

PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



Chapter 10: **Statecharts**



Specifying Behaviour

- Interaction diagrams
 - show how object behave in particular interactions
 - do not specify all the possible behaviours of objects
- Different notation is needed to summarize the overall behaviour of objects
- UML defines *statecharts* for this purpose

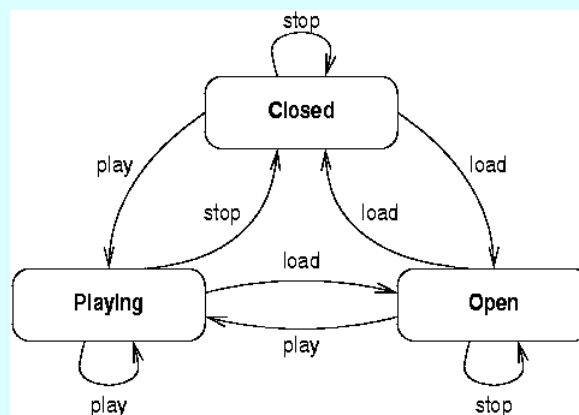


State-dependent Behaviour

- Objects respond differently to the same stimulus at different times
- This is modelled by defining a set of *states*
 - an object can be in one state at any time
 - the state it is in determines how it responds to *events* detected or messages received
 - in particular, an event can cause the object to move from one state to another (a *transition*)

States, Events and Transitions

- A simple statechart for a CD player:

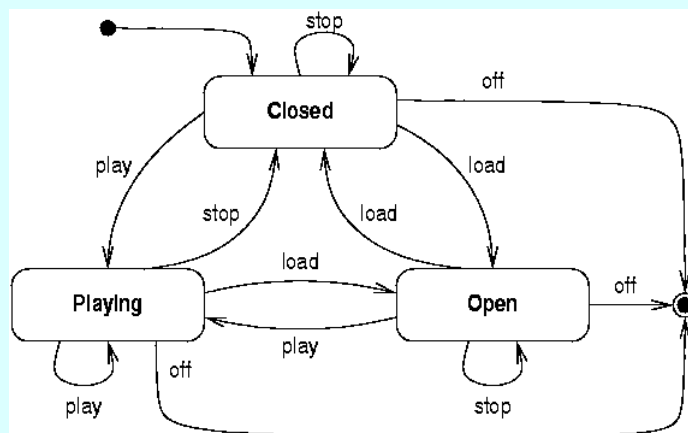


Statechart Semantics

- A *statechart* defines the behaviour of instances of a given class
- An object is in one *active state* at a time
- *Events* may be received at any time
- An event can *trigger* a transition
 - a transition from the active state will *fire* if it is labelled with the event just received
- The transition leads to the next active state

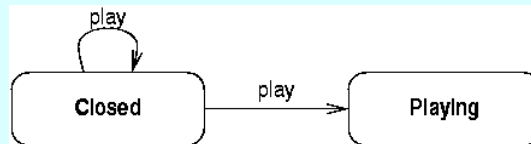
Initial and Final States

- Model the creation and destruction of objects



Non-determinism

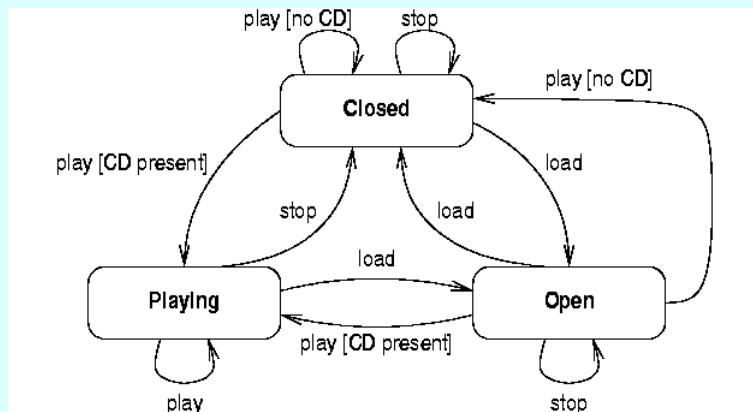
- Sometimes there are two transitions with the same event label leaving a state



- Some systems are non-deterministic
 - but usually the non-determinism can be removed by adding more information

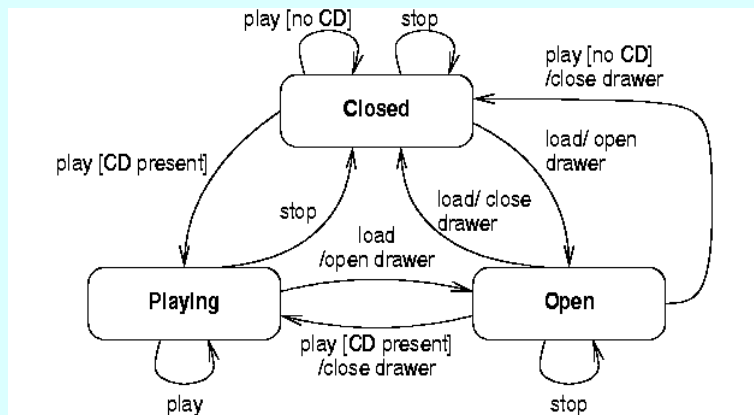
Guard Conditions

- *Conditions* added to events indicate when a transition can fire



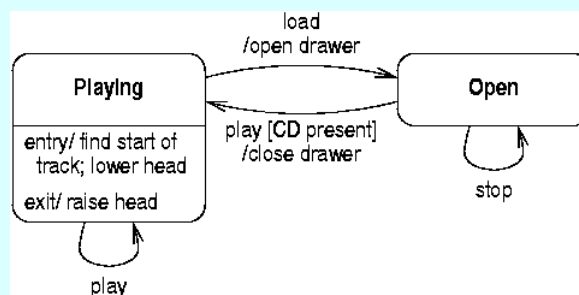
Actions

- *Actions* are performed when a transition fires



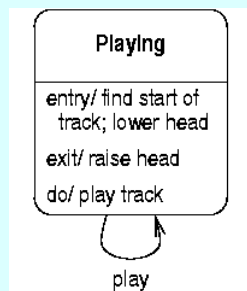
Entry and exit actions

- Entry and exit actions are properties of states
 - they are performed whenever an object arrives at or leaves the state, respectively



Activities

- Activities are also properties of states
 - they are performed while the object is in a state
 - unlike actions, they can be interrupted by new events



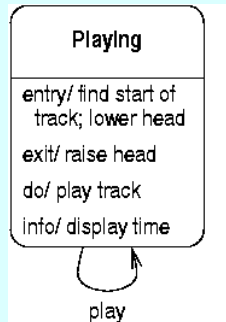
Completion Transitions

- If an activity completes uninterrupted it can trigger a *completion transaction*
 - these are transitions without event labels



Internal Transitions

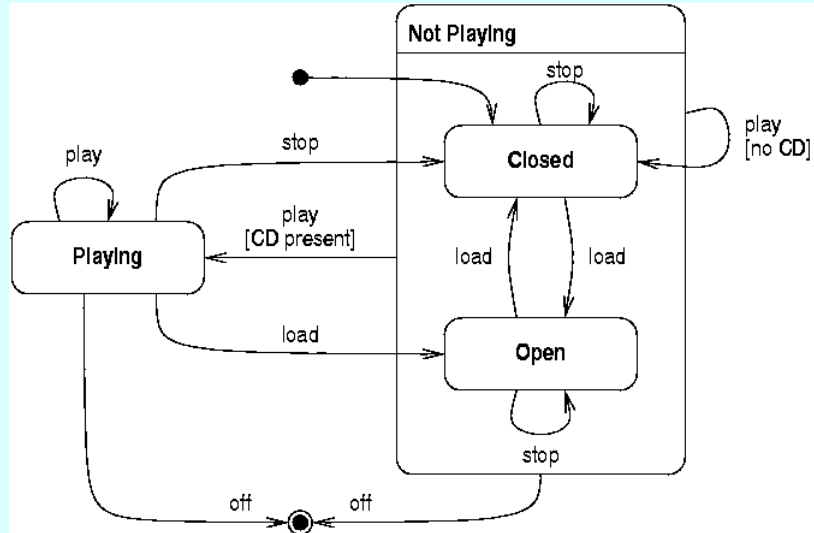
- Internal transitions do not change state
 - and so do not execute entry and exit actions



Composite States

- Composite states group together states
 - this can simplify a statechart by grouping together states with similar behaviour
- An object still has one 'bottom-level' active state
 - but may be in a composite state as well

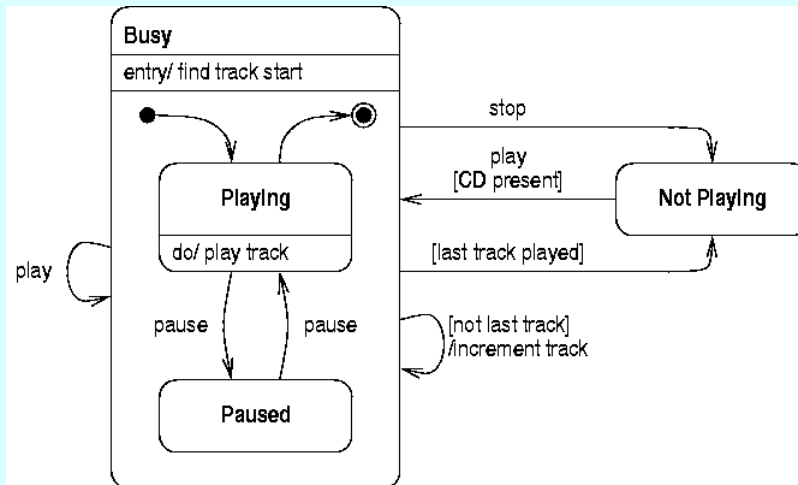
A Composite State



Properties of Composite States

- Transitions
 - from a composite state apply to all substates
 - to a composite state go to the state indicated by a nested initial state
 - can cross composite state boundaries
- Composite states can have activities and entry and exit actions
- A final state in a composite triggers a completion transition from the composite

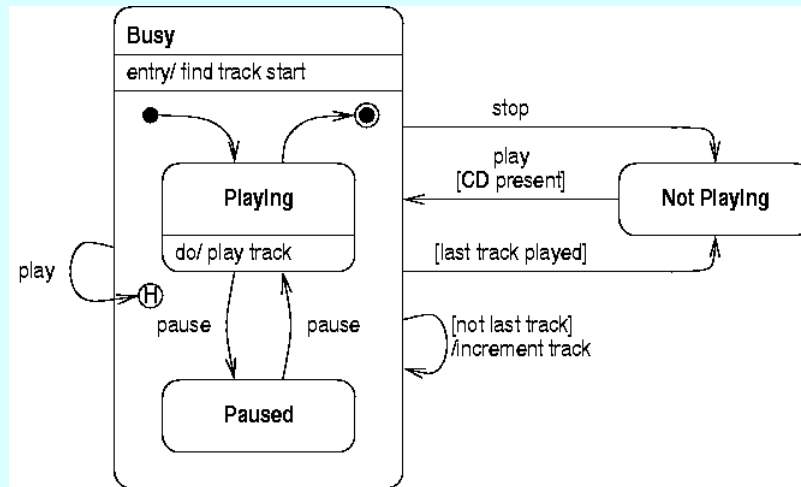
A Complex Composite State



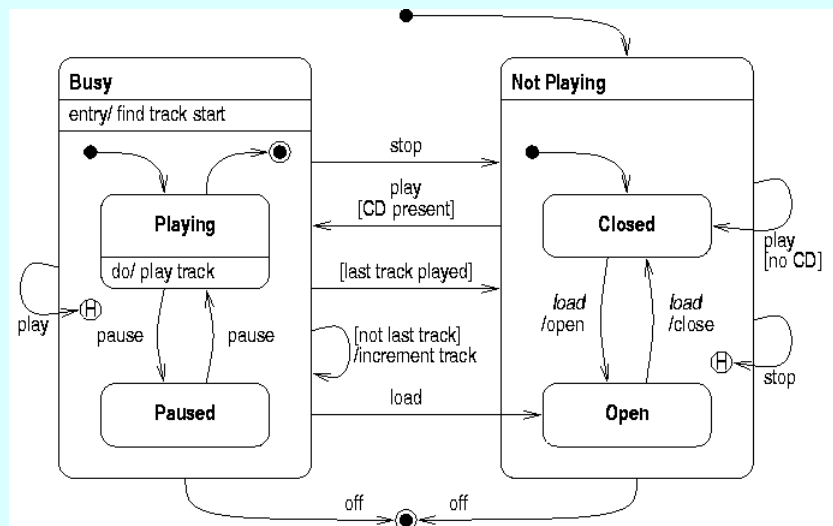
History States

- Often an object needs to 'return to the previous state'
- This can be done by defining conditions
 - such as '[if the last state was Playing]'
- Composite states can have a *history* state
 - this 'remembers' the last active substate
 - a transition to a history state makes that substate active again

Use of a History State



Complete CD Player Statechart



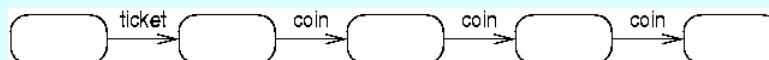
Creating a Statechart

- It can be hard to identify all necessary states
- Statecharts can be developed incrementally
 - consider individual sequences of events received by an object
 - these might be specified on interaction diagrams
 - start with a statechart for one interaction
 - add states as required by additional interactions



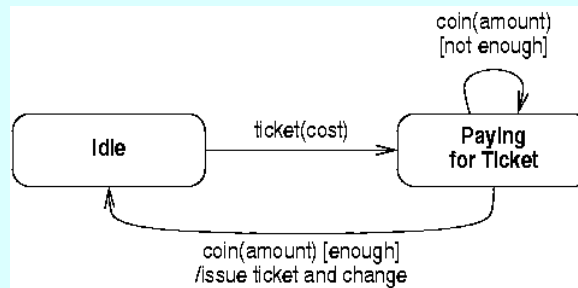
Ticket Machine

- Consider a ticket machine with two events
 - select a ticket type
 - enter a coin
- Basic interaction is to select a ticket and then enter coins
 - model this as a ‘linear’ statechart



Refining the Statechart

- This can be improved by adding 'loops'
 - the number of coins entered will vary: entry will continue until the ticket is paid for
 - the whole transaction can be repeated

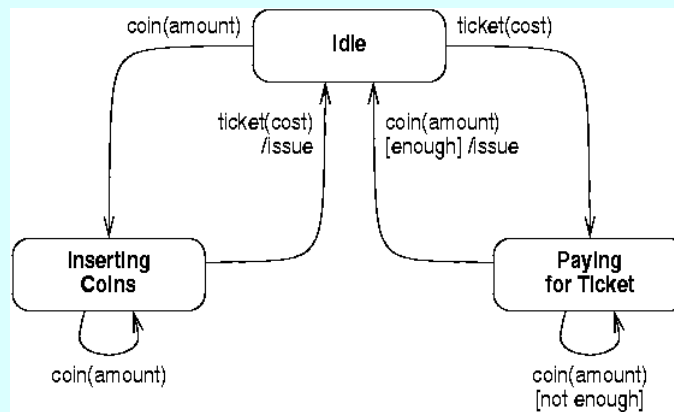


Adding Another Interaction

- The user could enter a coin before selecting a ticket
- A 'coin' transition from the 'Idle' state is needed to handle this event
 - this transition can't go to the 'Paying for Ticket' state as the ticket is not yet selected
 - so a new state 'Inserting Coins' is required
- The statechart is thus built up step-by-step

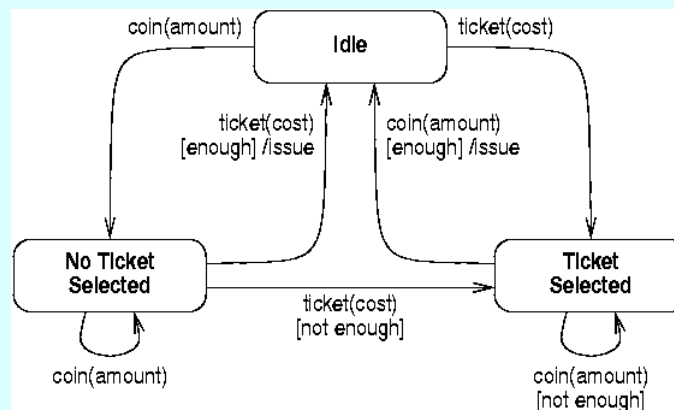
Adding a Second Interaction

- If all coins are entered before ticket selected:



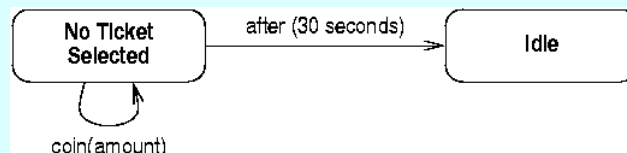
Integrating the Interactions

- In fact, events can occur in any sequence:



Time Events

- Suppose the ticket machine times out after 30 seconds
 - we need to fire a transition that is not triggered by a user-generated event
 - UML define *time events* to handle these cases

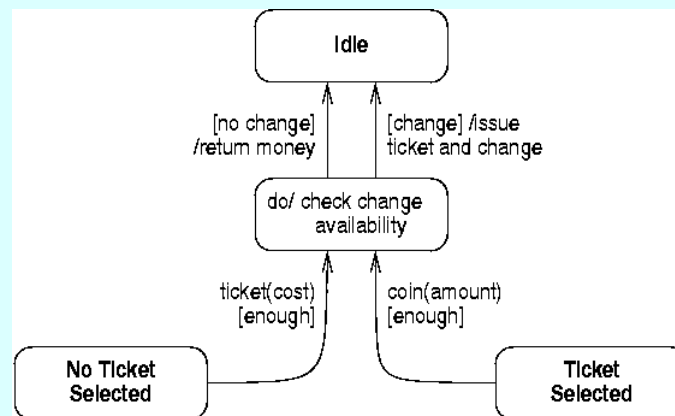


Activity States

- *Activity states* defines periods of time when the object is carrying out internal processing
 - unlike normal activities, these cannot be interrupted by external events
 - only *completion* transitions leading from them
 - useful for simplifying the structure of complex statecharts

Returning Change

- Use an activity state to calculate change



Ticket Machine Statechart

