State Chart

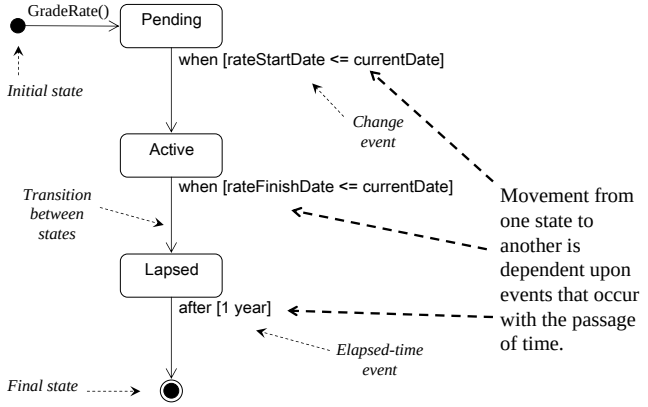
Lecture 4C

•A statechart describes the possible dynamic behavior and associated state changes of a class.

•Statecharts can also be used to model the dynamic behavior of other classifiers

•E.g. sub-systems and use cases.

Basic Notation for Statechart



There are three basic component in Statechart

•**Event** – Occurrence that is relevant to an object or application.

•**State** – The state of an object is determined by the value of some of its attributes and the presence or absences of links with other objects.

•**Transition** – The movement from one state to another, triggered by an event.

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

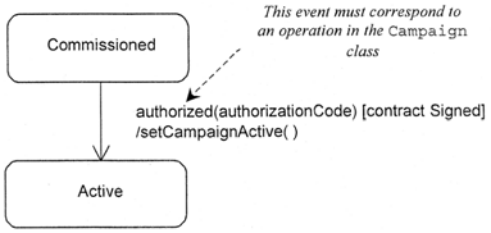
•A **Guard Condition** is evaluated when a particular event occurs and only if the condition is true does the associated transition fire.

•All the guard conditions from a state should be mutually exclusive so that for each set of circumstances there is only one valid transition from a state.

•If they are not mutually exclusive more than one transition may be valid and the behavior of the statechart is indeterminate.

State and Events

- A **State** is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action or waits for some event.
- An **Event** is an occurrence of a stimulus that can trigger a state change and that is relevant to the object or to an application



3

A **State** is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action or waits for some event.

•Conceptually, an object remains in a state for an interval of time. However, the semantics allow for modelling "flow-through" states which are instantaneous as well as transitions that are not instantaneous.

•The current state of an object is a result of the events that have occurred to the object, and is determined by the current value of the object's attributes and the links that it has with other objects.

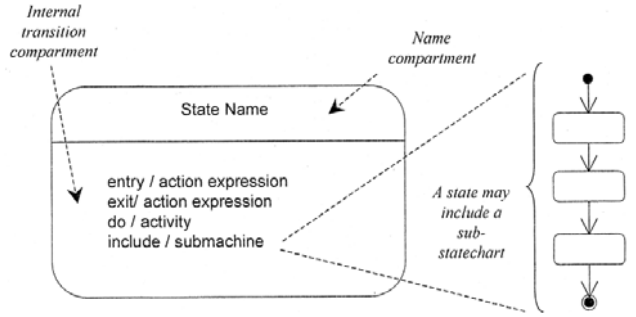
•Some attributes and links of an object are significant for the determination of its state while others are not.

- •For example, in the Agate case study `staffName` and `staffNo` attributes of a `StaffMember` object have no impact upon its state, whereas the date that a staff member started his or her employment at Agate determines when the probationary period of employment ends (after six months, say).
- •The `StaffMember` object is in the `Probationary` state for the first six months of employment.
- •While in this state, a staff member has different employment rights and is not eligible for redundancy pay in the event that they are dismissed by the company.

Events can be grouped into several general types.

- •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)

Actions and Activities

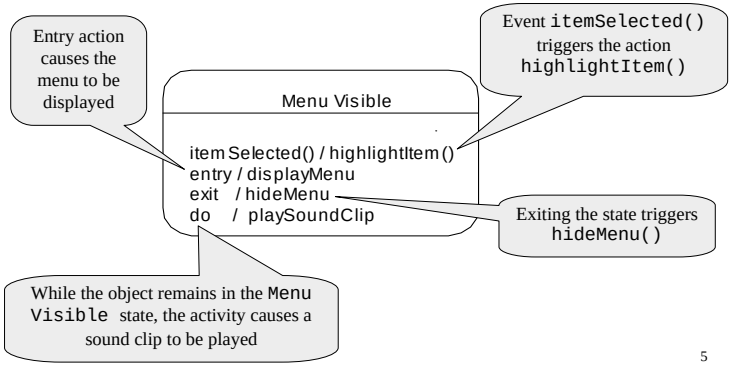


4

•An action is considered to be instantaneous (its duration is actually determined by the processing environment) while an activity persists for the duration of a state, for example (its duration is dependent upon the occurrence of events that may cause a state change).

Example

- Menu Visible state for a DropDownMenu object



5

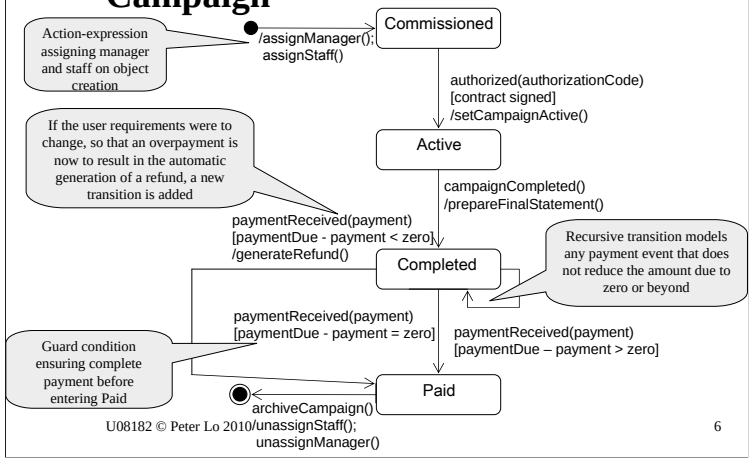
•The entry action causes the menu to be displayed.

•While the object remains in the `Menu Visible` state, the activity causes a sound clip to be played and, if the event `itemSelected()` occurs, the action `highlightItem()` is invoked.

•It is important to note that when the event `itemSelected()` occurs the `Menu Visible` state is not exited and entered and as a result the exit and entry actions are not invoked.

•When the state is actually exited the menu is hidden.

Example: Statechart for the class Campaign



6

•The recursive transition from the `Completed` state models any payment event that does not reduce the amount due to zero or beyond.

•Only one of the two transitions from the `Completed` state (one of which is recursive) can be triggered by the `paymentReceived` event since the guard conditions are mutually exclusive.

•It would be bad practice to construct a statechart where one event can trigger two different transitions from the same state.

•A life cycle is only unambiguous when all the transitions from each state are mutually exclusive.

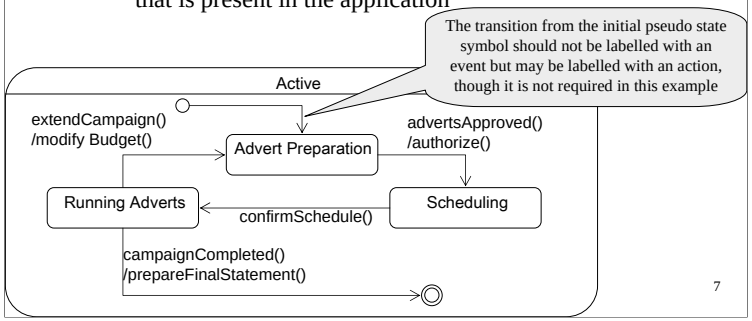
•If the user requirements were to change, so that an overpayment is now to result in the automatic generation of a refund, the statechart can be changed as follows.

•Since the action that results from an overpayment is different from the action that results from a payment that reduces *`paymentDue`* to zero, a new transition is needed from the *`Completed`* state to the *`Paid`* state.

•The guard conditions from the *`Completed`* state must also be modified.

Nested Substates

- When the state behavior for an object or an interaction is complex it may be necessary to represent it at different detail levels of detail and to reflect any hierarchy of states that is present in the application



In the nested statechart within the ***`Active`*** state, there is an initial state symbol with a transition to the first substate that a `Campaign` object enters when it becomes active.

The transition from the initial pseudostate symbol to the first substate (***`Advert Preparation`***) should not be labelled with an event but it may be labelled with an action, though it is not required in this example.

It is implicitly fired by any transition to the ***`Active`*** state.

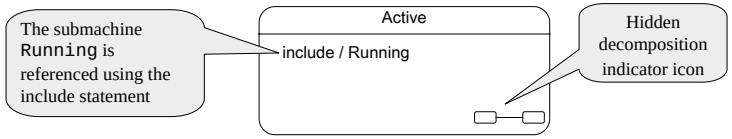
A final pseudostate symbol may also be shown on a nested state diagram.

A transition to the final pseudostate symbol represents the completion of the activity in the enclosing state (i.e. ***`Active`***) and a transition out of this state triggered by the completion event.

This transition may be unlabelled (as long as this does not cause any ambiguity) since the event that triggers it is implied by the completion event.

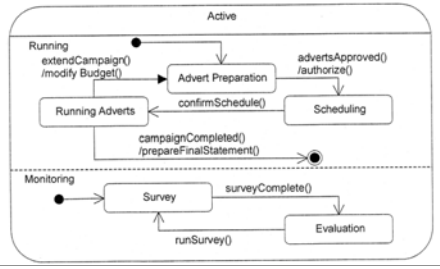
Nested States

- A high level statechart for the class can be drawn to include within the main diagram the detail that is shown in the nested statechart if so desired.
- If the detail of the submachine is not required on the higher level statechart or is just too much to show on one diagram the higher level statechart can be annotated with the hidden composition indicator icon.



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)



•An object can be in concurrent states if it has a complex composite state with concurrent substates.

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

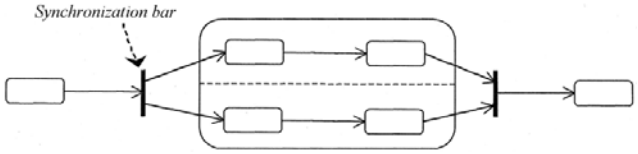
•An initial state must be specified in both nested statecharts in order to avoid ambiguity about which substate should first be entered in each concurrent region

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

> •E.g. Suppose that further investigation reveals that at Agate a campaign is surveyed and evaluated while it is also active.
>
> •A campaign may occupy either the ***`Survey`*** substate or the ***`Evaluation`*** substate when it is in the ***`Active`*** state. Transitions between these two states are not affected by the campaign's current state in relation to the preparing and running of adverts.
>
> •We model this by splitting the ***`Active`*** state into two concurrent nested statecharts, ***`Running`*** and ***`Monitoring`***, each in a separate sub-region of the `Active` statechart.
>
> •This is shown by dividing the state icon with a dashed line.

Synchronized Concurrent Threads

- 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

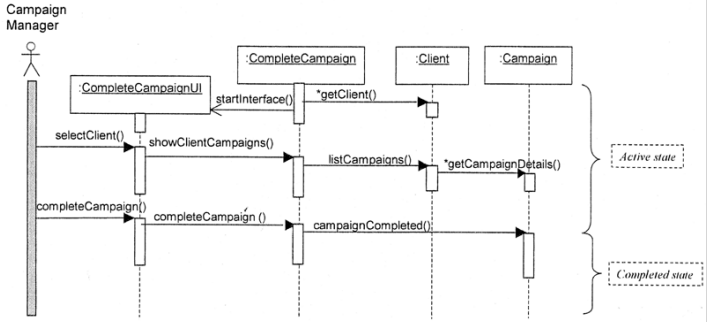
Behavioural Approach Steps

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.
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.

Life Cycle Approach Steps

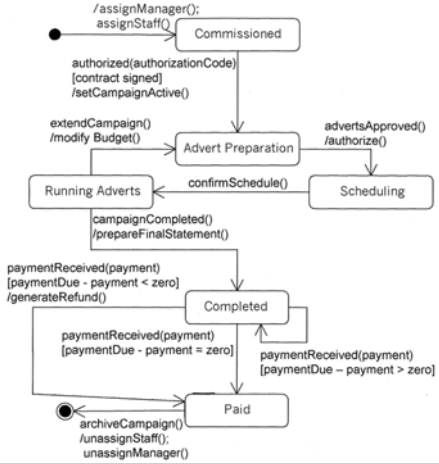
1. Identify major system events.
2. Identify each class that is likely to have a state dependent response to these events.
3. For each of these classes produce a first-cut statechart by considering the typical life cycle of an instance of the class.
4. Examine the statechart and elaborate to encompass more detailed event behaviour.
5. Enhance the statechart to include alternative scenarios.
6. Review the statechart to ensure that is consistent with the use cases. In particular, check that the constraints that the statechart implies are appropriate.
7. Iterate through steps 4, 5 and 6 until the statechart captures the necessary level of detail.
8. Ensure consistency with class diagram and interaction diagrams and other statecharts.

Drawing Sequence Diagram with Implicit States



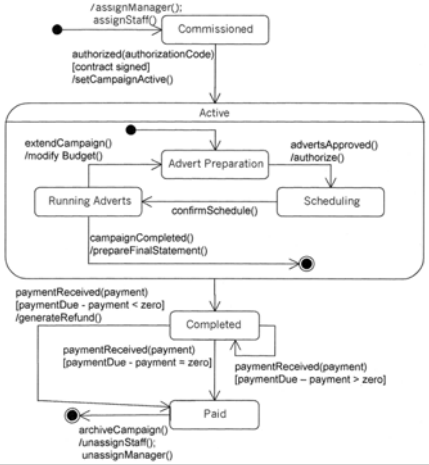
1. Sequence diagram for use case `Set Campaign Completed`
2. Note that getCampaignDetails() does not change the state as this message only queries aspects of the objects state.
3. However, campaignCompleted() does cause a state change and changes the values of some attributes of the object.

Initial Statechart for the Campaign



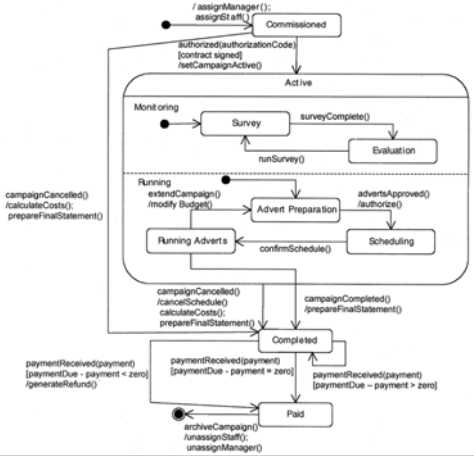
13

Revised Statechart for the Campaign



14

Final version of Campaign Statechart



15

Consistency Checking

- A statechart should be cross-checked with relevant class diagrams to ensure that the class specifications are consistent, with interaction sequence diagrams and collaboration diagrams that involve this class (and hence its statechart), with any nested statecharts and also with any linked activity diagrams (e.g. parent activity diagram).
- Consistency checking is an important task in the preparation of a complete set of models
- Highlights omissions and errors, and encourages the clarification of any ambiguity or incompleteness in the requirements
- The consistency checks should include class names, location of operations and signature of operations.

16

The need for consistency between different models in relation to interaction diagrams. Statecharts must also be consistent with other models.

•Every event should appear as an incoming message for the appropriate object on an interaction diagram.

•Every action should correspond to the execution of an operation on the appropriate class, and perhaps also to the despatch of a message to another object.

•Every event should correspond to an operation on the appropriate class (but note that not all operations correspond to events).

•Every outgoing message sent from a statechart must correspond to an operation on another class.

•Consistency checks are an important task in the preparation of a complete set of models. This process highlights omissions and errors, and encourages the clarification of any ambiguity or incompleteness in the requirements.