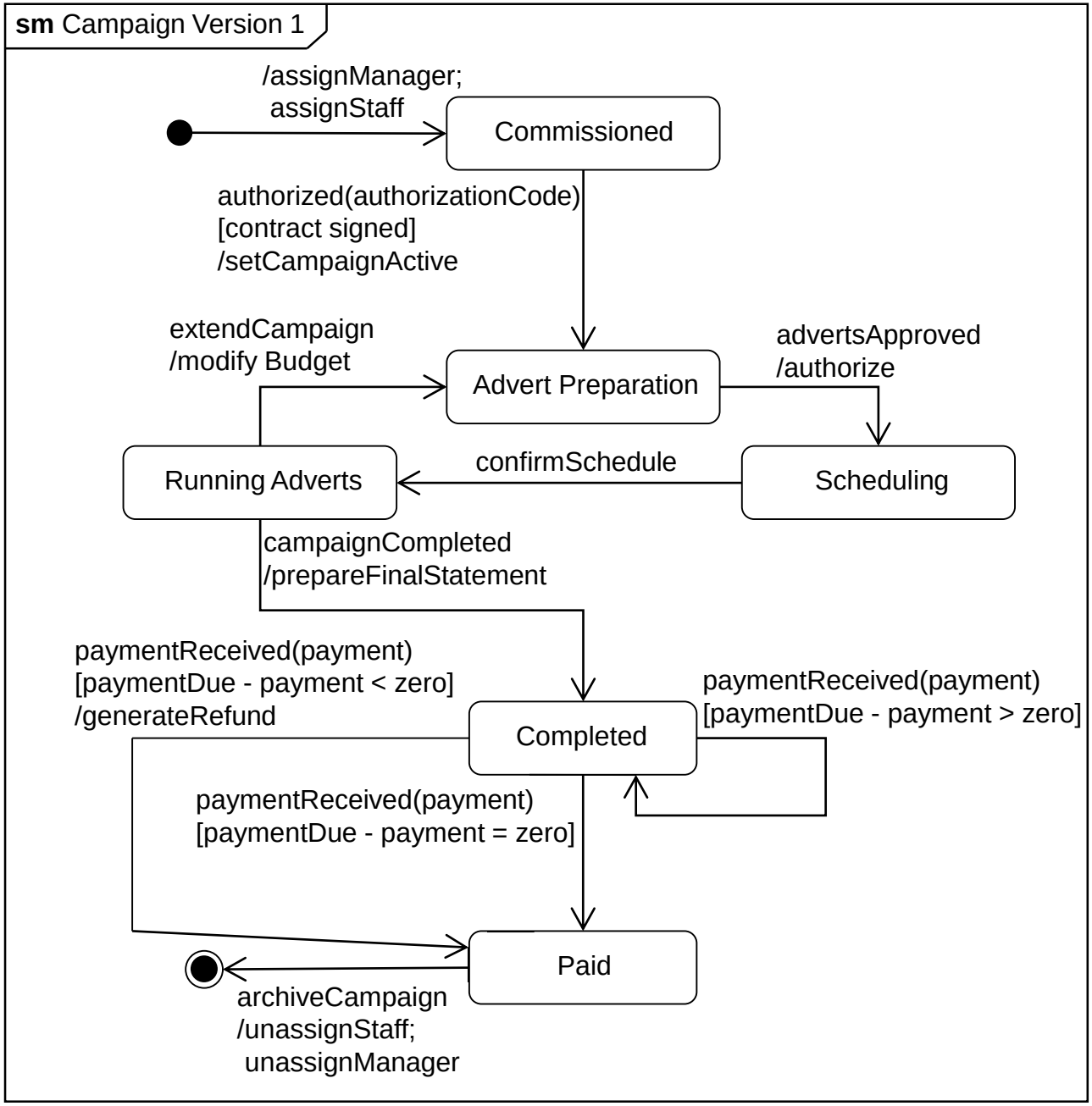
5. Develop any nested state machines (unless this has already been done in an earlier step).
6. Review the state machine to ensure consistency with use cases. In particular, check that any constraints that are implied by the state machine are appropriate.

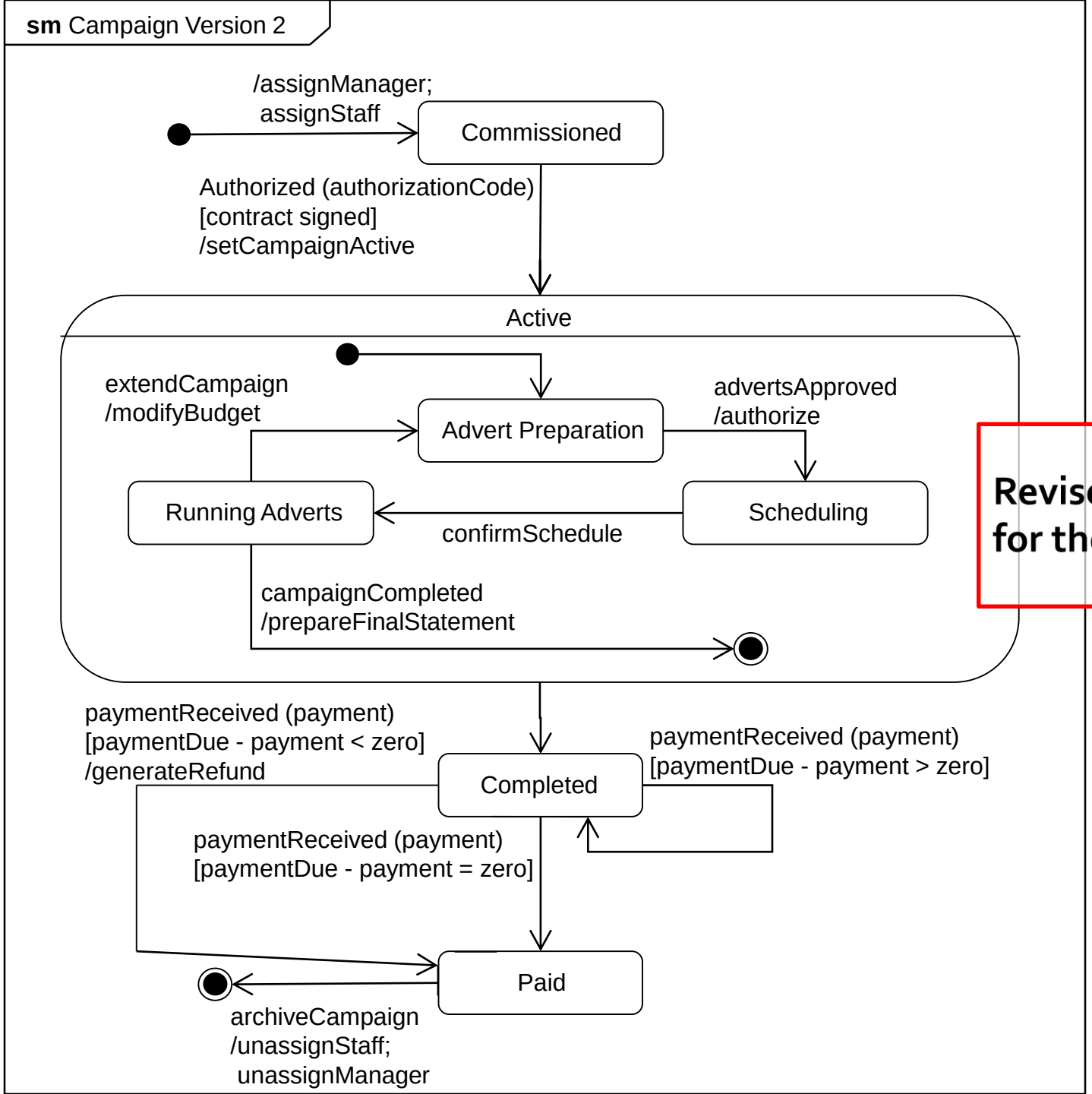
Preparing state machines

7. Iterate steps 4, 5 and 6 until the state machine captures the necessary level of detail.
8. Check the consistency of the state machine with the class diagram, with interaction diagrams and with any other state machines and models.

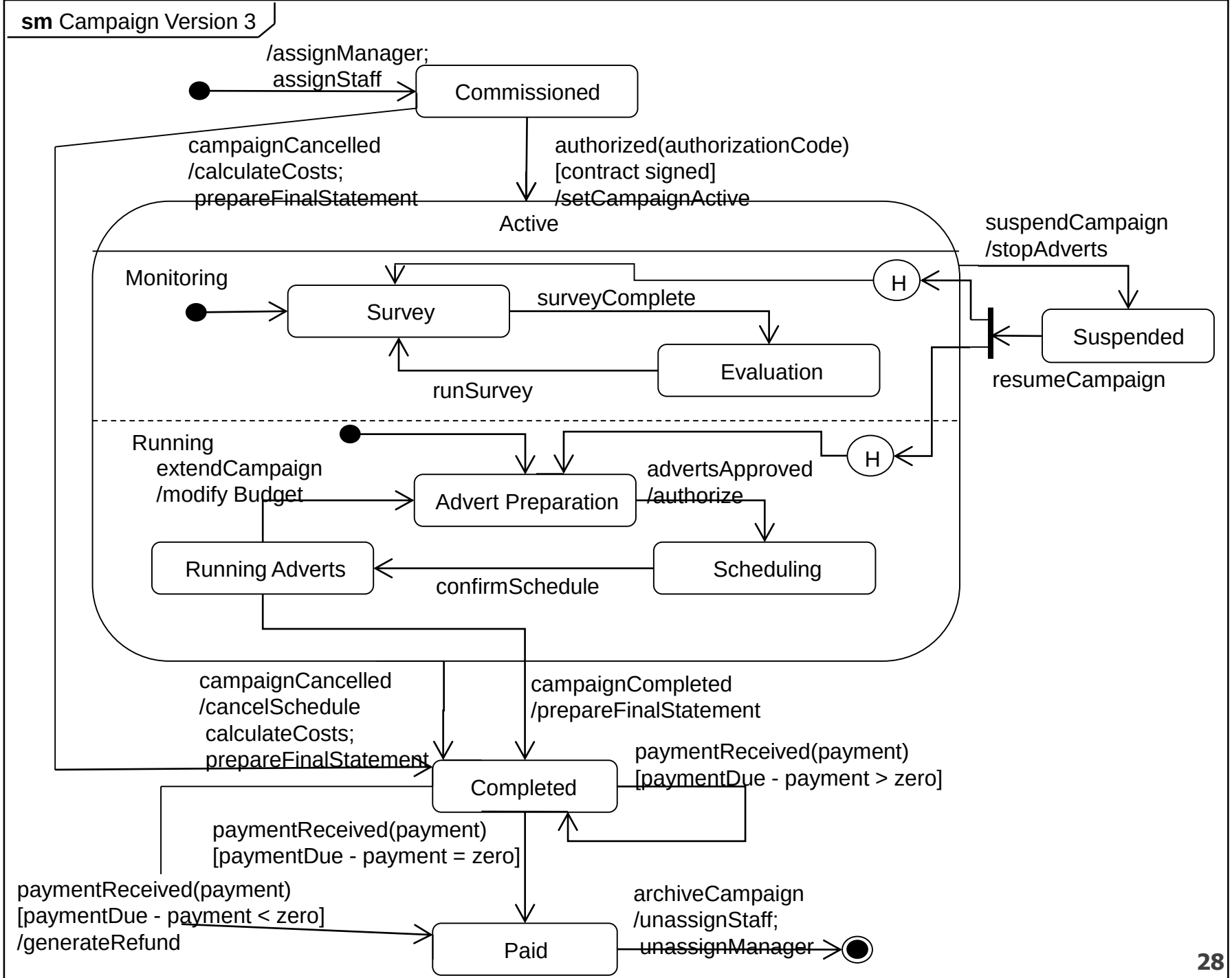

sd Record completion of a campaign
:CampaignManager
:CompleteCampaign
:Client
:Campaign
loop
[For all clients]
getClient
:CompleteCampaignUI
startInterface
Active
selectClient
showClientCampaigns
listCampaigns
loop
[For all client's campaigns]
getCampaignDetails()
completeCampaign
completeCampaign
completeCampaign
Completed
Active state
Completed state

Initial state machi ne for the Campaig n class—a behavio ural approach.





Revised state machine for the `Campaign` class


sm Campaign Version 3
/assignManager;
assignStaff
Commissioned
campaignCancelled
/calculateCosts;
prepareFinalStatement
authorized(authorizationCode)
[contract signed]
/setCampaignActive
Active
suspendCampaign
/stopAdverts
Monitoring
H
Survey
surveyComplete
Suspended
Evaluation
runSurvey
resumeCampaign
Running
extendCampaign
/modify Budget
H
advertsApproved
/authorize
Advert Preparation
Running Adverts
Scheduling
confirmSchedule
campaignCancelled
/cancelSchedule
calculateCosts;
prepareFinalStatement
campaignCompleted
/prepareFinalStatement
paymentReceived(payment)
[paymentDue - payment > zero]
Completed
paymentReceived(payment)
[paymentDue - payment = zero]
paymentReceived(payment)
[paymentDue - payment < zero]
/generateRefund
archiveCampaign
/unassignStaff;
unassignManager
Paid

Protocol State Machines

- UML 2.0 introduces a distinction between protocol and behavioural state machines.
- All the state machines so far have been behavioural.
- Protocol state machines differ in that they only show all the legal transitions with their pre- and post-conditions.

Protocol State Machines

- The states of a protocol state machine cannot have
  - entry, exit or do activity sections
- All transitions must be protocol transitions

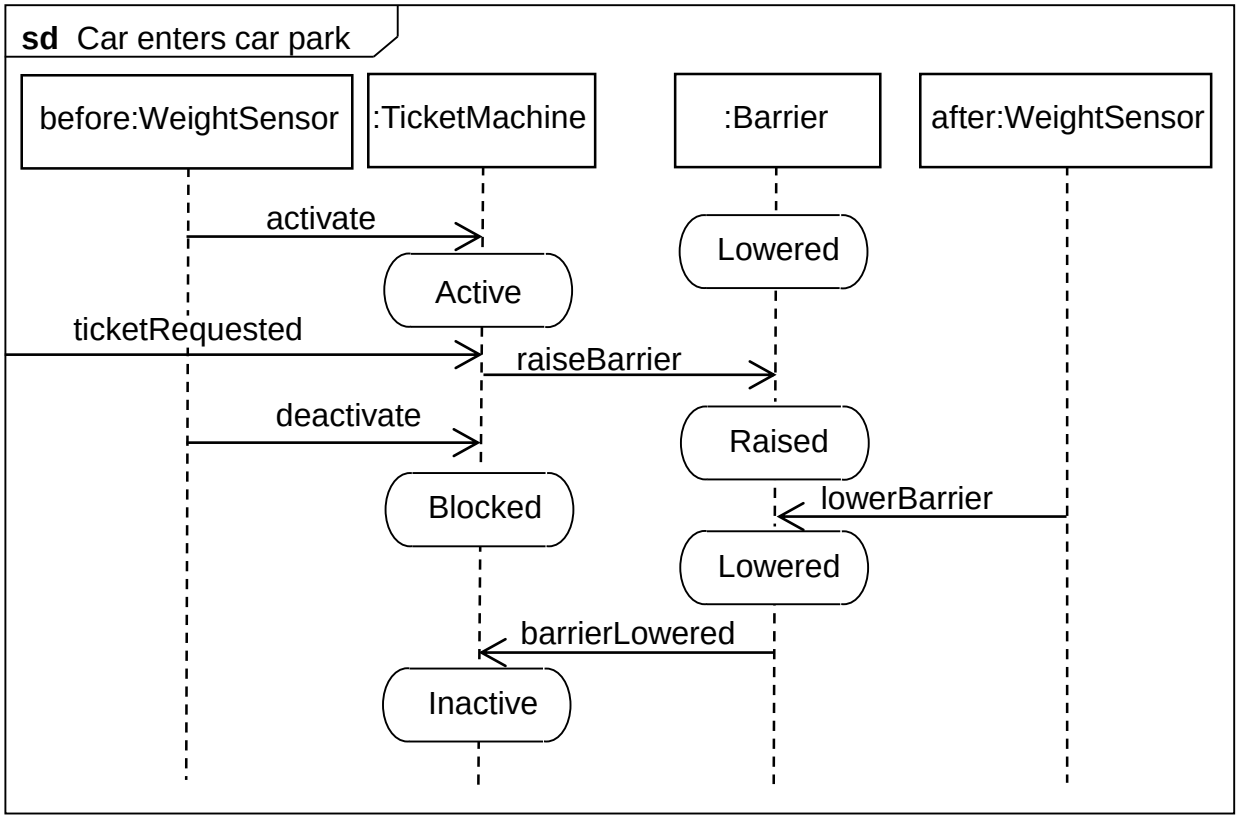
Protocol State Machines

- The syntax for a protocol transition label is as follows.

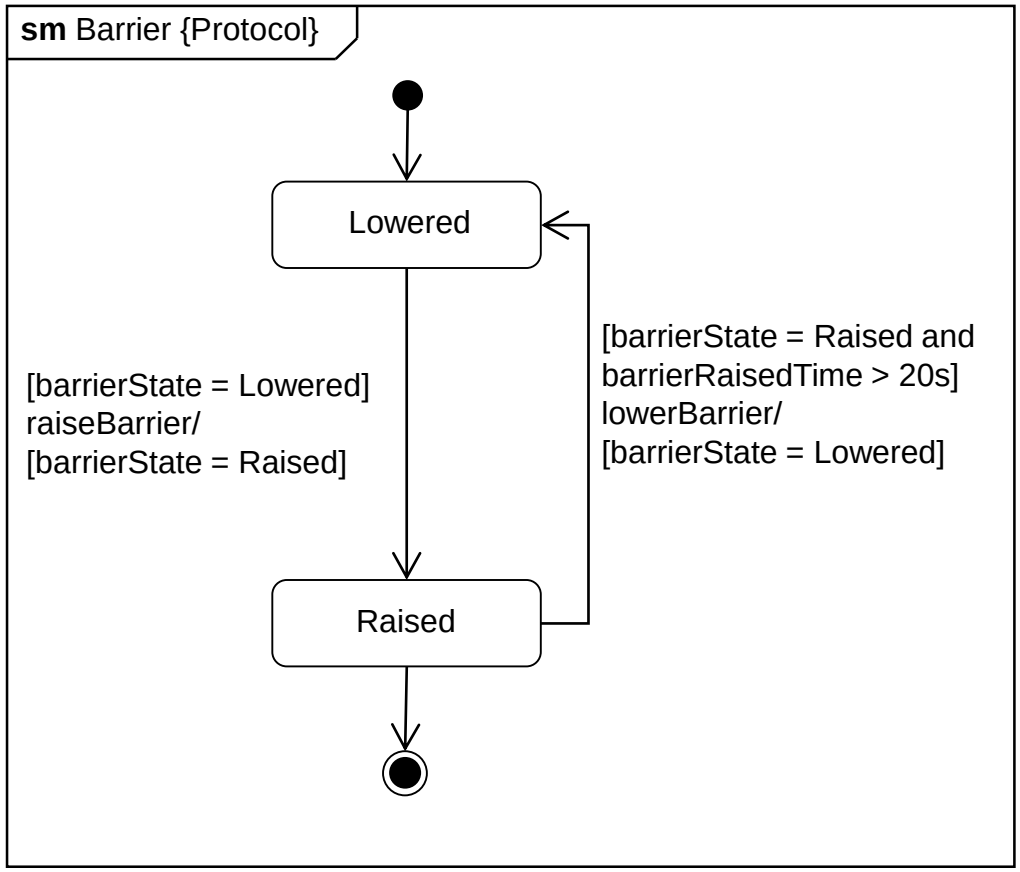
```
'[' pre-condition ']' trigger '/'      '['
  post-condition ']'
```

- Unlike behavioural transitions protocol transitions do not have activity expressions.

Sequence Diagram for Protocol State Machine Example



Protocol State Machine



Consistency Checking

- Every event should appear as an incoming message for the appropriate object on an interaction diagram(s).
- Every event should correspond to an operation on the appropriate class (but note that not all operations correspond to events).
- Every action should correspond to the execution of an operation on the appropriate class, and perhaps also to the dispatch of a message to another object.
- Every outgoing message sent from a state machine must correspond to an operation on another class.