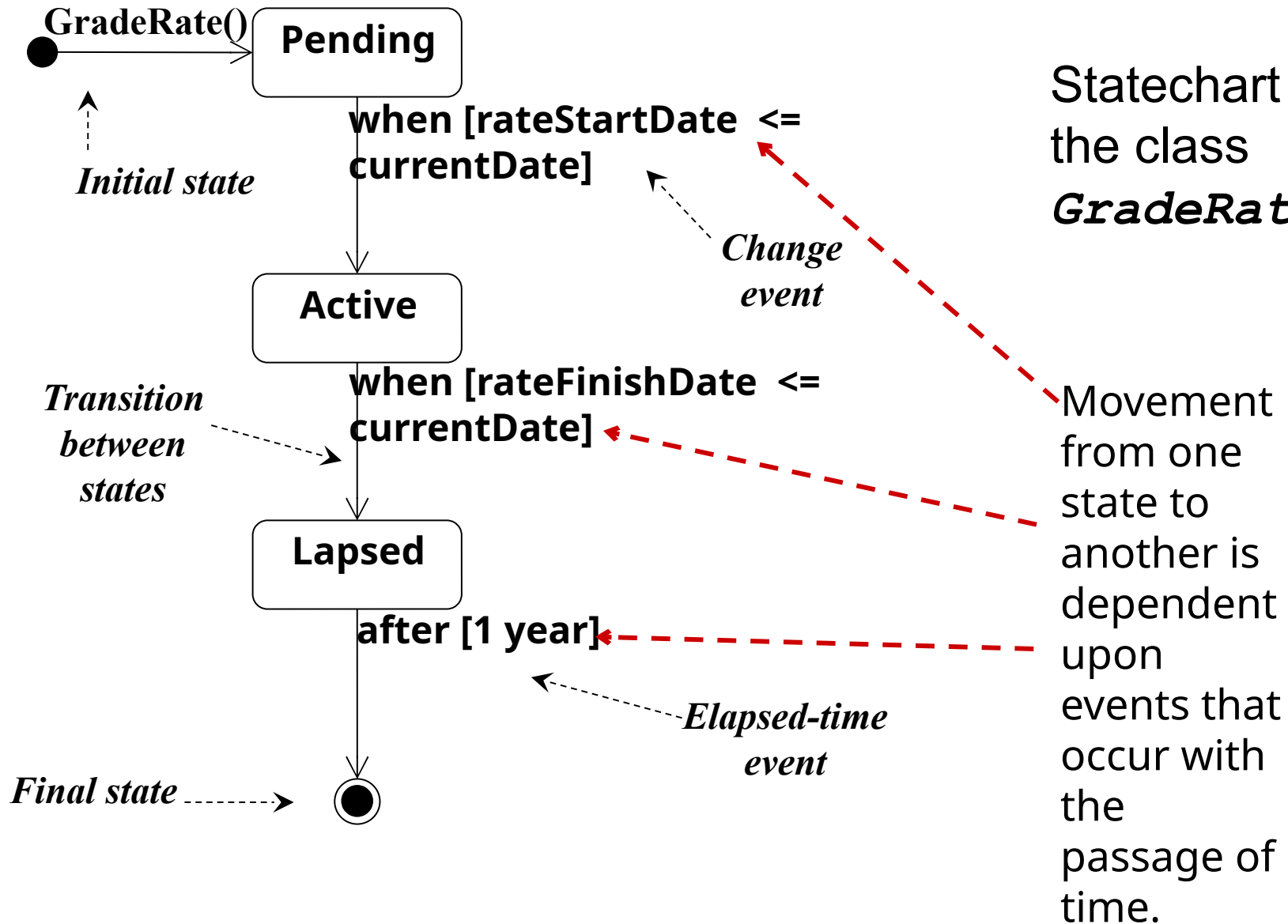**State**

- 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

- For example the class **StaffMember** has an attribute **startDate** which determines whether a **StaffMember** object is in the probationary state

- 'A state is a <span style="color:red">condition during the life of an object</span> or an interaction during which it satisfies some condition, performs some action or waits for some event....

**State Diagram and Statechart Diagram**

- **State Diagrams** are used to capture the behavior of a software system.

- UML State diagrams can be used to model the behavior of a class, a subsystem, a package, or even an entire system.

- It is also called a Statechart diagram.

- These diagrams are used to model the event-based system.

- A state of an object is controlled by an event.

- Statechart diagrams are used to describe various states of an entity within the application system.
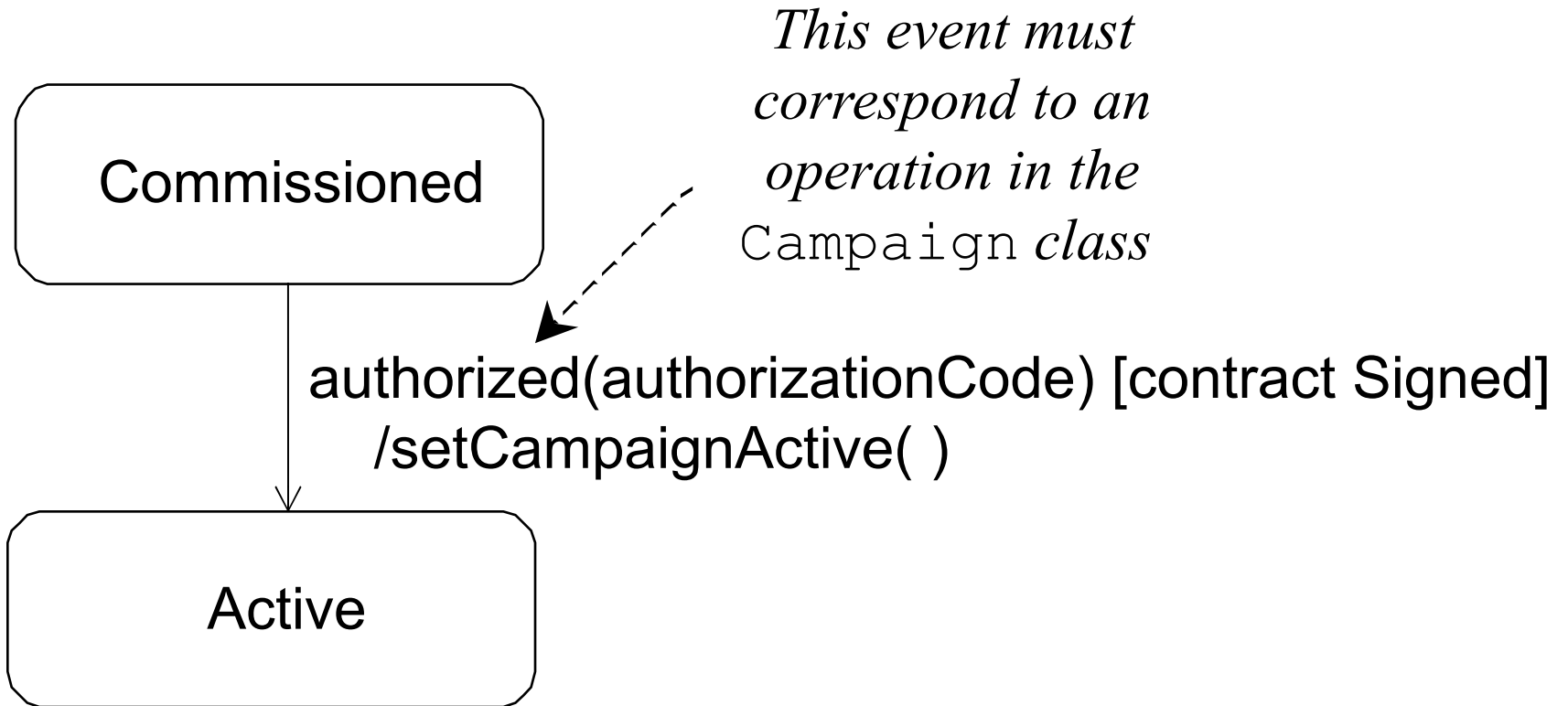
# Statechart Diagram

**GradeRate()**

**Pending**

*Initial state*

**when [rateStartDate <= currentDate]**

*Change event*

**Active**

*Transition between states*

**when [rateFinishDate <= currentDate]**

**Lapsed**

**after [1 year]**

*Elapsed-time event*

*Final state*

Statechart for the class *GradeRate*

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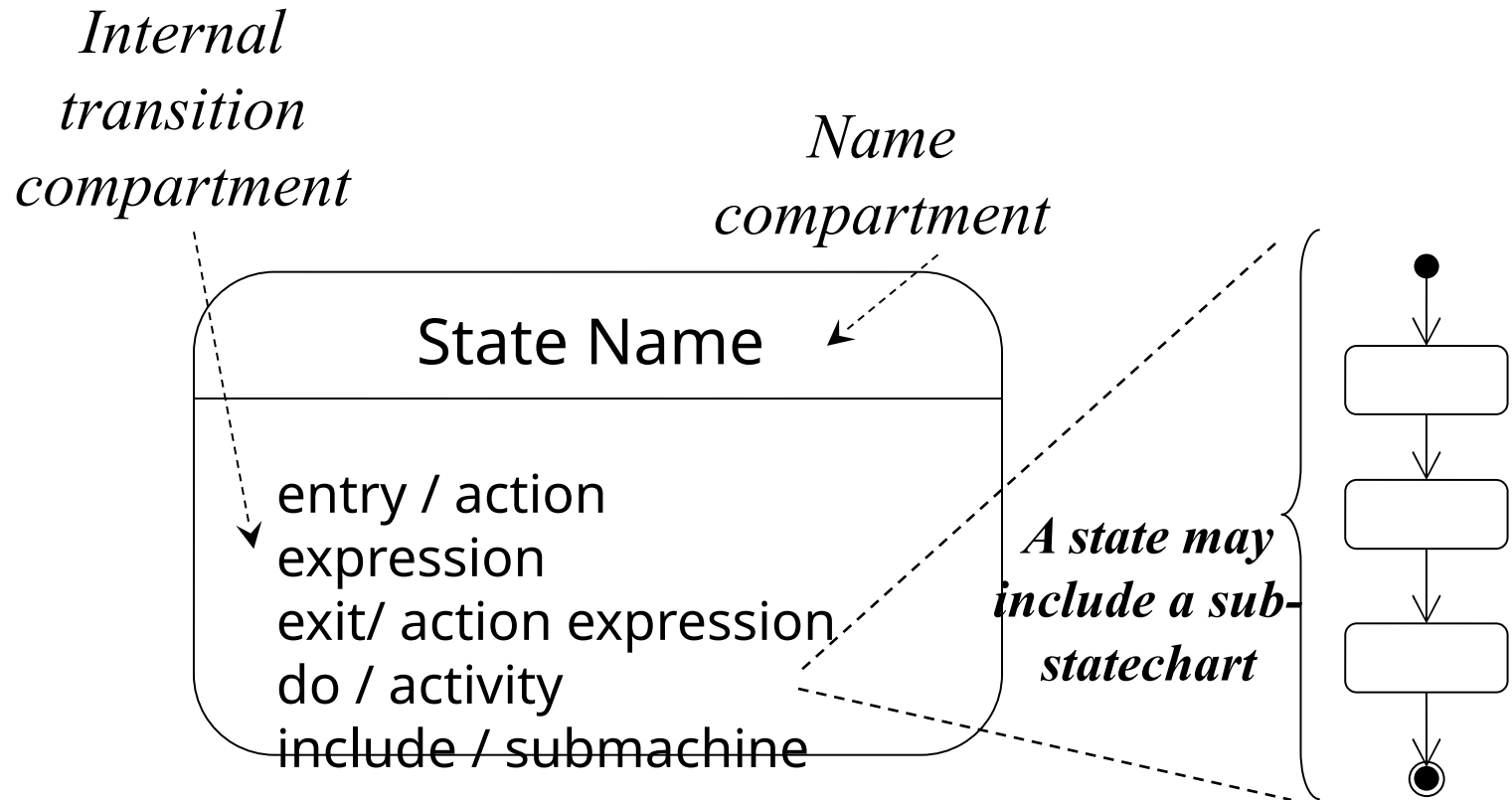
# Types of Event

- A *change event* occurs when a condition becomes true
- A *call event* occurs when an object receives a call to one of its operations either from another object or from itself
- A *signal event* occurs when an object receives a signal (an asynchronous communication)
- An *elapsed-time event* is caused by the passage of a designated period of time after a specified event (frequently the entry to the current state)
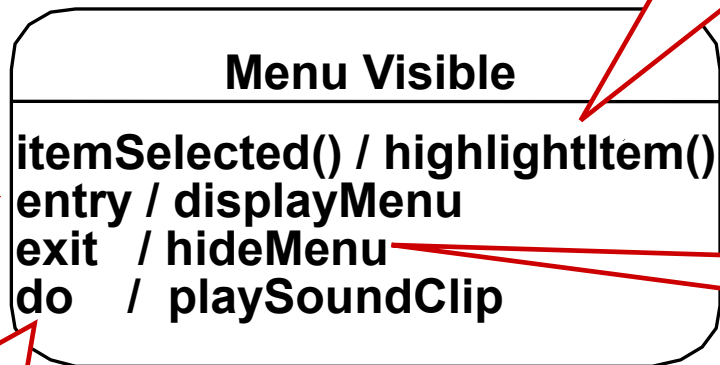
# Events



Commissioned

*This event must correspond to an operation in the* `Campaign` *class*

authorized(authorizationCode) [contract Signed] /setCampaignActive( )

Active

**Actions and Activities**

# Internal actions and activities for a state

*Internal transition compartment*

*Name compartment*

**State Name**

entry / action
expression
exit/ action expression
do / activity
include / submachine

*A state may include a sub-statechart*

# Actions and Activities

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

Event **`itemSelected()`**
triggers the action
**`highlightItem()`**

**Entry action
causes the menu
to be displayed**

### Menu Visible

**itemSelected() / highlightItem()**
**entry / displayMenu**
**exit   / hideMenu**
**do    / playSoundClip**

**Exiting the state
triggers
`hideMenu()`**

**While the object remains
in the `Menu Visible`
state, the activity causes a
sound clip to be played**
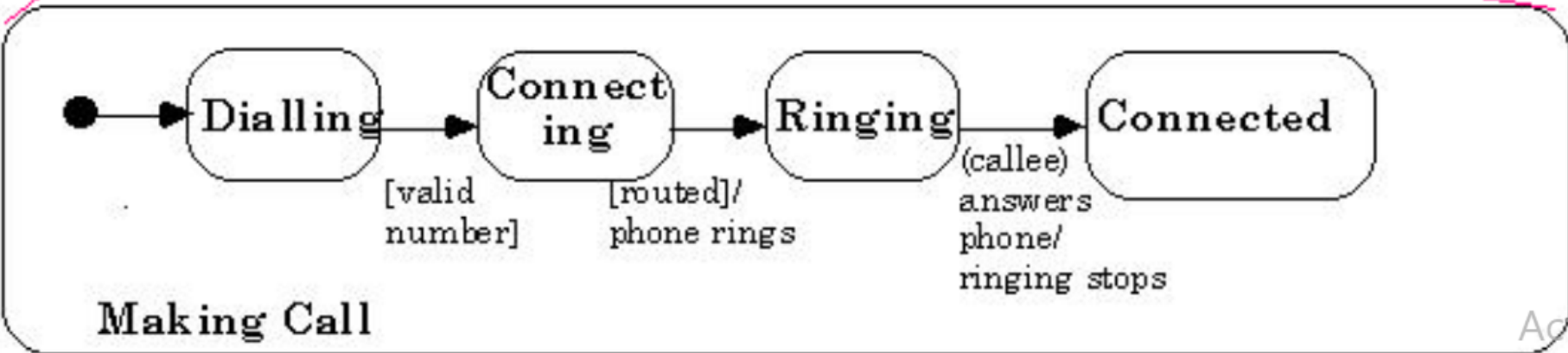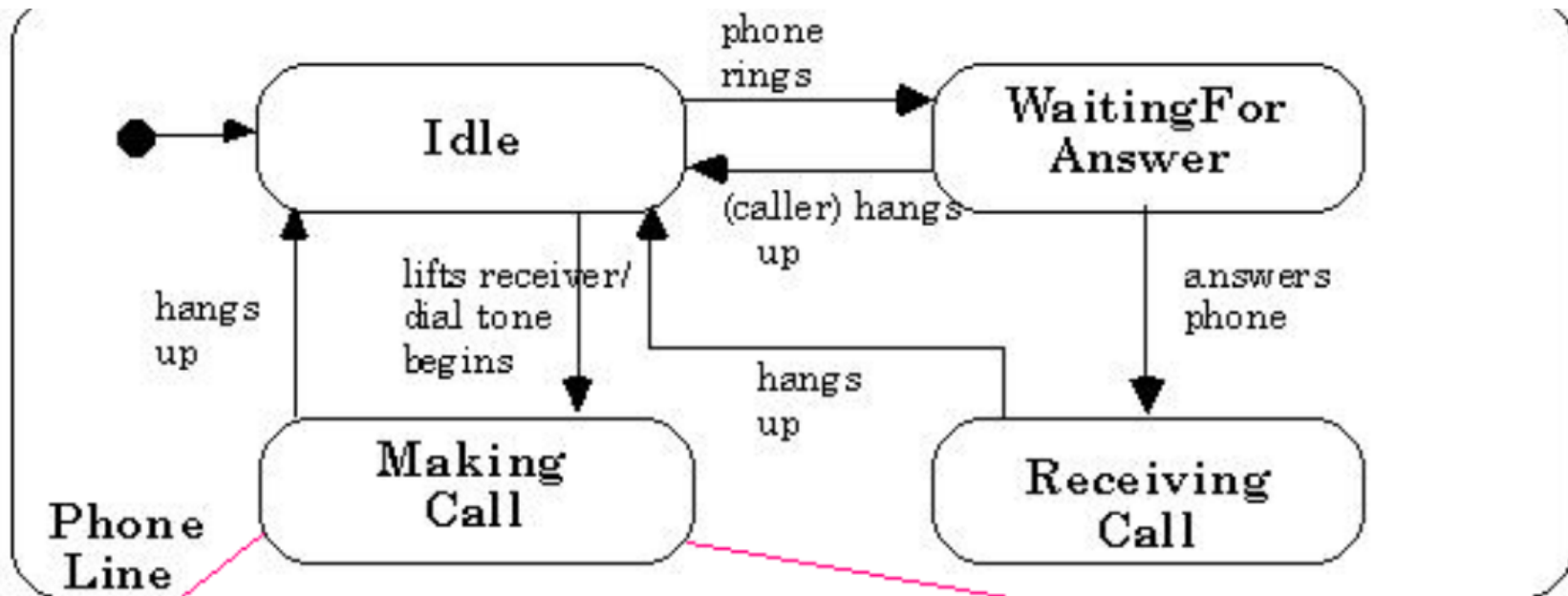
# Statechart diagram of an order management system
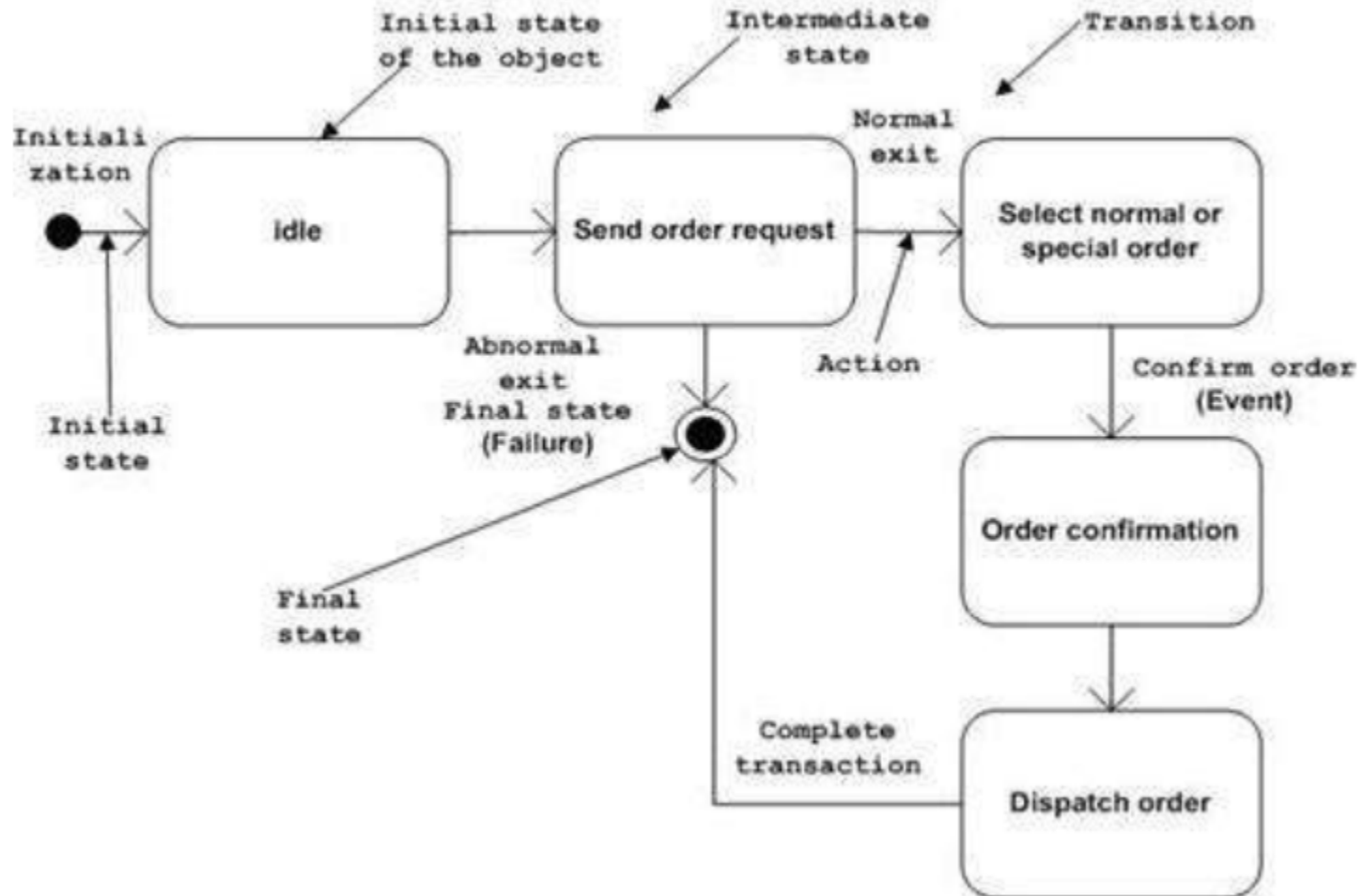
**Action-expression assigning manager and staff on object creation**

**/assignManager()**
**assignStaff()**

**Commissioned**

**authorized(authorizationCode)**
**[contract signed]**
**/setCampaignActive()**

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

**Active**

**Guard condition ensuring complete payment before entering `Paid`**

**campaignCompleted()**
**/prepareFinalStatement()**

**Completed**

**paymentReceived(payment)**
**[paymentDue – payment <= 0]**

**paymentReceived(payment)**
**[paymentDue - payment > 0]**

**Statechart for the class**
*Campaign*

**Paid**

**archiveCampaign()**
**/unassignStaff();**
**unassignManager()**

**A revised statechart for the class** *Campaign*

/assignManager();
assignStaff() **Commissioned**

●→ **Commissioned**

authorized(authorizationCode)
[contract signed]
/setCampaignActive()

**Active**

campaignCompleted()
/prepareFinalStatement()

paymentReceived(payment)
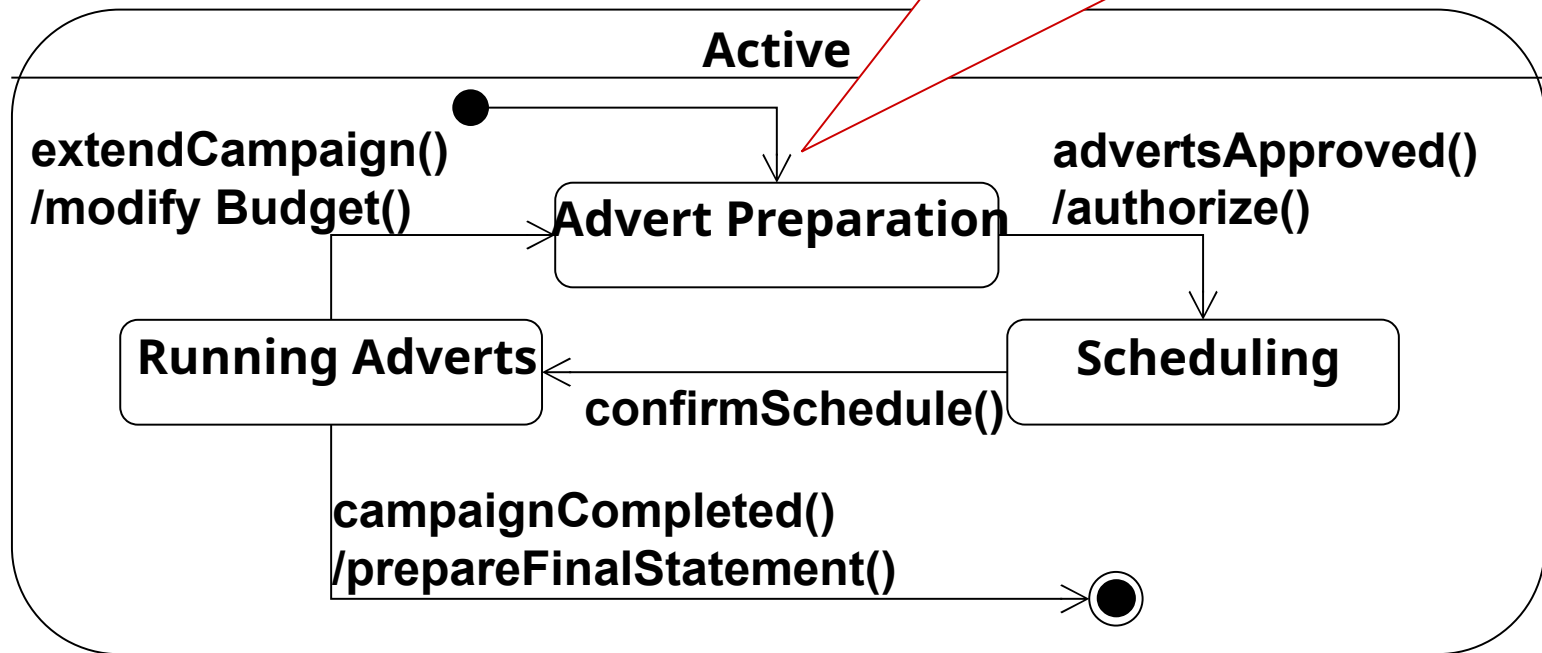[paymentDue - payment < 0]
/generateRefund()

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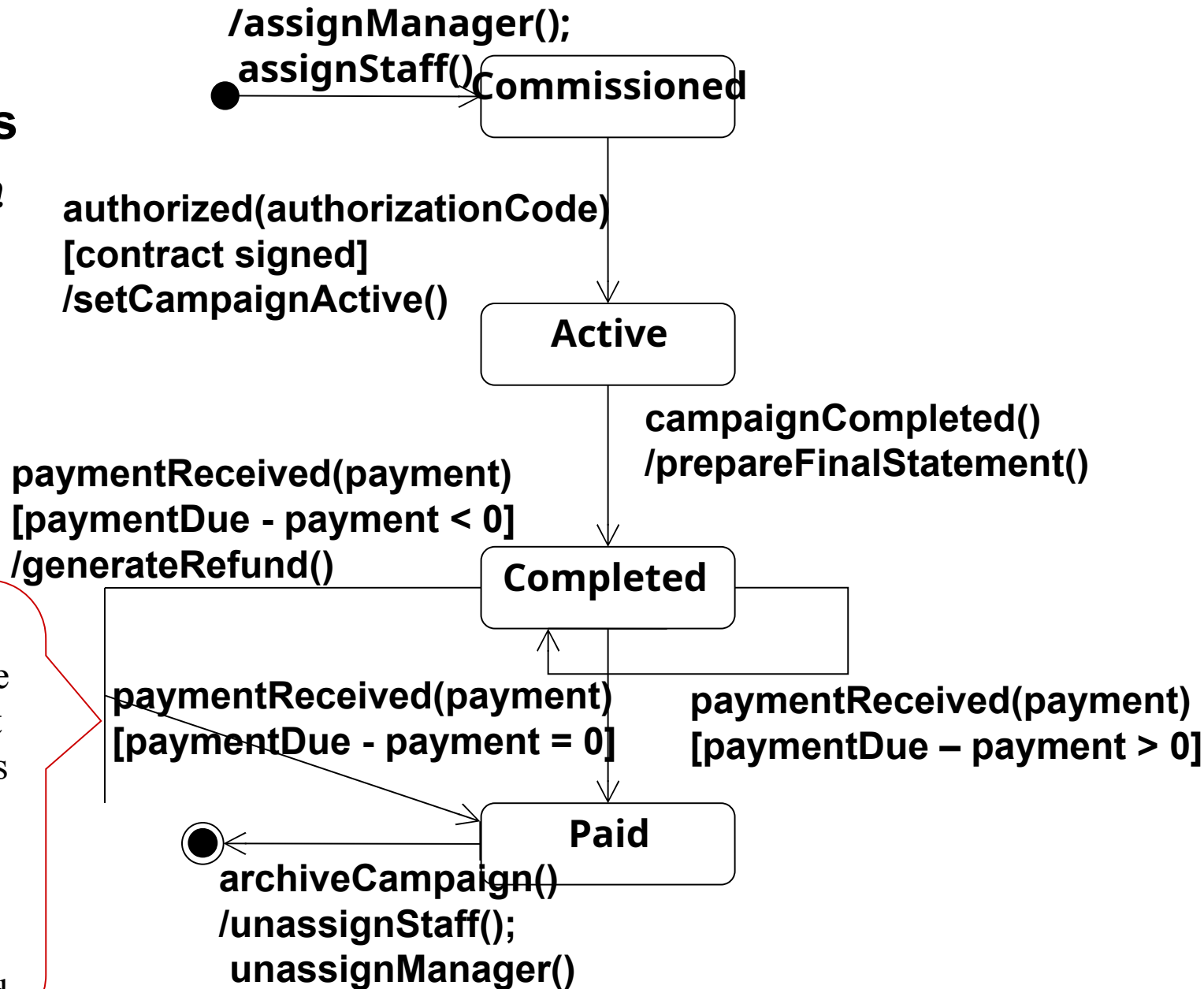
If the user requirements were to change, so that an overpayment is now to result in the automatic generation of a refund, a new transition is added

paymentReceived(payment)
[paymentDue - payment = 0]

paymentReceived(payment)
[paymentDue – payment > 0]

**Paid**

archiveCampaign()
/unassignStaff();
unassignManager()
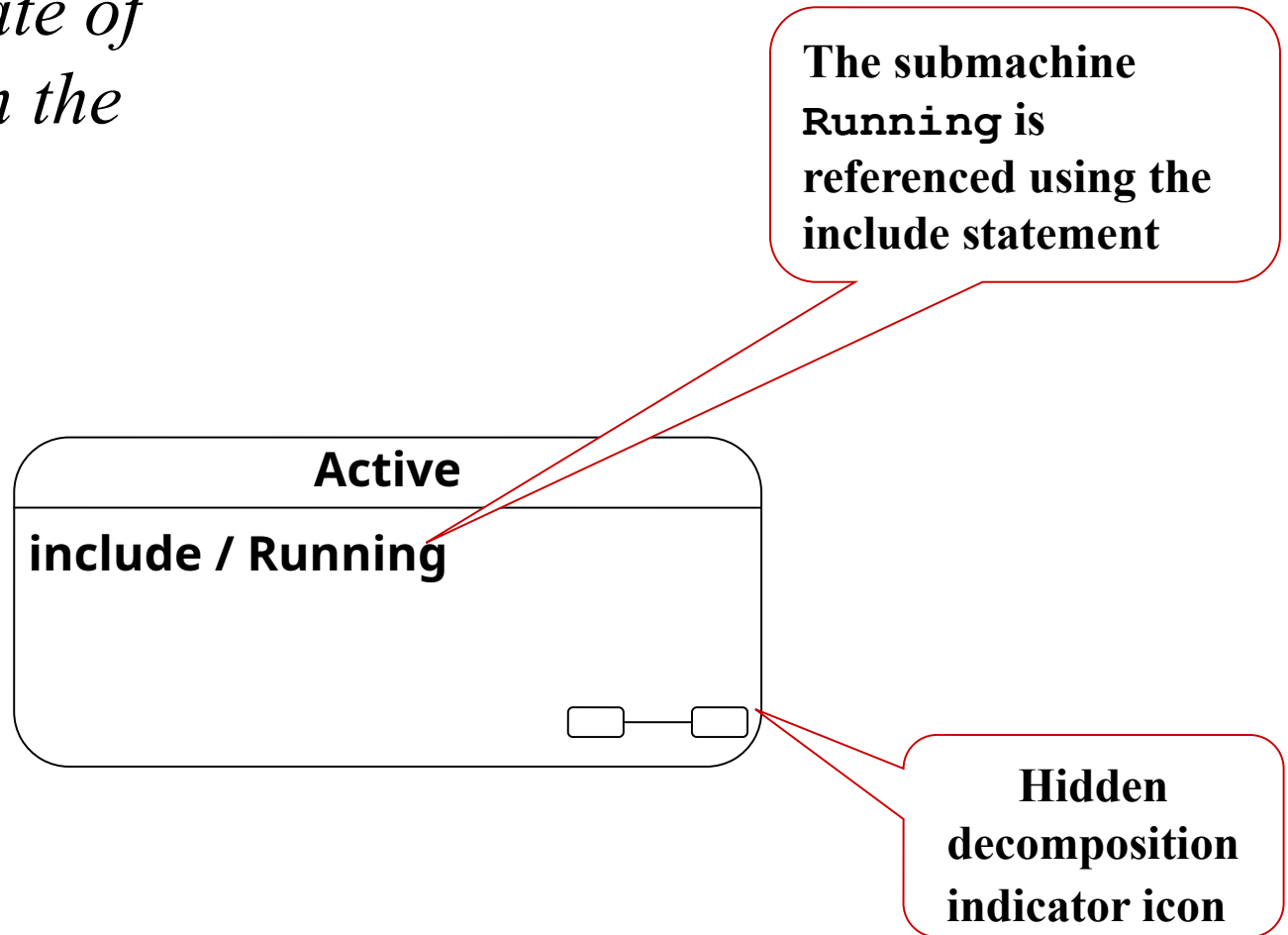
# Nested Substates

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

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

**Active**

extendCampaign()
/modify Budget()

**Advert Preparation**

advertsApproved()
/authorize()

**Running Adverts**

**Scheduling**

confirmSchedule()

campaignCompleted()
/prepareFinalStatement()

# Nested States

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

The submachine `Running` is referenced using the include statement

**Active**

**include / Running**

Hidden decomposition indicator icon

# The `Active` State with Concurrent Substates
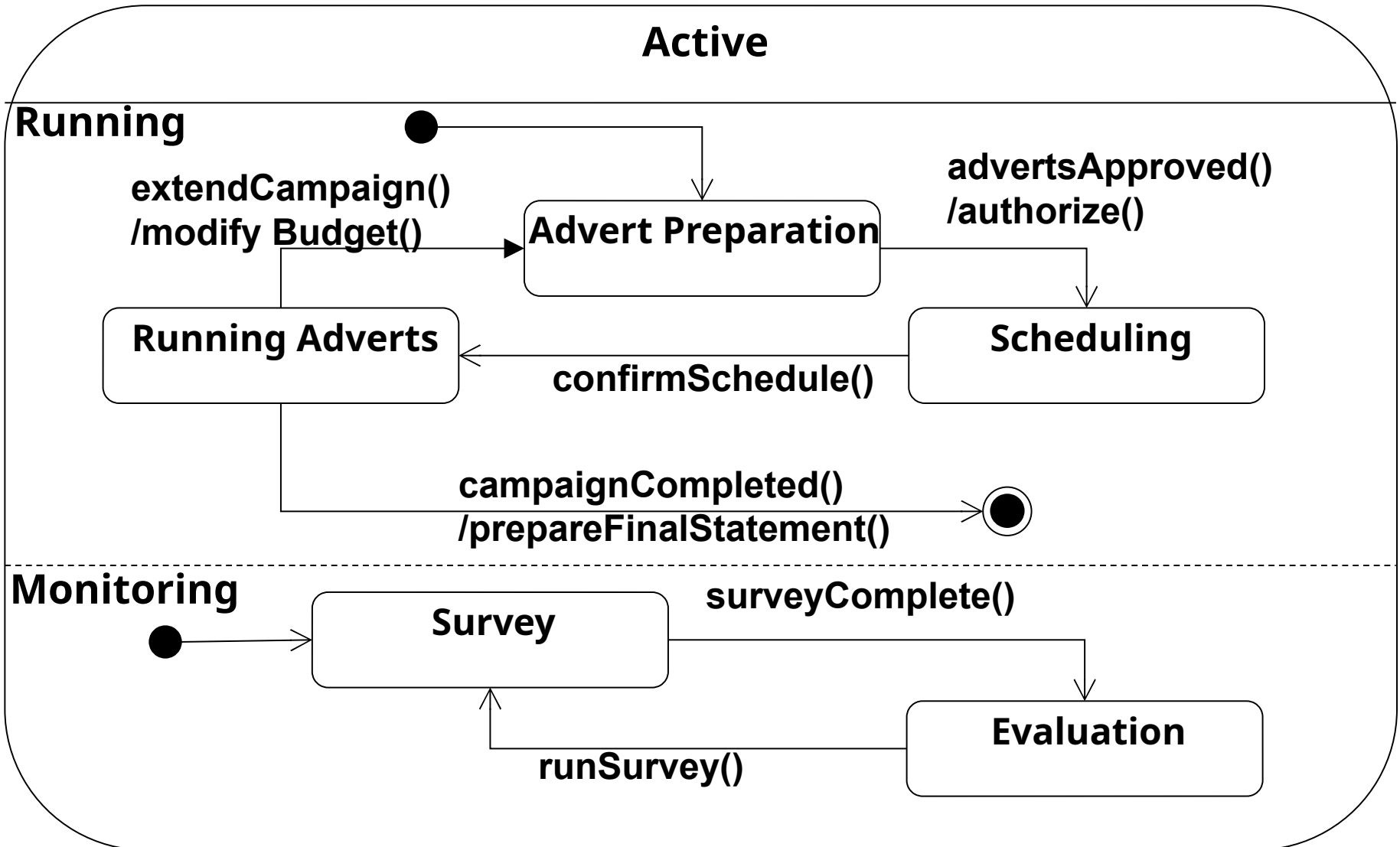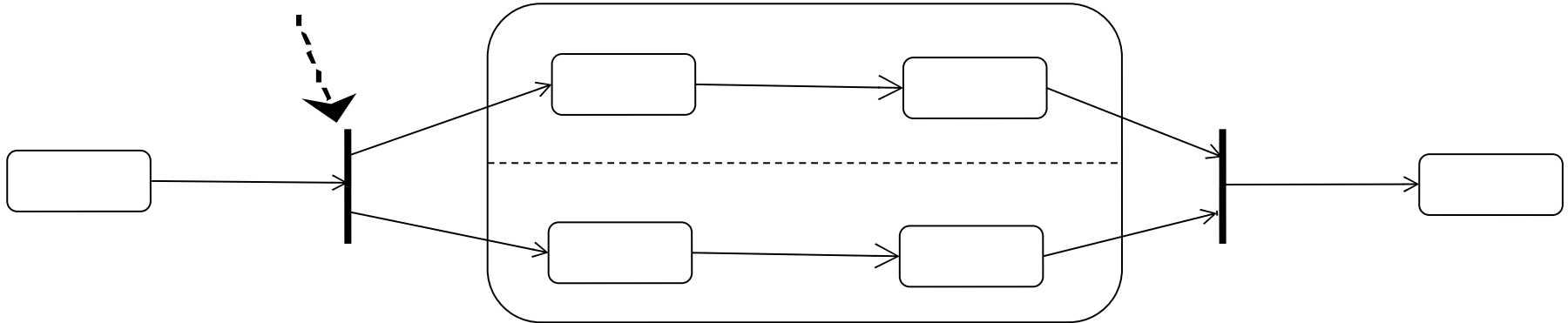
**Concurrent States**

- A transition to a complex state is equivalent to a simultaneous transition to the initial states of each concurrent statechart

- An initial state must be specified in both nested statecharts in order to avoid ambiguity about which substate should first be entered in each concurrent region

- A transition to the **`Active`** state means that the **`Campaign`** object simultaneously enters the **`Advert Preparation`** and **`Survey`** states

**Concurrent States**

- Once the composite states is entered a transition may occur within either concurrent region without having any effect on the state in the other concurrent region

- A transition out of the `Active` state applies to all its substates (no matter how deeply nested)

# Synchronized Concurrent Threads.

**Synchronization bar**



- Explicitly showing how an event triggering a transition to a state with nested concurrent states causes specific concurrent substates to be entered
- Shows that the composite state is not exited until both concurrent nested statecharts are exited

# Preparing Statecharts

- Two approaches may be used:
  - Behavioural approach
  - Life cycle approach

# Behavioural Approach

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 statechart.

4. Elaborate the statechart as necessary to cater for additional interactions as these become evident, and add any exceptions.

## Behavioural Approach

5. Develop any nested statecharts (unless this has already been done in an earlier step).

6. Review the statechart to ensure consistency with use cases. In particular, check that any constraints that are implied by the statechart are appropriate.

7. Iterate steps 4, 5 and 6 until the statechart captures the necessary level of detail.

8. Check the consistency of the statechart with the class diagram, with interaction diagrams and with any other statecharts and models.

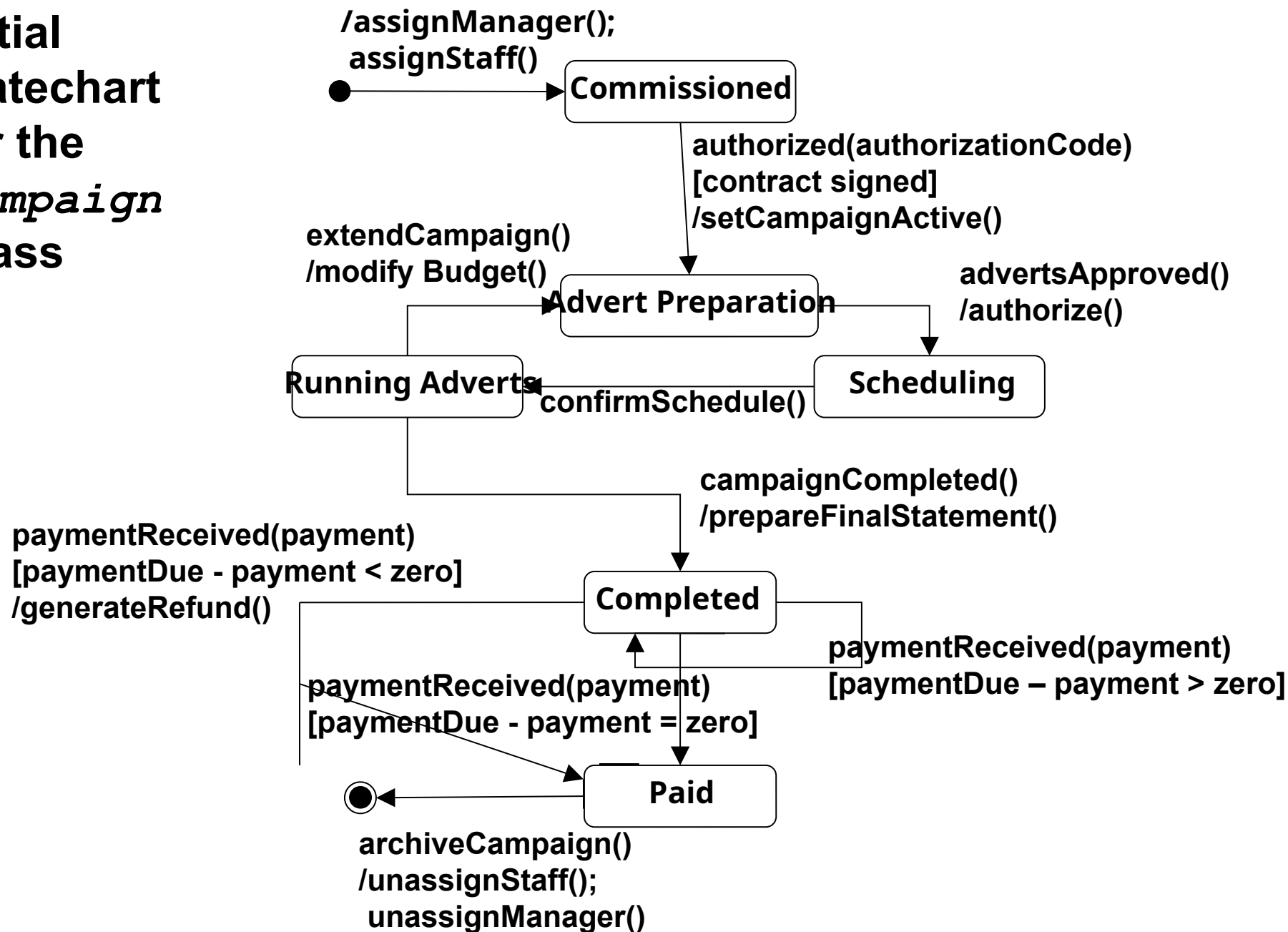# Sequence Diagram with Implicit States

*Sequence diagram for use case* `Set Campaign Completed`
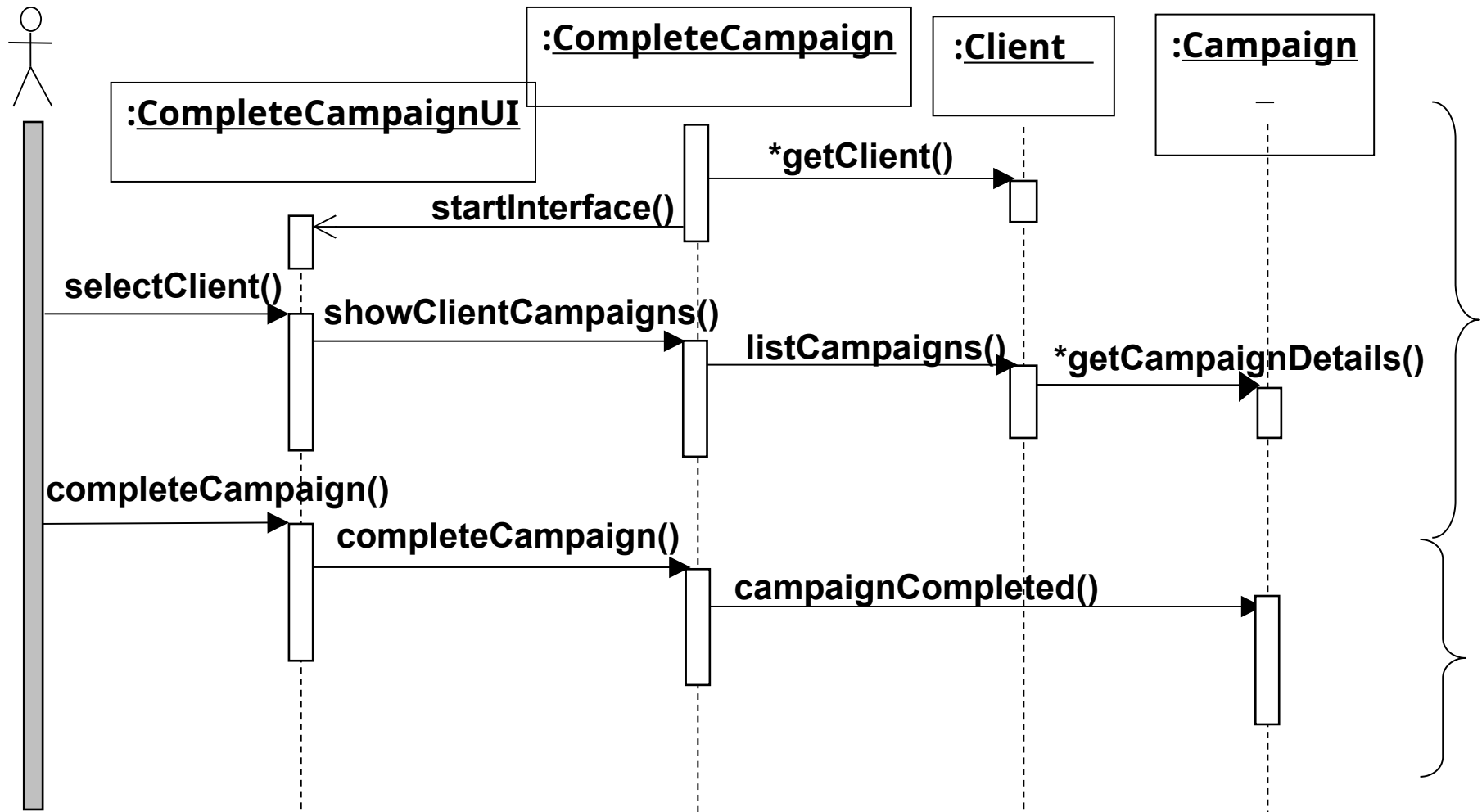
**Initial statechart for the** *Campaign* **Class**

**Revised statechart**

**for the**
*Campaign*

**class**



**/assignManager();**
**assignStaff()**

**Commissioned**

**authorized(authorizationCode)**
**[contract signed]**
**/setCampaignActive()**

**Active**

**extendCampaign()**
**/modify Budget()**

**advertsApproved()**
**/authorize()**

**Advert Preparation**

**Running Adverts**

**Scheduling**

**confirmSchedule()**

**campaignCompleted()**
**/prepareFinalStatement()**

**paymentReceived(payment)**
**[paymentDue - payment < 0]**
**/generateRefund()**

**Completed**

**paymentReceived(payment)**
**[paymentDue - payment = 0]**

**paymentReceived(payment)**
**[paymentDue – payment > 0]**

**Paid**

**archiveCampaign()**
**/unassignStaff();**
**unassignManager()**

23

**Final version of *Campaign* statechart**

start

/get first item

[Not all items checked]
/get next item

[All items checked &&
all items available]

**Checking**

do/check
item

**Dispatching**

do/initiate
delivery

[All items checked &&
some items not in stock]

Item Received
[all items available]

Delivered

Item Received
[some items not in stock]

*activity*

*transition*

**Waiting**

**Delivered**

*self-transition*

*State*