SWE 621:
Software Modeling and Architectural Design

Lecture 2
OO Software Life Cycle
Use Case Modeling

Hassan Gomaa

Dept of Computer Science
George Mason University
Fairfax, VA

Overview

- Collaborative Object Modeling and architectural design mEThod (COMET)
  - Object Oriented Analysis and Design Method
  - Uses UML (Unified Modeling Language) notation
    - Standard approach for describing a software design
  - COMET = UML + Method
- Provides steps and guidelines for
  - Software Modeling and Design
  - From Use Case Models to Software Architecture
- H. Gomaa, *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, Cambridge University Press, February 2011

Model Driven Architecture

- Promoted by Object Management Group (OMG)
- Model Driven Architecture
  - Develop UML models of software architecture before implementation
- Platform Independent Model (PIM)
  - Precise model of software architecture before commitment to specific platform
- Platform Specific Model (PSM)
  - Map PIM UML model to a specific middleware technology
    - CORBA, .NET, J2EE, Web Services
  - Tool support for mapping from PIM to PSM

Unified Modeling Language (UML)

- UML
  - A standardized notation for object-oriented development
  - Combines notations of OMT, Booch, and use cases
  - A graphical language for describing the products of OO requirements, analysis, and design
  - Approved as a standard by Object Management Group (OMG)
  - Methodology independent
- Needs to be used with an analysis and design method

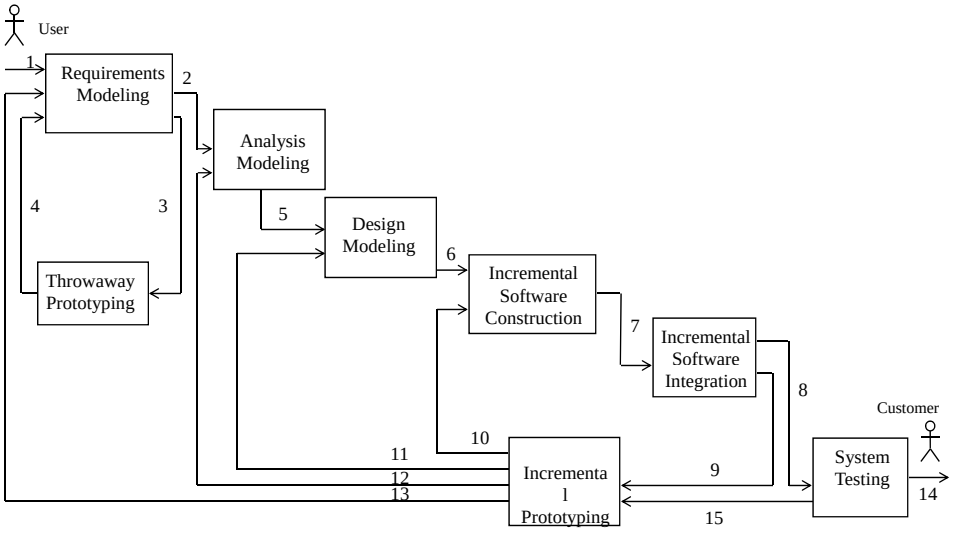
SWE 621:
Lecture 2:
Object-Oriented Software Life Cycle with UML

Hassan Gomaa

Reference: H. Gomaa, "Chapters 5 - "Software Modeling and Design", Cambridge University Press, February 2011

H. Gomaa, "Chapter 6 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

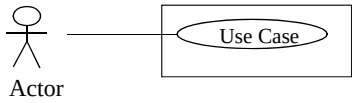
Figure 6.1 COMET object-oriented software life cycle model



Object-Oriented Software Life Cycle

Requirements Modeling

- Requirements Modeling
  - Use Case Modeling
    - – Define software functional requirements in terms of use cases and actors



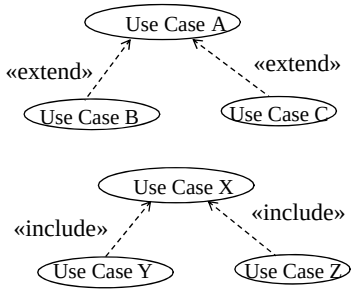


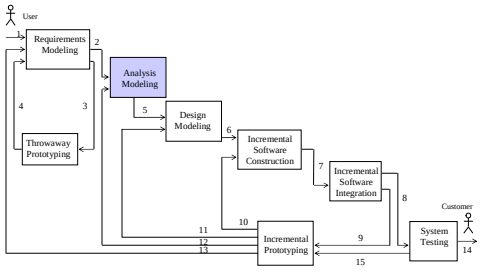
Figure 2.1 UML notation for use case diagram

Object-Oriented Software Life Cycle

Analysis Modeling

- Analysis Modeling consists of
  - Static Modeling
  - Dynamic Modeling
    - State Machine modeling using statecharts
    - Object interaction modeling

Figure 6.1 COMET object-oriented software life cycle model



Object-Oriented Software Life Cycle

Analysis Modeling

- Static Modeling
  - Define structural relationships between classes
  - Depict classes and their relationships on class diagrams

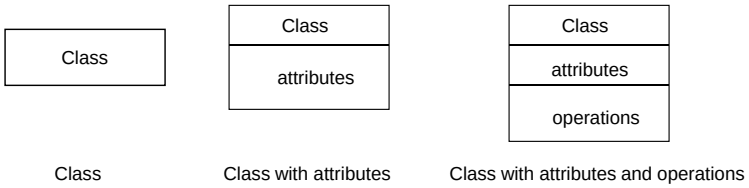


Figure 2.2 UML notation for classes

Object-Oriented Software Life Cycle

Analysis Modeling

- **Dynamic Modeling**
  - Define statecharts for state dependent control objects

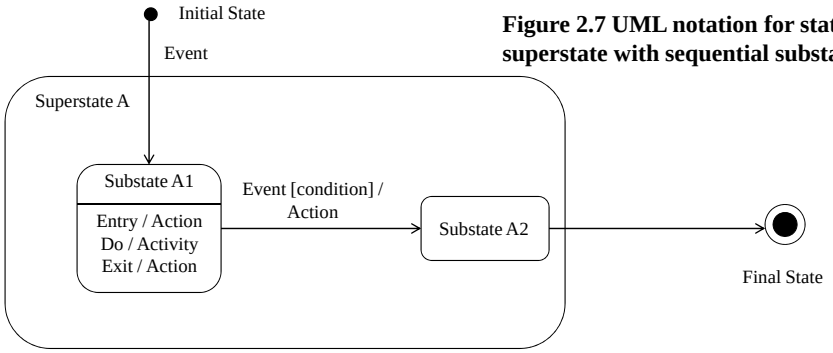


Figure 2.7 UML notation for statechart: superstate with sequential substates

Object-Oriented Software Life Cycle

Analysis Modeling

- **Dynamic Modeling**
  - Defines how objects participate in use cases using communication diagrams or sequence diagrams

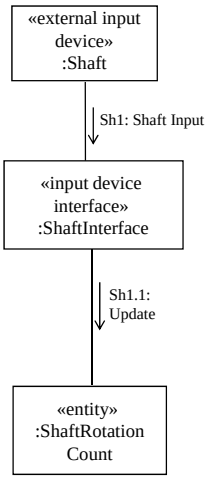


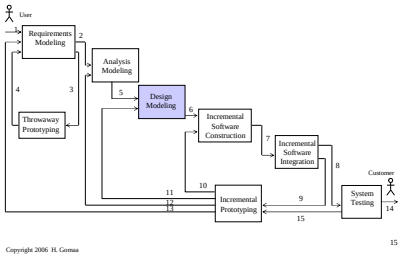
Figure 11.1 Communication diagram for Update Shaft Rotation Count use case

Object-Oriented Software Life Cycle

Design Modeling

- Develop overall software architecture
  - Structure system into subsystems
- Design software architecture
  - Design object-oriented software architectures
  - Design client/server software architectures
  - Design service-oriented architectures
  - Design component-based software architectures.
  - Design concurrent and real-time software architectures
  - Design software product line architectures

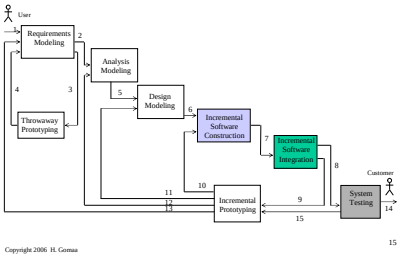
Figure 6.1 COMET object-oriented software life cycle model



Object-Oriented Software Life Cycle

- Incremental Software Construction
  - Select subset of system based on use cases
  - Detailed design, code, unit test of classes in subset
- Incremental Software Integration
  - Integration testing of each system increment
  - Integration test based on use cases
- System Testing
  - Testing of software functional requirements
  - Based on use cases

Figure 6.1 COMET object-oriented software life cycle model



Steps in Using COMET/UML

1 Develop Software Requirements Model
  - Develop Use Case Model (Chapter 6)

2 Develop Software Analysis Model
  - Develop static model of problem domain (Chapter 7)
  - Structure system into objects (Chapter 8)
  - Develop statecharts for state dependent objects (Chapter 10)
  - Develop object interaction diagrams for each use case (Chapter 9, 11)

3 Develop Software Design Model

Lecture 2:
Requirements Modeling

Hassan Gomaa

Reference: H. Gomaa, "Chapters 5, 7 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Objectives of Software Requirements Specification

- Communication vehicle among multiple audiences
  - Customers
  - Users
  - Analysts
  - Designers
- Basis for Software Design
  - Provide precise statement of requirements to designers
- Basis for Software Validation
  - Basis for system acceptance criteria
- Basis for controlling evolution of system
  - Changes to existing requirements
  - Addition of new requirements

Components of Software Requirements Specification

- Functional Requirements
- Behavioral Requirements
- Information Requirements
- External Interface Requirements
- Non-Functional Requirements
- Design Constraints

Components of Software Requirements Specification

- Functional Requirements
  - Inputs to software system
  - Outputs from software system
  - Processing to be performed
- Behavioral Requirements
  - Externally observable states
  - Transitions between states
- Information Requirements
  - Entities (classes), Attributes, Relationships
  - Data Dictionary

Components of Software Requirements Specification

- External Interface Requirements
  - User Interfaces
    - Specify characteristics of user interface
      - E.g., Windows, WWW
    - Can be detailed
      - Specify individual screens
  - Hardware Interfaces
    - Very important for embedded systems
  - Software Interfaces
    - Interfaces to other software systems
- System context model
  - Depict boundary of system

Non-Functional Requirements

- User interface characteristics
- Reliability
- Security
- Availability
- Performance
- Modifiability
- Portability
- Cost

Examples of Design Constraints

- Hardware to be supported
- System configuration
  - Centralized v. Distributed
  - Windows v. Unix
- Existing software to be utilized
- Portability requirements
- Anticipated changes to be accommodated

Attributes of Well-Written Software Requirements Specification

- Correct
  - Each requirement is accurate interpretation of user needs
- Complete
  - Includes every significant requirement
  - Defines system responses to every realizable input
  - No "TBD"s
- Unambiguous
  - Every stated requirement has only one interpretation
- Consistent
  - Individual requirements do not conflict
    - Conflicting Terms
    - Conflicting characteristics
    - Temporal inconsistency

Attributes of Well-Written Software Requirements Specification

- Verifiable
  - Every requirement can be tested to determine that system meets requirement
- Understandable by non-computer specialists
  - Formal vs informal notations (Consistent/unambiguous vs Understandability dilemma)
- Modifiable
  - Need Table of Contents, Index, Cross-references
  - Redundancy
    - Modifiability vs Understandability dilemma

Attributes of Well-Written Software Requirements Specification

- Traceable
  - Backwards:
    - To System Level Requirements
    - To User Needs
  - Forwards:
    - To design component(s) that satisfy requirement
    - To code components that satisfy requirement

Approaches to Developing Software Requirements Specification

- Black Box Requirements Specification
  - System considered as black box
  - Specify
    - External inputs and outputs
    - Externally visible states and transitions
    - Functions that produce outputs
- Methods for Requirements Analysis and Specification
  - Structured Analysis
  - Object-Oriented Analysis
  - Use Case Modeling

Use Case Modeling

Section 3

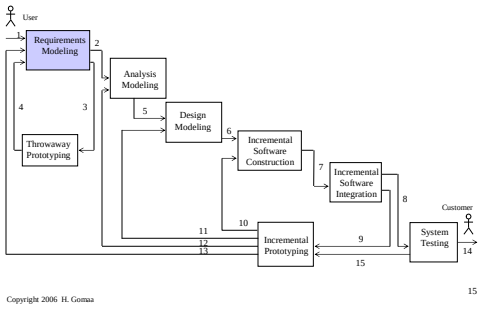
Hassan Gomaa

Reference: H. Gomaa, Chapter 6 - *Software Modeling and Design*, Cambridge University Press, February 2011

Steps in Using COMET/UML

1 **Develop Software Requirements Model**
   - **Develop Use Case Model (Chapter 7)**
2 Develop Software Analysis Model
3 Develop Software Design Model

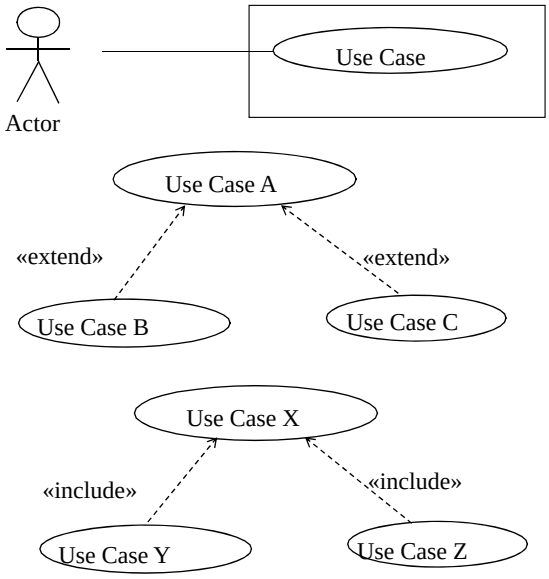
Figure 6.1 COMET object-oriented software life cycle model



Use Case Modeling

- Use Case
  - Describes sequence of interactions between user (actor) and system
  - Narrative description
- Use Case model
  - Define system functional requirements in terms of Actors and Use cases
- Use case relationships
  - include
  - extend

Figure 2.1 UML notation for use case diagram



Actors

- Actor models external entities of system
- Actors interact directly with system
  - Human user
  - External I/O device
  - External system
  - Timer
- Actor initiates actions by system
  - May use I/O devices or external system to physically interact with system
  - Actor initiates use cases

Figure 6.1 Example of actor and use case

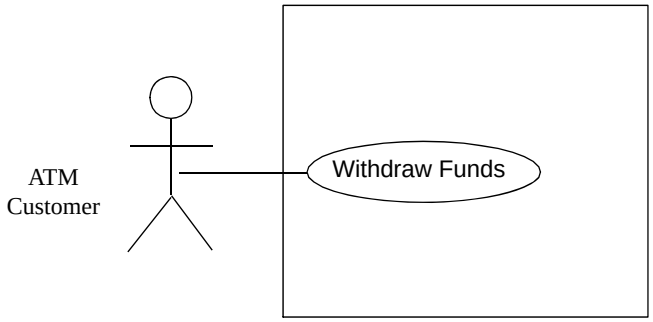
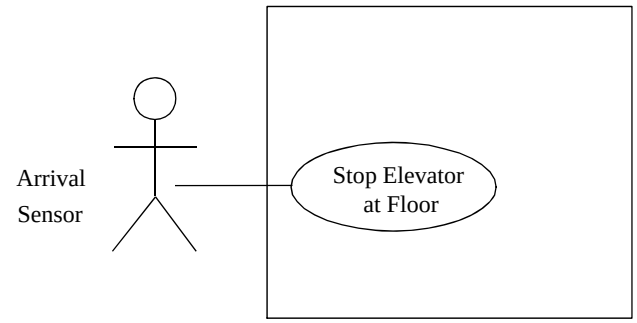


Figure 6.4 Example of input device actor



Actors

- Primary Actor
  - Starts the use case by providing input to the system
- Secondary Actor
  - Participates in use case
  - Can be Primary Actor of a different use case
- Actor
  - Represents all users who use system in the same way
    - A user is an instance of an actor
  - Represents a role played by all users of the same type
    - Human user may play more than one role

Figure 6.3 Example of external system actor

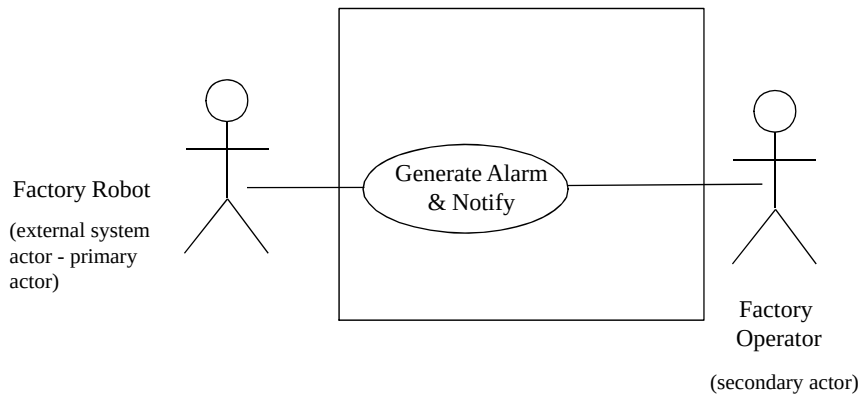
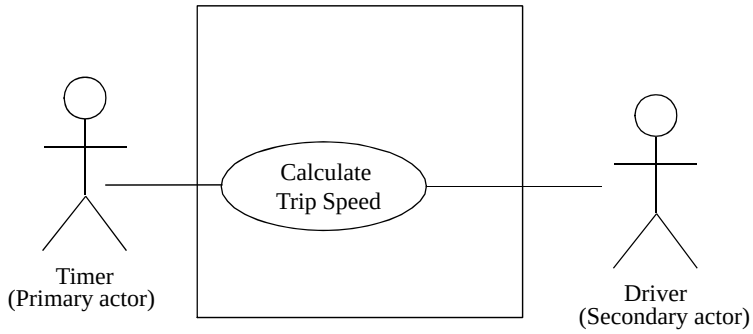


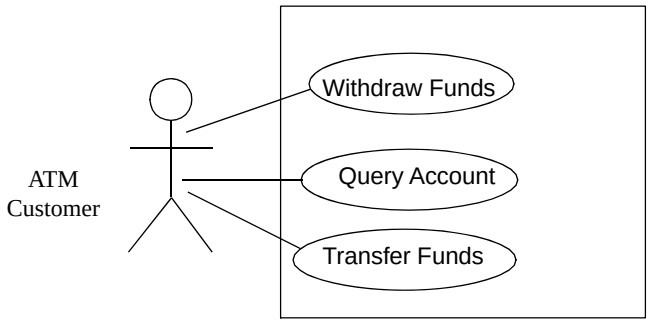
Figure 6.5 Example of timer actor



Use Cases

- Identifying use cases
  - Consider each major function an actor needs to perform
  - Provides value to actor
  - Use case is a complete sequence of events initiated by an actor
    - Specifies interaction between actor and system
  - Use case starts with input from an actor
  - Basic path
    - Most common sequence
  - Alternative branches
    - Variants of basic path
      - E.g., for error handling

Figure 6.7 Banking system actor & use cases



Documenting Use Cases

- Name
- Summary
  - Short description of use case
- Dependency (on other use cases)
- Actors
- Preconditions
  - Conditions that are true at start of use case
- Description
  - Narrative description of basic path
- Alternatives
  - Narrative description of alternative paths
- Postcondition
  - Condition that is true at end of use case

Example of Use Case

Use Case Name: Withdraw Funds

Summary: Customer withdraws a specific amount of funds from a valid bank account.

Actor: ATM Customer

Precondition: ATM is idle, displaying a Welcome message.

Description:

1. Customer inserts the ATM Card into the Card Reader.
2. If the system recognizes the card, it reads the card number.
3. System prompts customer for PIN number.
4. Customer enters PIN.
5. System checks the expiration date and whether the card is lost or stolen.
6. If card is valid, the system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, the system checks what accounts are accessible with the ATM Card.
8. System displays customer accounts and prompts customer for transaction type: Withdrawal, Query, or Transfer.
9. Customer selects Withdrawal, enters the amount, and selects the account number.
10. System checks whether customer has enough funds in the account and whether daily limit has been exceeded.
11. If all checks are successful, system authorizes dispensing of cash.
12. System dispenses the cash amount.
13. System prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.
14. System ejects card.
15. System displays Welcome message.

Example of Use Case (continued)

Alternatives:

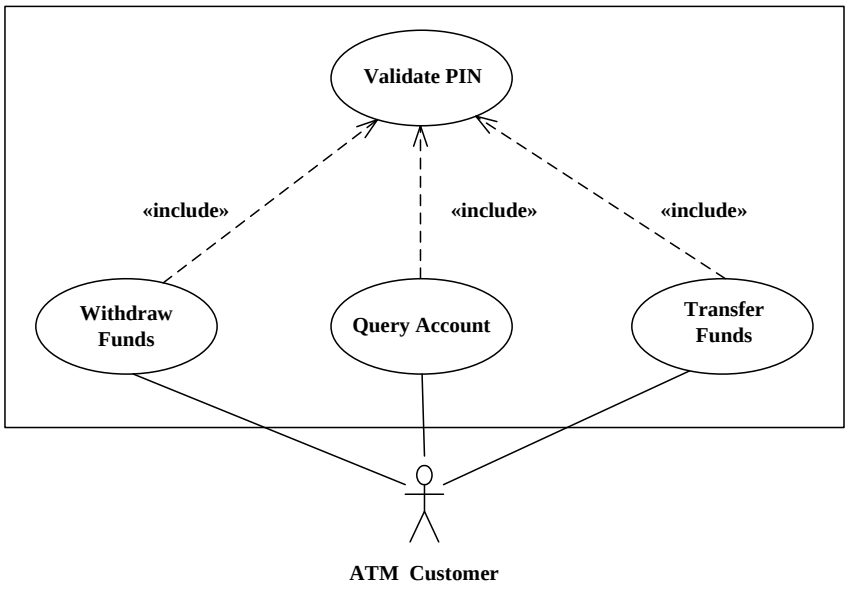
- If the system does not recognize the card, the card is ejected.
- If the system determines that the card date has expired, the card is confiscated.
- If the system determines that the card has been reported lost or stolen, the card is confiscated.
- If the customer entered PIN does not match the PIN number for this card, then the system re-prompts for the PIN.
- If the customer enters the incorrect PIN three times, then the system confiscates the card.
- If the system determines that the account number is invalid, then it displays an error message and ejects the card.
- If the system determines that there are insufficient funds in the customer's account, then it displays an apology and ejects the card.
- If the system determines that the maximum allowable daily withdrawal amount has been exceeded, then it displays an apology and ejects the card.
- If the ATM is out of funds, then the system displays an apology, ejects the card, and shuts down the ATM.
- If the customer enters Cancel, the system cancels the transaction and ejects the card.

Postcondition: Customer funds have been withdrawn.

Use Case Relationships

- **Include** relationship
  - Identify common sequences of interactions in several use cases
    - Extract common sequence into **inclusion use case**
    - Base use cases **includes** abstract use case
- Example
  - Withdraw Funds use case **includes** Validate PIN use case

Figure 6.9 Example of inclusion use case and include relationships



Example of Inclusion Use Case

Use Case Name: Validate PIN

Summary: System validates customer PIN.

Actor: ATM Customer

Precondition: ATM is idle, displaying a Welcome message.

Description:

1. Customer inserts the ATM Card into the Card Reader.
2. If the system recognizes the card, it reads the card number.
3. System prompts customer for PIN number.
4. Customer enters PIN.
5. System checks the expiration date and whether the card is lost or stolen.
6. If card is valid, the system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, the system checks what accounts are accessible with the ATM Card.
8. System displays customer accounts and prompts customer for transaction type: Withdrawal, Query, or Transfer.

Alternatives:

- If the system does not recognize the card, the card is ejected.
- If the system determines that the card date has expired, the card is confiscated.
- If the system determines that the card has been reported lost or stolen, the card is confiscated.
- If the customer-entered PIN does not match the PIN number for this card, the system re-prompts for the PIN.
- If the customer enters the incorrect PIN three times, the system confiscates the card.
- If the customer enters Cancel, the system cancels the transaction and ejects the card.

Postcondition: Customer PIN has been validated.

Example of Base Use Case

Use Case Name: Withdraw Funds

Summary: Customer withdraws a specific amount of funds from a valid bank account.

Actor: ATM Customer

Dependency: Include Validate PIN abstract use case.

Precondition: ATM is idle, displaying a Welcome message.

Description:

1. Include Validate PIN abstract use case.
2. Customer selects Withdrawal, enters the amount, and selects the account number.
3. System checks whether customer has enough funds in the account and whether the daily limit will not be exceeded.
4. If all checks are successful, system authorizes dispensing of cash.
5. System dispenses the cash amount.
6. System prints a receipt showing transaction number, transaction type, amount withdrawn, and account balance.
7. System ejects card.
8. System displays Welcome message.

Alternatives:

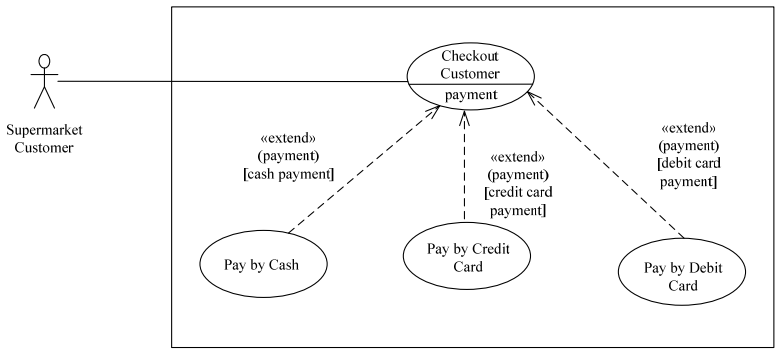
- If the system determines that the account number is invalid, it displays an error message and ejects the card.
- If the system determines that there are insufficient funds in the customer's account, it displays an apology and ejects the card.
- If the system determines that the maximum allowable daily withdrawal amount has been exceeded, it displays an apology and ejects the card.
- If the ATM is out of funds, the system displays an apology, ejects the card, and shuts down the ATM.

Postcondition: Customer funds have been withdrawn.

Use Case Relationships

- **Extend** relationship
  - Use case A is an extension of use case B
  - Under certain conditions use case B will be extended by description given in use case A
  - Same use case can be extended in different ways
- When to use **extend**
  - Show conditional parts of use case
  - Model complex or alternative paths
- Example
  - Pay by Cash **extends** Checkout Customer

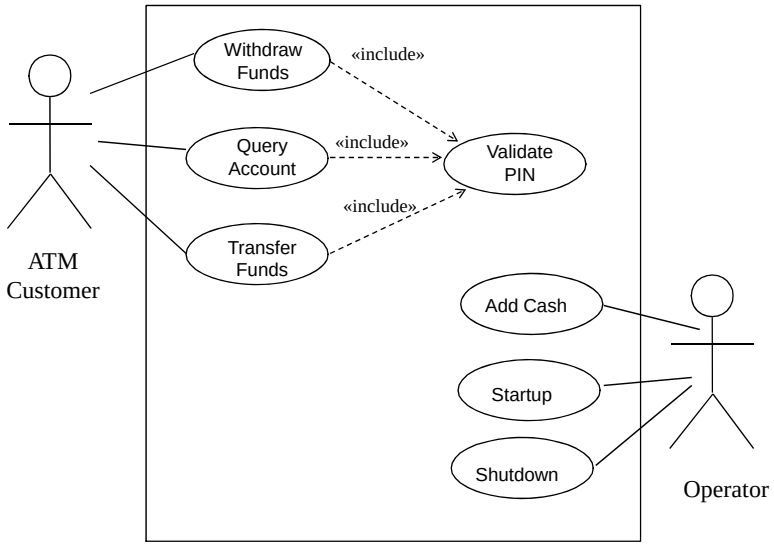
Figure 6.11 Example of extend relationship



Case Study: Banking System

- Multiple Automated Teller Machines (ATM)
  - Customer inserts ATM Card
  - Enters Personal Identification Number (PIN)
  - ATM Transactions
    - PIN Validation
    - Withdraw Funds from Checking or Savings Account
    - Query Account
    - Transfer funds between accounts
- Banking System maintains information about
  - Customers
  - Debit cards
  - Checking and Savings Accounts

Figure 21.1 Banking System use case model



Steps in Using COMET/UML

1 **Develop Software Requirements Model**
   - **Develop Use Case Model (Chapter 7)**
2 Develop Software Analysis Model
3 Develop Software Design Model

Figure 6.1 COMET object-oriented software life cycle model