Domain Modeling

Chapter 4
Lecture 3

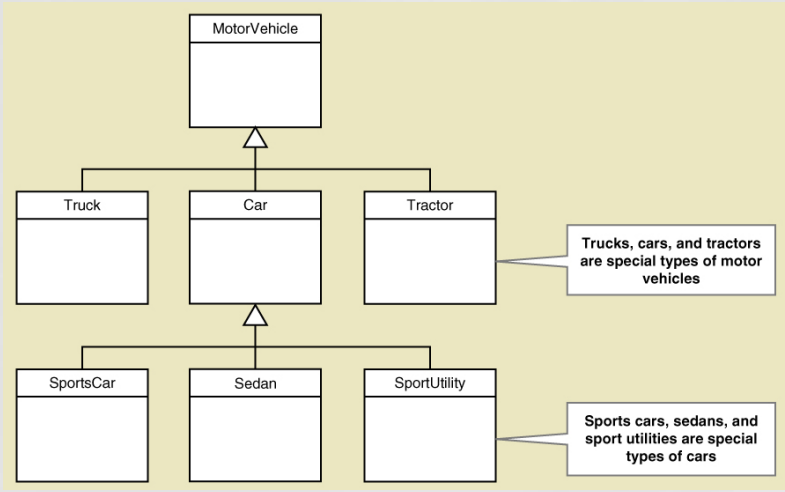
Systems Analysis and Design in a Changing World 7th Ed

Satzinger, Jackson & Burd

More Complex Issues about Classes:
Generalization/Specialization Relationships

- Generalization/Specialization
  - A hierarchical relationship where subordinate classes are special types of the superior classes. Often called an Inheritance Hierarchy
- Superclass
  - the superior or more general class in a generalization/specialization hierarchy
- Subclass
  - the subordinate or more specialized class in a generalization/specialization hierarchy
- Inheritance
  - the concept that subclasses classes inherit characteristics of the more general superclass

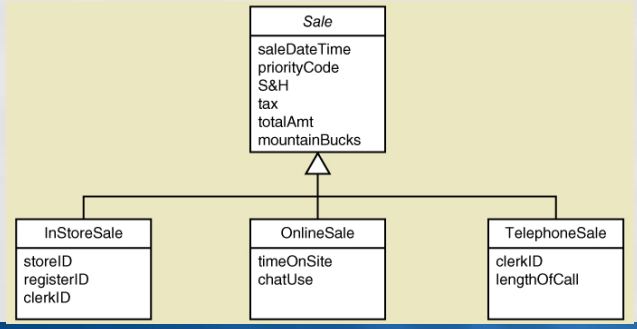
Generalization/Specialization



Generalization/Specialization

Inheritance for RMO Three Types of Sales

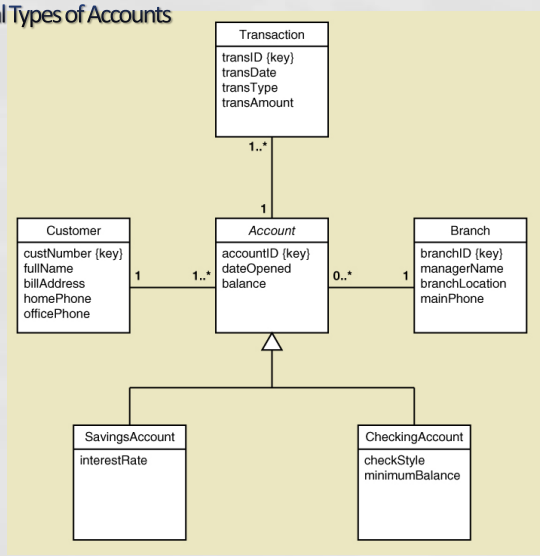
- Abstract class— a class that allow subclasses to inherit characteristics but never gets instantiated. In Italics (*Sale*)
- Concrete class— a class that can have instances
- Inheritance – Attributes of OnlineSale are:
  - timeOnSite, chatUse, saleDateTime, priorityCode, S&H, tax, totalAmt…



Generalization/Specialization

Inheritance for the Bank with Special Types of Accounts

- A SavingsAccount has 4 attributes
- A CheckingAccount has 5 attributes
- Note: the subclasses inherit the associations too



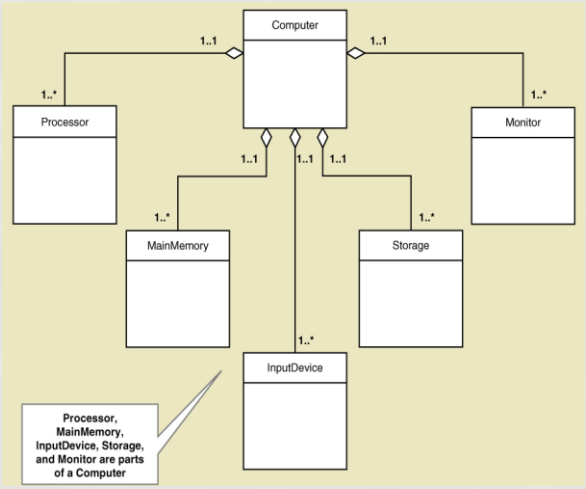
More Complex Issues about Classes:

Whole Part Relationships

- Whole-part relationship— a relationship between classes where one class is part of or a component portion of another class
- Aggregation— a whole part relationship where the component part exists separately and can be removed and replaced (UML diamond symbol on next slide)
  - Computer has disk storage devices (storage devices exist apart from computer)
  - Car has wheels (wheels can be removed and still be wheels)
- Composition— a whole part relationship where the parts cannot be removed (filled in diamond symbol)
  - OrderItem on an Order (without the Order, there are no OrderIterms)
  - Chip has circuits (without the chip, there are no circuits)

Whole Part Relationships
Computer and its Parts

- Note: this is composition, with diamond symbol.
- Whole part can have multiplicity symbols, too (not shown)



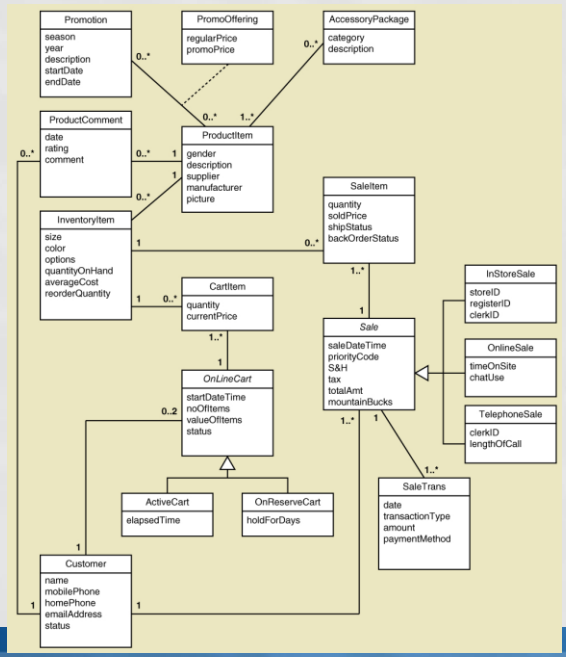
More on UML Relationships

- There are actually three types of ***relationships*** in class diagrams
  - Association Relationships
    - These are associations discussed previously, just like ERD relationships
  - Whole Part Relationships
    - One class is a component or part of another class
  - Generalizations/Specialization Relationships
    - Inheritance
- Try not to confuse relationship with association

RMO CSMS Project
Domain Model Class Diagrams

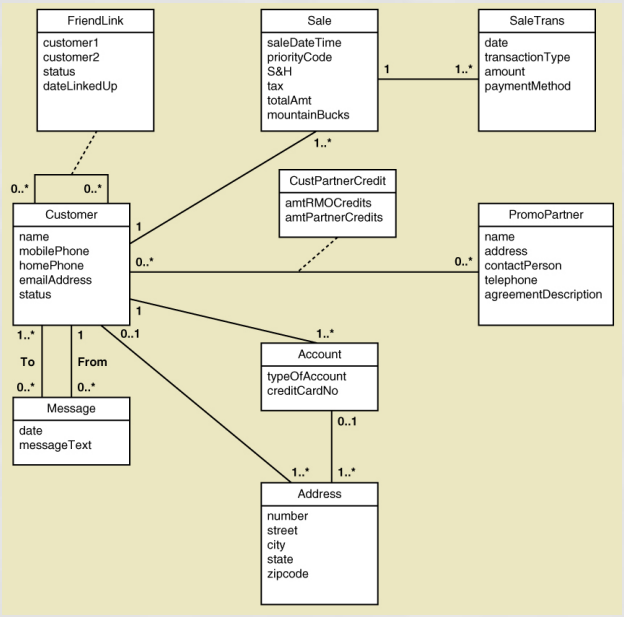
- There are several ways to create the domain model class diagram for a project
- RMO CSMS has 27 domain classes overall
- Can create one domain model class diagram per subsystem for those working on a subsystem
- Can create one overall domain model class diagram to provide an overview of the whole system
- Usually in early iterations, an initial draft of the domain model class diagram is completed and kept up to date. It is used to guide development.

RMO CSMS Project
Sales Subsystem Domain Model Class Diagrams



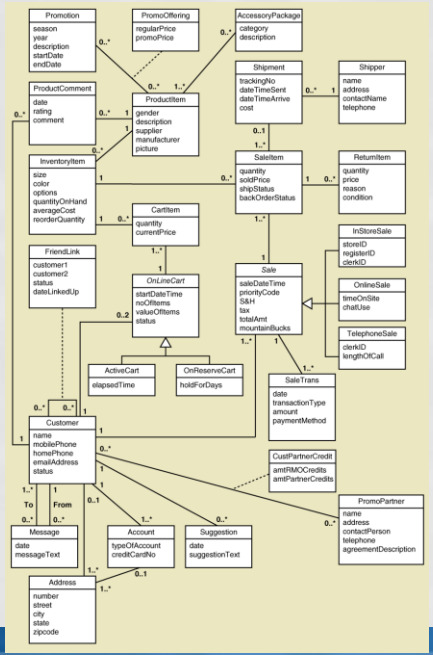
RMO CSMS Project

Customer Account Subsystem Domain Model Class Diagram



RMO CSMS Project

Complete Domain Model Class Diagram



RMO CSMS Project
Domain Model Class Diagrams

- Given the complete RMO CSMS Domain Model Class Diagram and Sales and Customer Account subsystem examples:
  - Try completing the Order Fulfilment Subsystem Domain Model Class Diagram
  - Try Completing the Marketing Subsystem Domain Model Class Diagram
  - Try Completing the Reporting Subsystem Domain Model Class Diagram
- Review the use cases from Chapter 3 and decide what classes and associations from the complete model are required for each subsystem
  - Classes and associations might be duplicated in more than one subsystem model

Domain Modeling

Chapter 4
Lecture 4

Systems Analysis and Design in a Changing World 7th Ed

Satzinger, Jackson & Burd

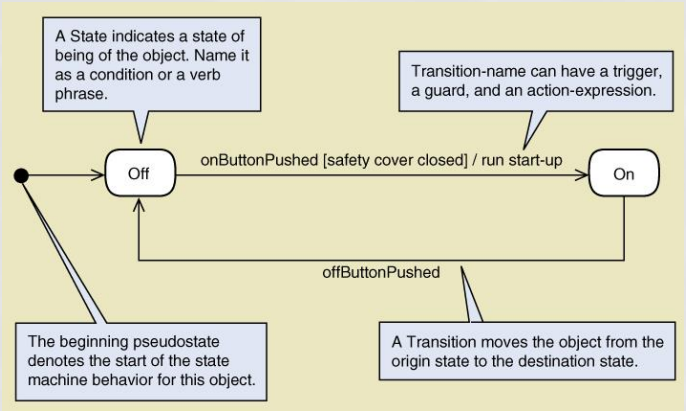
Object Behavior – State Machine Diagram

- Each class has objects that may have status conditions or "states"
- Object behavior consists of the various states and the movement between these states
- **State** – a condition during an object's life when it satisfies some criterion, performs an action, or waits for an event
- **Transition** – the movement of an object from one state to another

State Machine Diagram

- **State Machine Diagram** – a diagram which shows the life of an object in states and transitions
- **Origin state** – the original state of an object before it begins a transition
- **Destination state** – the state to which an object moves after completing a transition
- **pseudostate** – the starting point in a state machine diagram. Noted by a black circle.
- **action-expression** – some activity that must be completed as part of a transition
- **guard-condition** – a true/false test to see whether a transition can fire

State Machine for a Printer

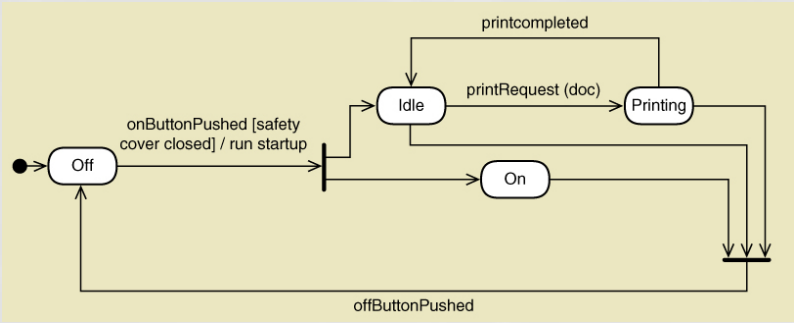


- Syntax of transition statement
  - *transition-name (parameters, …) [guard-condition] / action-expression*

Concurrency in a State Machine Diagram

- **Concurrent states** – when an object is in one or more states at the same time
- **Path** – a sequential set of connected states and transitions
- **Concurrent paths** – when multiple paths are being followed concurrently, i.e. when one or more states in one path are parallel to states in another path

Printer with Concurrent Paths



- Concurrent paths often shown by synchronization bars (same as Activity Diagram)
- Multiple exits from a state is an "OR" condition.
- Multiple exits from a synchronization bar is an "AND" condition.

Creating a State Machine Diagram

Steps

1. Review the class diagram and select classes that might require state machine diagrams
2. For each class, make a list of status conditions (states) you can identify
3. Begin building diagram fragments by identifying transitions that cause an object to leave the identified state
4. Sequence these states in the correct order and aggregate combinations into larger fragments
5. Review paths and look for independent, concurrent paths

Creating a State Machine Diagram
Steps (continued)

6. Look for additional transitions and test both directions
7. Expand each transition with appropriate message event, guard condition, and action expression
8. Review and test the state machine diagram for the class
   - Make sure state are really state for the object in the class
   - Follow the life cycle of an object coming into existence and being deleted
   - Be sure the diagram covers all exception condition
   - Look again for concurrent paths and composite states

RMO – Creating a State Machine Diagram
Steps -- SaleItem

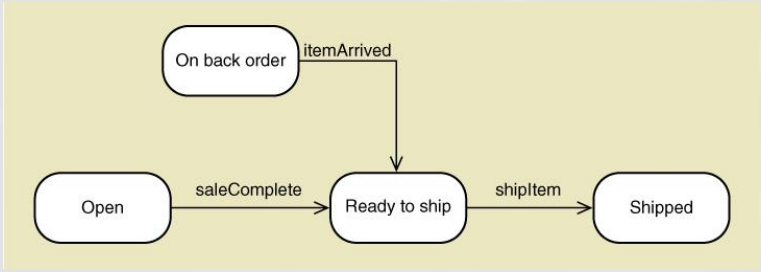
1. Choose SaleItem. It has status conditions that need to be tracked
2. List the states and exit transitions

State	Transition causing exit
Open	saleComplete
Ready to Ship	shipItem
On back order	itemArrived
Shipped	No exit transition defined

RMO – Creating a State Machine Diagram

Steps -- SaleItem

3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – none

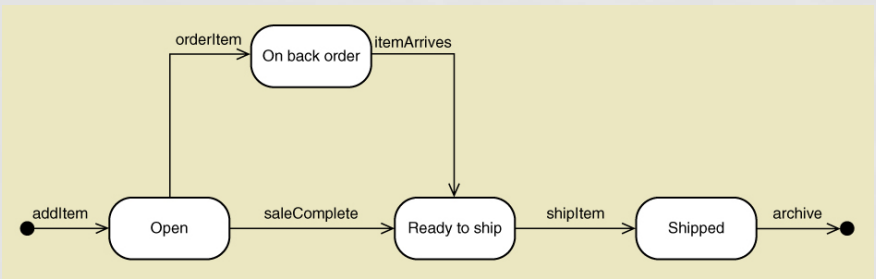


RMO – Creating a State Machine Diagram

Steps -- SaleItem

6. Add other required transitions
7. Expand with guard, action-expressions etc.
8. Review and test

Below is the final State Machine Diagram



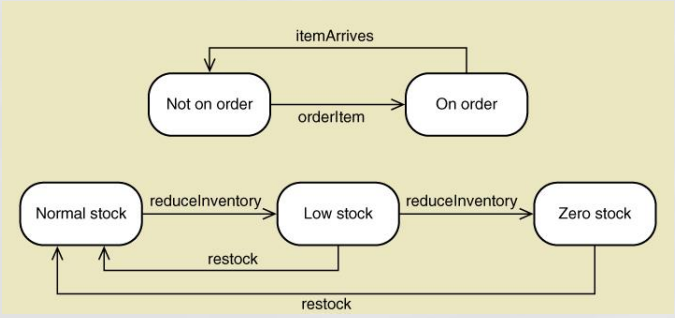
RMO – Creating a State Machine Diagram
Steps -- InventoryItem

1. Choose InventoryItem. It has status conditions that need to be tracked
2. List the states and exit transitions

State	Transition causing exit
Normal stock	reduceInventory
Low stock	reduceInventory OR restock
Zero stock	removeItem OR restock
On order	itemArrives
Not on order	orderItem

RMO – Creating a State Machine Diagram
Steps -- InventoryItem

3. Build fragments – see figure below
4. Sequence in correct order – see figure below
5. Look for concurrent paths – see figure below

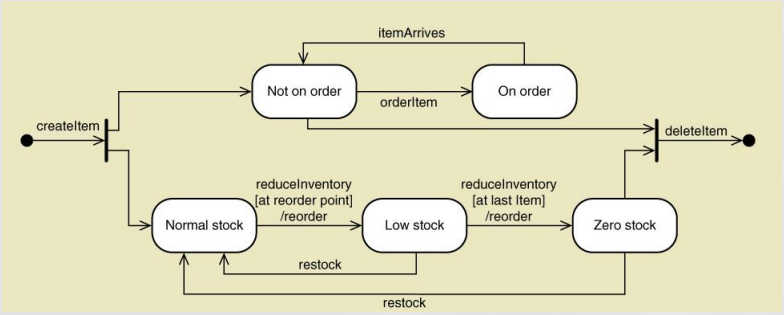


RMO – Creating a State Machine Diagram

Steps -- InventoryItem

6. Add other required transitions
7. Expand with guard, action-expressions etc.
8. Review and test

Below is the final State Machine Diagram



Summary

- This chapter focuses on modeling functional requirements as a part of systems analysis
- "Things" in the problem domain are identified and modeled, called domain classes or data entities
- Two techniques for identifying domain classes/data entities are the brainstorming technique and the noun technique
- Domain classes have attributes and associations
- Associations are naturally occurring relationships among classes, and associations have minimum and maximum multiplicity

Summary

- Entity-relationship diagrams (ERDs) show the information about data entities
- ERDs are often preferred by database analysts and are widely used
- ERDs are not UML diagrams, and an association is called a relationship, multiplicity is called cardinality, and generalization/specialization (inheritance) and whole part relationships are usually not shown

Summary

- The UML class diagram notation is used to create a domain model class diagram for a system. The domain model classes do not have methods because they are not yet software classes.
- There are actually three UML class diagram relationships: association relationships, generalization/specialization (inheritance) relationships, and whole part relationships
- Other class diagram concepts are abstract versus concrete classes, compound attributes, composition and aggregation, association classes, super classes and subclasses

Summary

- Some objects have a life cycle with status conditions that change and should be tracked
- A State Machine Diagram tracks the behavior of these objects with states and transitions
- To develop a State Machine Diagram
  - Choose a single object class.
  - Identify the states and exit transitions
  - Identify concurrent paths
  - Identify additional paths
  - Build the State Machine Diagram