


**OBJECT-ORIENTED
MODELLING (OOM) - 32536**

MODULE 8

**State Machine Diagrams (SMD),
Part of Dynamic Modelling**
(POOD – Chapter 5; POOD – Chapter 7)

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Module Outline



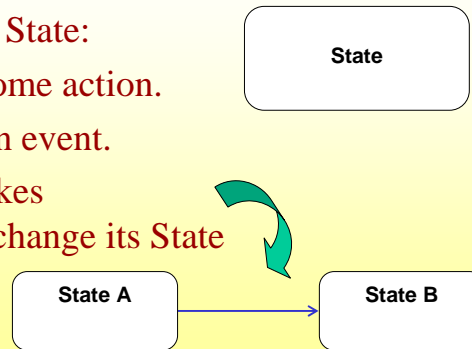
- Purpose of State Machine Diagrams
- Major Ingredients of State Machine Diagrams
- Nested State Diagrams
- How to Draw a State Machine Diagram

©(c) Thomson Publishing, 2005: Practical OO Design,
1998-2005; v 3.1

2

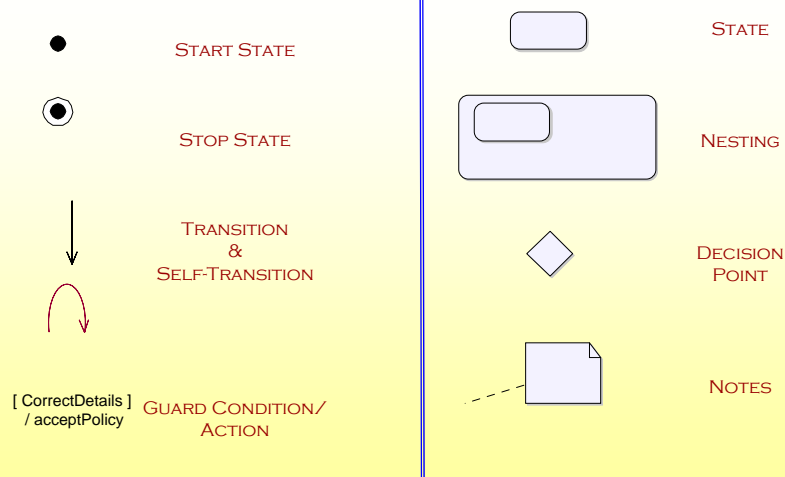
States, Transitions

- States Describe the Values of Attributes of an Object
- An object in a State:
 - Performs some action.
 - Waits for an event.
- Transition makes an object change its State



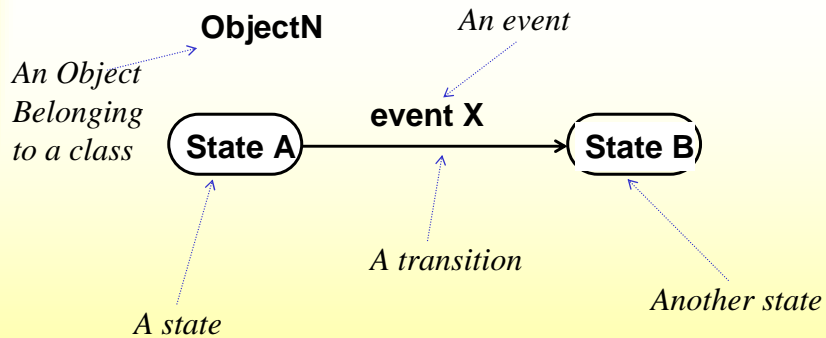
"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Ingredients of a State Machine Diagram



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

State Machine Diagram Symbols



This state diagram tells us that when ObjectN receives event X, ObjectN changes from State A to State B.

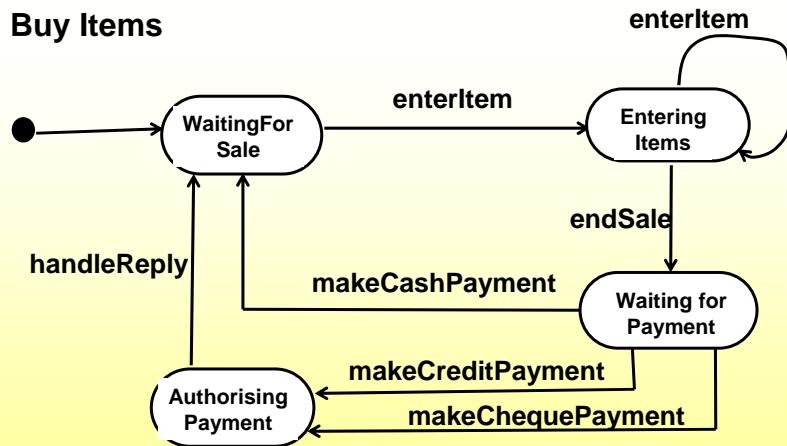
"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Initial and Final States

- **Start State** ●
 - The state of the object at the start of its lifetime.
 - A statechart must have exactly one start state.
- **Stop State** ⊙
 - The state of the object at the end of its lifetime.
 - Optional – a statechart will not have a stop state if the object is never destroyed.
 - A statechart can have many stop states.

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

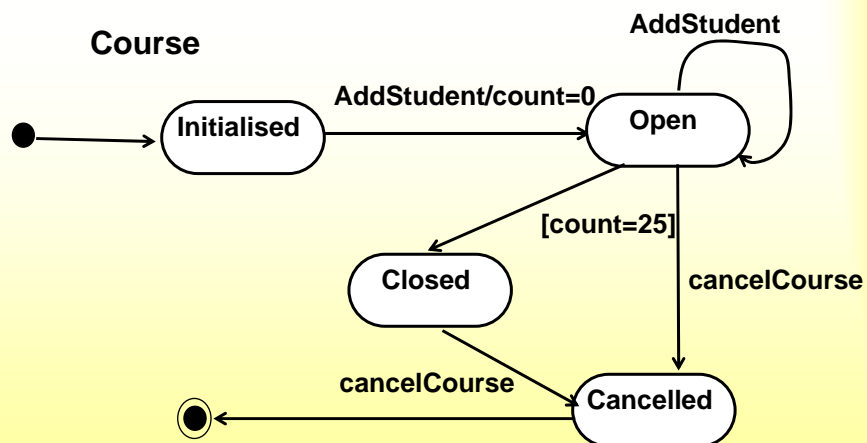
Example: Buy Items state diagram



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Source: Larman(2004), page 491

Another State Diagram example



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Utility of State Diagrams

- State diagrams illustrates the interesting events and states of an object, the behaviour of an object in reaction to an event
- A state diagram shows the life cycle of an object such as what events it experiences, its transitions, and the states it is in between these events
- It need not illustrate every possible event
- If an event arises that is not represented in the diagram, the event is ignored as far as the state diagram is concerned
- Given a set of use case state diagrams, a designer can methodically develop a design that ensures correct system event order. Possible design solutions include
 - hard-core conditional tests for out of order events
 - use of state pattern
 - a state machine interpreter that runs a state table representing use case state diagram

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Types that Need State Diagrams

- If an object always responds the same way to an event, then it is considered **state-independent** with respect to that event
- For example, if an object receives a message, and the responding method always does the same thing -the method will have probably no conditional logic. The object is state-independent with respect to that message
- If for all events of interest a type always reacts the same way it is a state-independent type
- **State-dependent** types react differently to events depending on their state
- In general, business information systems have a minority of interesting state-dependent types
- Process control and telecommunication domains often have many state-dependent objects

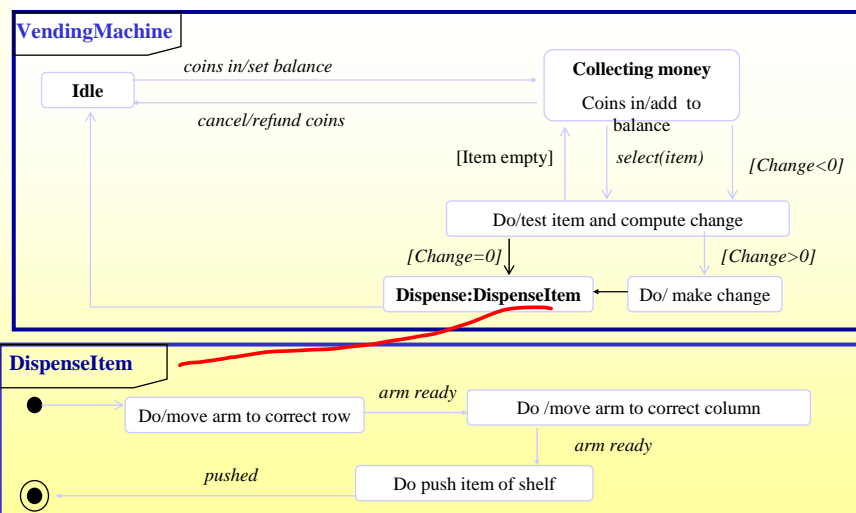
"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Nested State Diagrams

- Problem with flat state diagrams:
 - Consider an object with n independent Boolean attributes that affect control
 - Representing such an object with a single flat state diagram would require 2^n states
 - By partitioning the state into n independent state diagrams, only $2n$ states are required
- Expanding States
 - One way to organise a model is by having high-level diagram with subdiagrams expanding certain states
 - A lower-level state diagram elaborating a state is called a submachine.
 - A submachine is a state diagram that may be invoked as part of another state diagram
 - The UML notation for invoking a submachine is to list a local state name followed by a colon and the submachine name
 - Conceptually, the submachine state diagram replaces the local state
 - Effectively, a submachine is a state diagram “subroutine”.

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

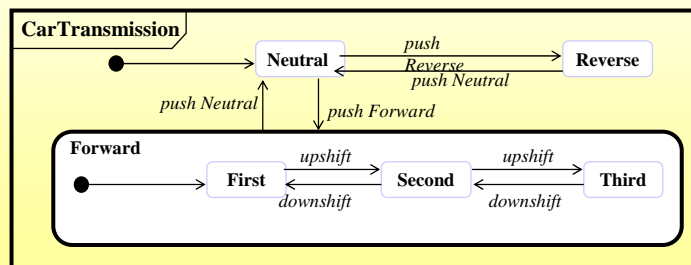
UML Example: Submachine



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

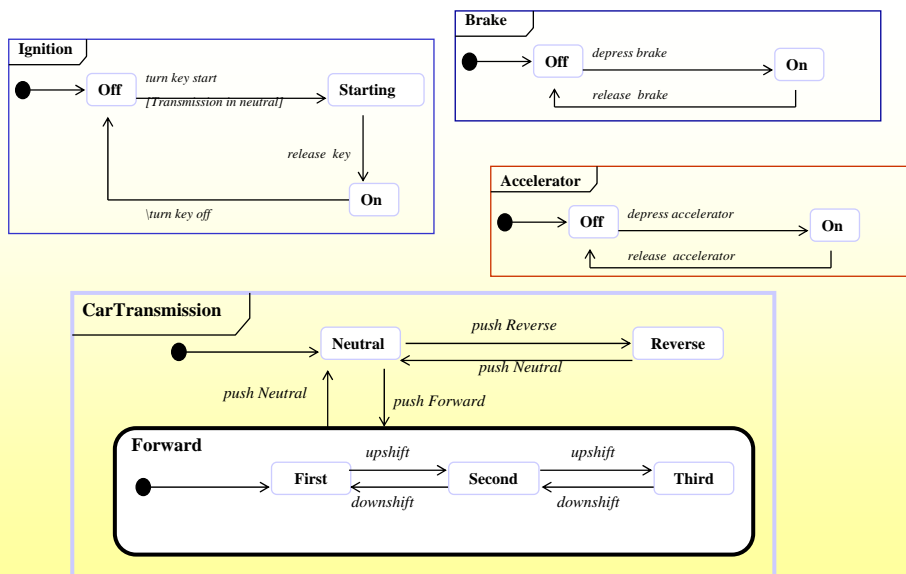
Nested States

- We can also structure states more deeply than just replacing a state with a submachine
- As a deeper alternative, we may nest states to show their commonality and share behaviour
- The **composite state name** labels the outer contour that entirely encloses the nested states
- We may nest states to an arbitrary depth
- A nested state receives the outgoing transitions of its composite state



"Copyright Practical OO Analysis, Thomson Publishing, 2005".

Example: Aggregation Concurrency



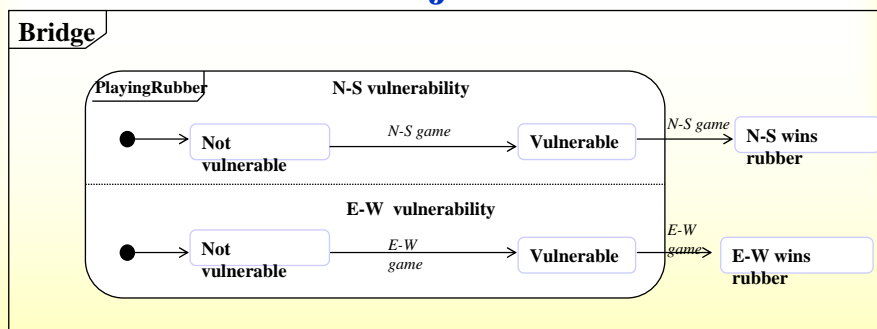
"Copyright Practical OO Analysis, Thomson Publishing, 2005".

Concurrency within an Object

- We can partition some objects onto subsets of attributes or links, each of which has its own subdiagram
- The state of the object comprises one state from each subdiagram
- The subdiagram need not be independent; the same event can cause transitions in more than one subdiagram
- The UML shows concurrency within an object by partitioning the composite state into regions with dotted lines
- Most programming languages lack intrinsic support for concurrency
- During analysis you should regard all objects as concurrent
- During design you devise the best accommodation; many implementations do not require concurrency, and a single thread of control suffices.

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Example: Concurrency within an Object



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Finding States

- List the possible states for each class
- Characterise the objects in each class
 - The **attribute values** that an object may have
 - The **associations** that it may participate in and their multiplicities
 - Attributes and associations that are meaningful only in certain states
- Give each identified state a meaningful name which could directly describe the state
- Don't focus on fine distinctions among states, particularly quantitative differences, such as small, medium, or large
- States should be based on qualitative differences in **behaviour, attributes, or associations**
- It is unnecessary to determine all the states before examining events
- By looking at events and considering transitions among states, missing states will become gradually clear.
- Example, some states for an "Account":
 - **Normal** (ready for normal access)
 - **Closed** (closed by the customer but still on file in the bank records), and
 - **Suspended** (access to the account is blocked for some reason)

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

How to Build a State Chart?

- Determine the Sequences and Events that involve this Object
- For each event, determine the effect it has on the state of the object.
- Apply transitions to the states of an Object
- Add guard conditions associated with the event.
- Determine Nested and Historical States
- Only for Complex, Dynamic and/or Important Objects

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Relation of Subclass and Submachine

State Models

- The state model of a class is inherited by its subclass
- The subclass inherit both the states of the ancestor and the transitions
- The subclasses have their own state models
- How do the state diagrams of the superclass and the subclass interact?
- If the superclass state diagram and the subclass state diagrams deal with disjoint sets of attributes, there is no conflict –the subclass has a composite state composed of concurrent state diagrams
- However, if the state diagram of the subclass involves some of the same attributes as the state diagram of the superclass, a conflict may exist
- The state diagram of the subclass must be a refinement of the state diagram of the superclass.
- Any state from the parent state diagram can be elaborated with nesting or split into concurrent parts
- Usually the state diagram of a subclass should be independent, orthogonal, concurrent addition to the state diagram inherited from a superclass

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Common State-Dependent Object/Classes

- Use cases
 - only those that react differently
- Systems
 - A type representing overall application domain or system
- Windows
 - Edit-paste action is only valid if there is something in the clipboard to paste
- Application coordinators
 - Applets in Java
- Controller
 - a non window application class that is responsible for handling system events
- Transactions

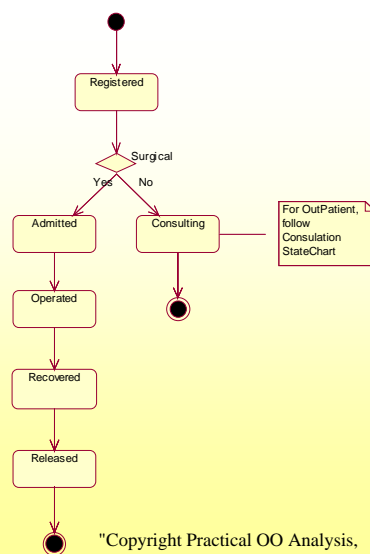
"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Sub-module

State machine diagrams for hospital management system

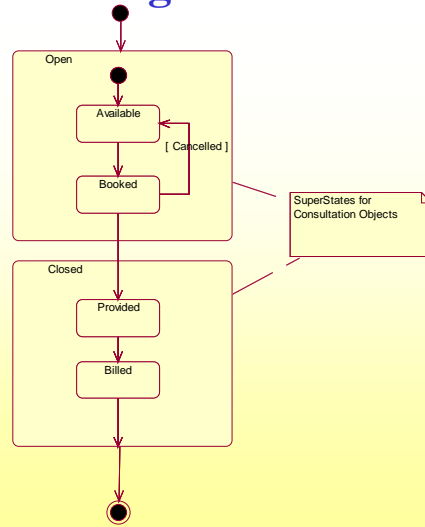
"Copyright Practical OO Analysis, Thomson Publishing, 2005".

“Patient “ State Machine Diagram



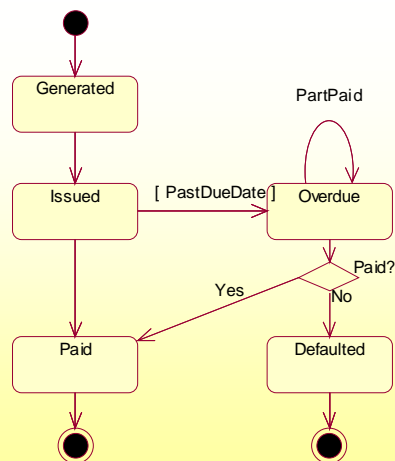
"Copyright Practical OO Analysis, Thomson Publishing, 2005".

“Consultation“ State Machine Diagram



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

“Bill Payment“ State Machine Diagram



"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Conclusions

- Discussed State Machine Diagrams (SMD)
- Presented the Notations and Process related to drawing of SMDs
 - Nesting
 - Concurrency
- Considered Examples of SMDs in Design, through a Hospital Management System

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

WORKBOOK



EXERCISES

For The Class Room!!

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".

Workbook Exercises

1. Identify two crucial Entity objects whose state change is important from both a functional and design viewpoint.
2. Draw the corresponding State Machine Diagrams for these two objects showing the start state, the various transitions, and one or more end states.
3. Comment the diagram with notes for understandability.

"Copyright Practical OO Analysis,
Thomson Publishing, 2005".