SOFTWARE ENGINEERING

CHAPTER X-1
UML MODELING

Software Engineering: A Practitioner's Approach, 7th edition
Originated by Roger S. Pressman

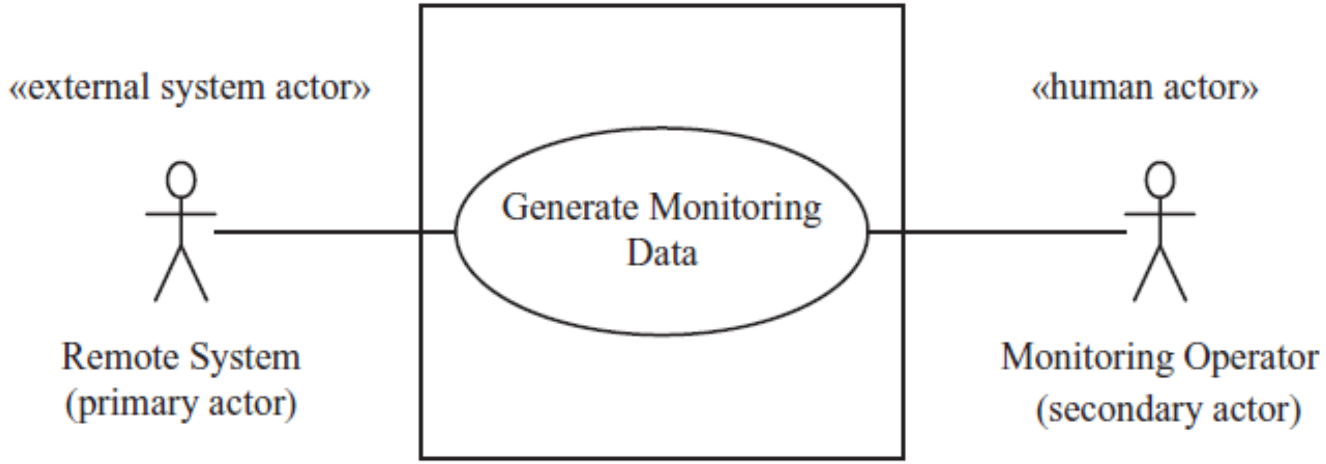
Software School, Fudan University
Spring Semester, 2016

USE CASE MODELING

Software School, Fudan University
Spring Semester, 2016

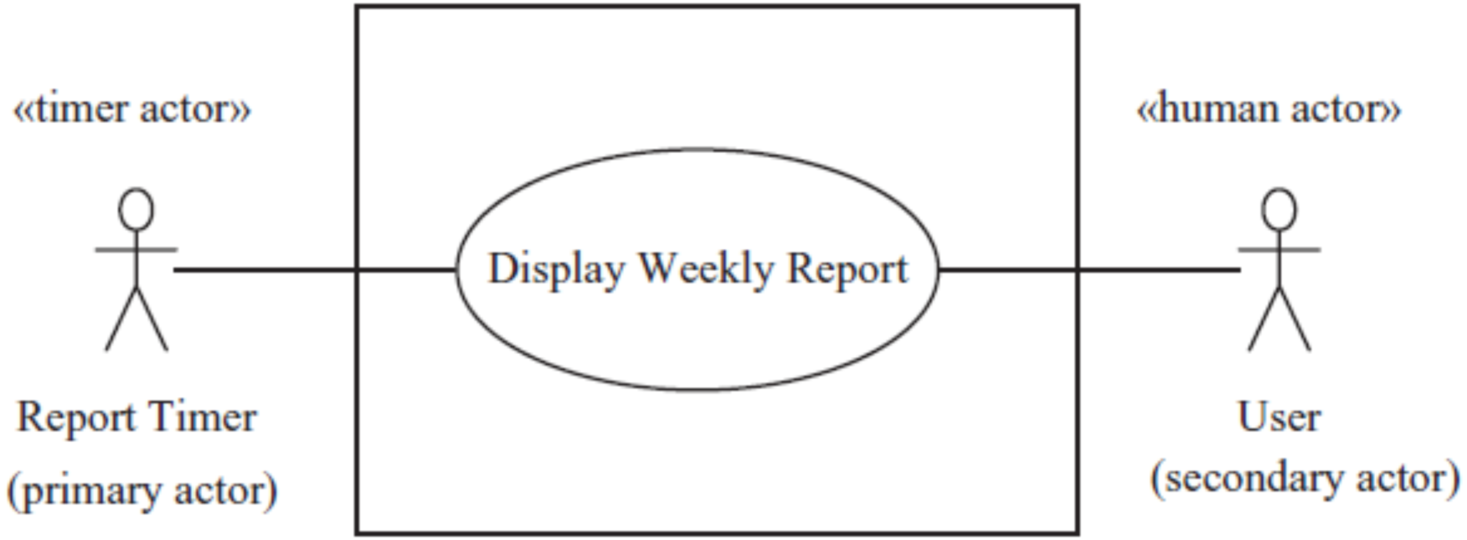
PRIMARY AND SECONDARY ACTORS

- **A primary actor** initiates a use case
- Other actors, referred to as **secondary actors**, can participate in the use case
- A primary actor in one use case can be a secondary actor in another use case



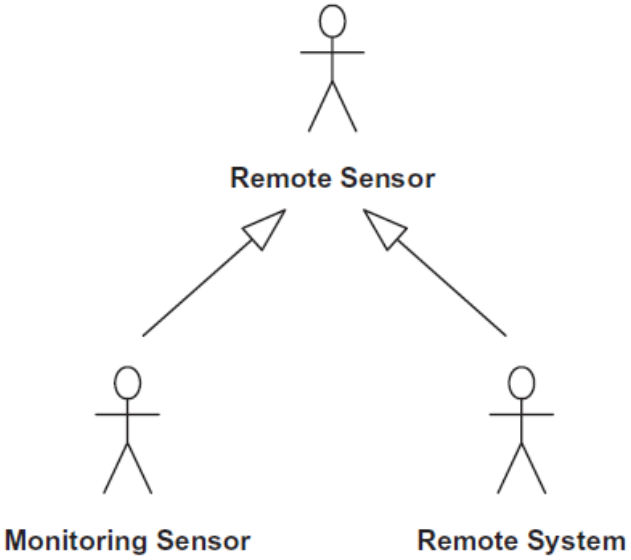
Timer Actor

- A **timer actor** periodically sends timer events to the system



GENERALIZATION AND SPECIALIZATION OF ACTORS

- Different actors might have some roles in common but other roles that are different
- The actors can be generalized: the common part is captured as a generalized actor and the different parts by specialized actors

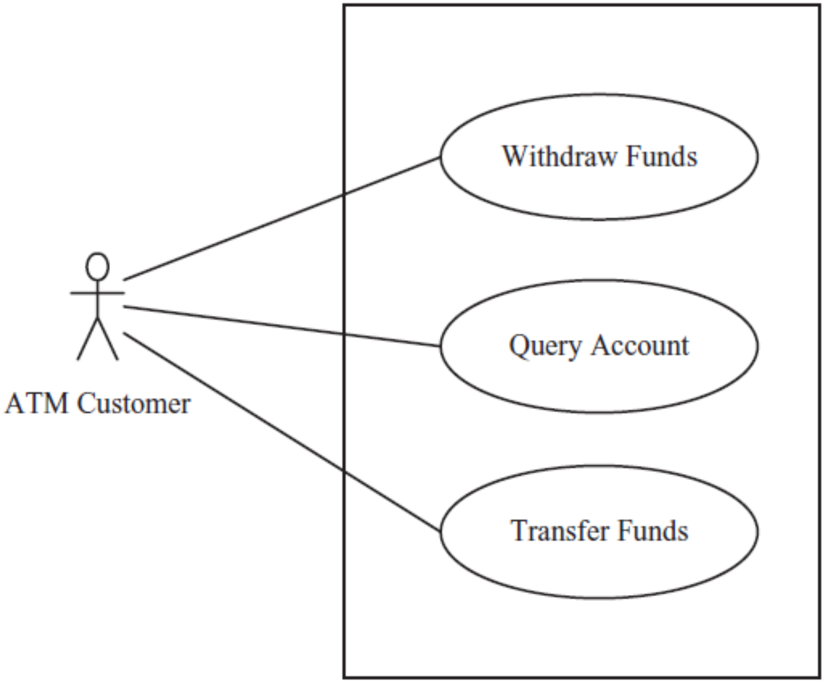


IDENTIFY USE CASE

- Start by considering the actors and the interactions they have with the system
- Avoid a functional decomposition in which
  - several small use cases describe small individual functions of the system
  - rather than describe a sequence of events that provides a useful result to the actor

IDENTIFY USE CASE (EXAMPLE)

- The three use cases are distinct functions initiated by the customer with different useful results

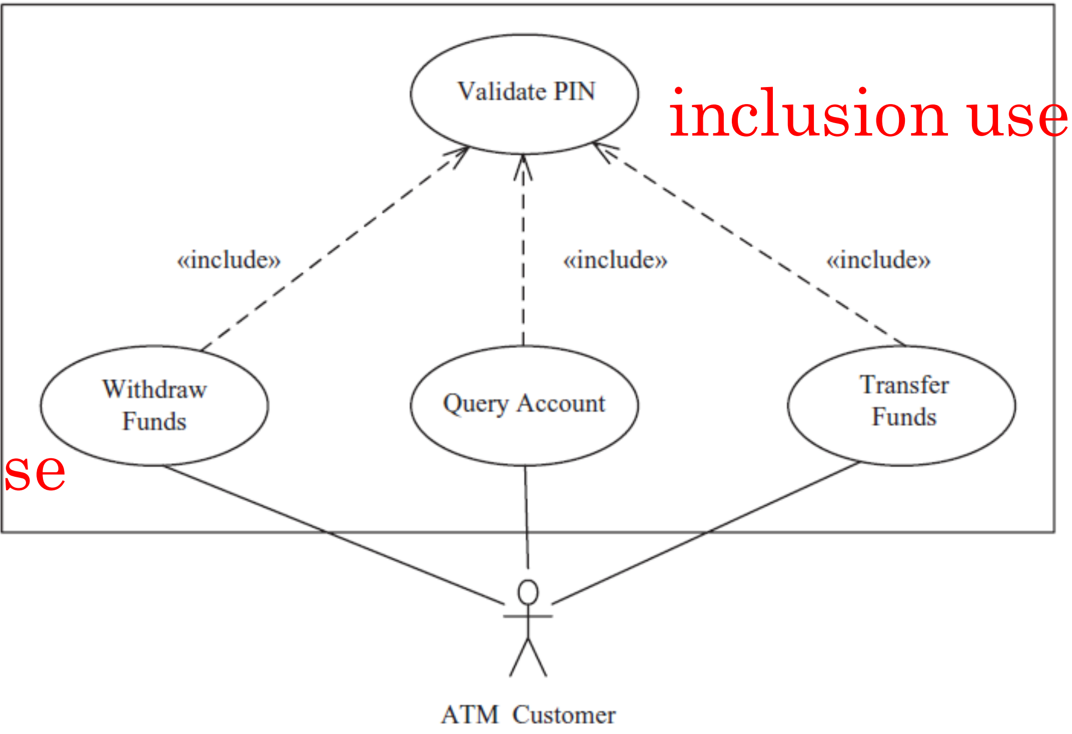


USE CASE RELATIONSHIPS

- **include**: identify common sequences of interactions in several use cases, which can then be extracted and reused
- **extend**: model alternative paths that a use case might take
  - base use case itself can be executed (does not depend on extension use case)
  - extension use case depends on base use case
- **generalization**: sub use case inherits the interaction sequences and behaviors of the base use case

Use Case Inclusion

- An inclusion use case is usually abstract: it must be executed as part of a concrete use case
- An inclusion use case can be reused by several base (executable) use cases



inclusion use case

base use case

USE CASE INCLUSION (EXAMPLE)

Use case name: Validate PIN
Summary: System validates customer PIN.
Actor: ATM Customer
Precondition: ATM is idle, displaying a "Welcome" message.
Main sequence:

1. Customer inserts the ATM card into the card reader.
2. If system recognizes the card, it reads the card number.
3. System prompts customer for PIN.
4. Customer enters PIN.
5. System checks the card's expiration date and whether the card has been reported as lost or stolen.
6. If card is valid, system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, 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.

Alternative sequences: …

inclusion use case

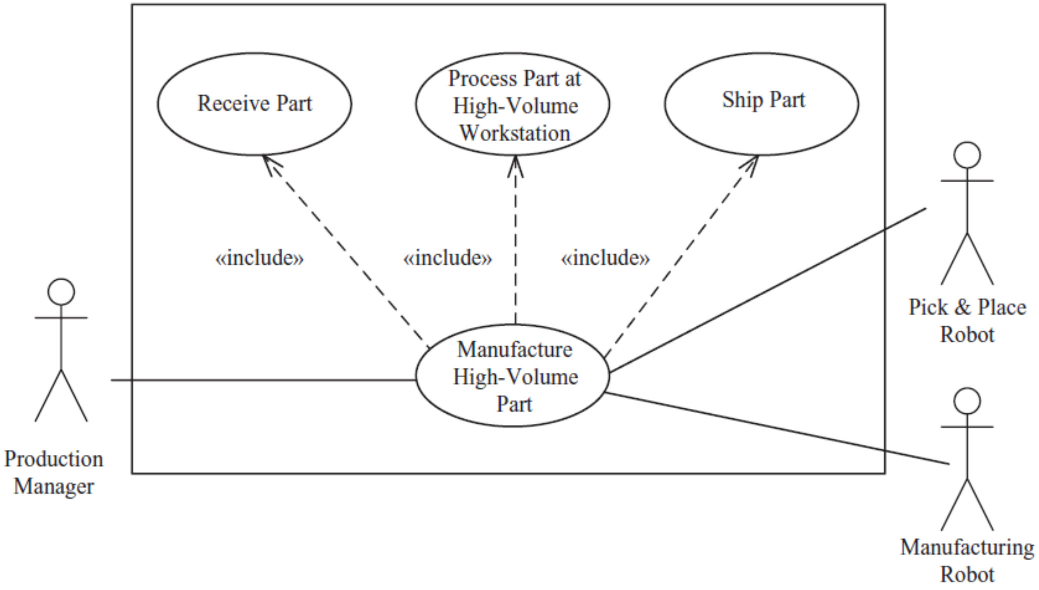
Use case name: Withdraw Funds
Summary: Customer withdraws a specific amount of funds from a valid bank account.
Actor: ATM Customer
Dependency: Include Validate PIN use case.
Precondition: ATM is idle, displaying a "Welcome" message.
Main sequence:

1. Include Validate PIN use case.
2. Customer selects **Withdrawal.**
3. …

base use case

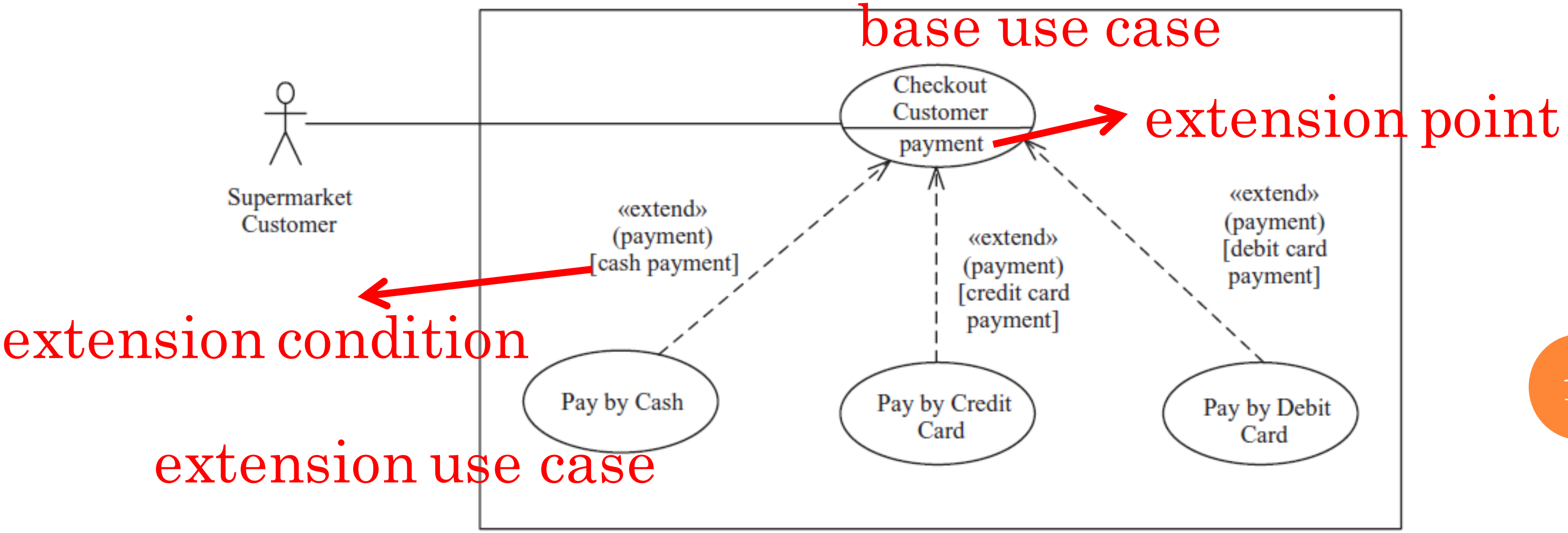
Structuring a Lengthy Use Case

- **include** can also be used to structure a lengthy use case
  - The base use case provides the high-level sequence of interactions between actor(s) and system
  - Inclusion use cases provide lower-level sequences of interactions between actor(s) and system



Use Case Extension

- The **extend** relationship can be conditional: a condition is defined that must be true for the extension use case to be invoked
- **Extension points** are used to specify the precise locations in the base use case at which extensions can be added



USE CASE EXTENSION (EXAMPLE)

Use case name: Checkout Customer

Summary: System checks out customer.

Actor: Customer

Precondition: Checkout station is idle, displaying a "Welcome" message.

Main sequence:

1. Customer scans selected item.
2. System displays the item name, price, and cumulative total.
3. Customer repeats steps 1 and 2 for each item being purchased.
4. Customer selects payment.
5. System prompts for payment by cash, credit card, or debit card.
6. **«payment»**
7. System displays thank-you screen.

base use case

Use case name: Pay by Cash

Summary: Customer pays by cash for items purchased.

Actor: Customer

Dependency: Extends Checkout Customer.

Precondition: Customer has scanned items but not yet paid for them.

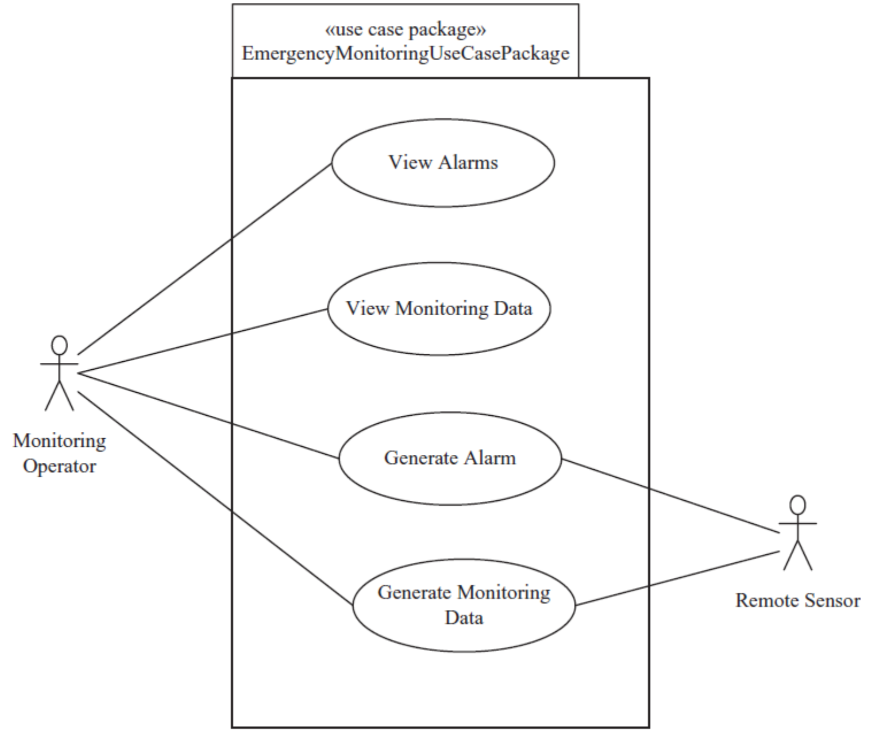
Description of insertion segment:

1. Customer selects payment by cash.
2. System prompts customer to deposit cash in bills and/or coins.
3. Customer enters cash amount.
4. System computes change.
5. System displays total amount due, cash payment, and change.
6. System prints total amount due, cash payment, and change on receipt.

extension use case

USE CASE PACKAGE

- use case package
  - groups together related use cases
  - represent high-level requirements that address major subsets of the functionality of the system



CLASS MODELING

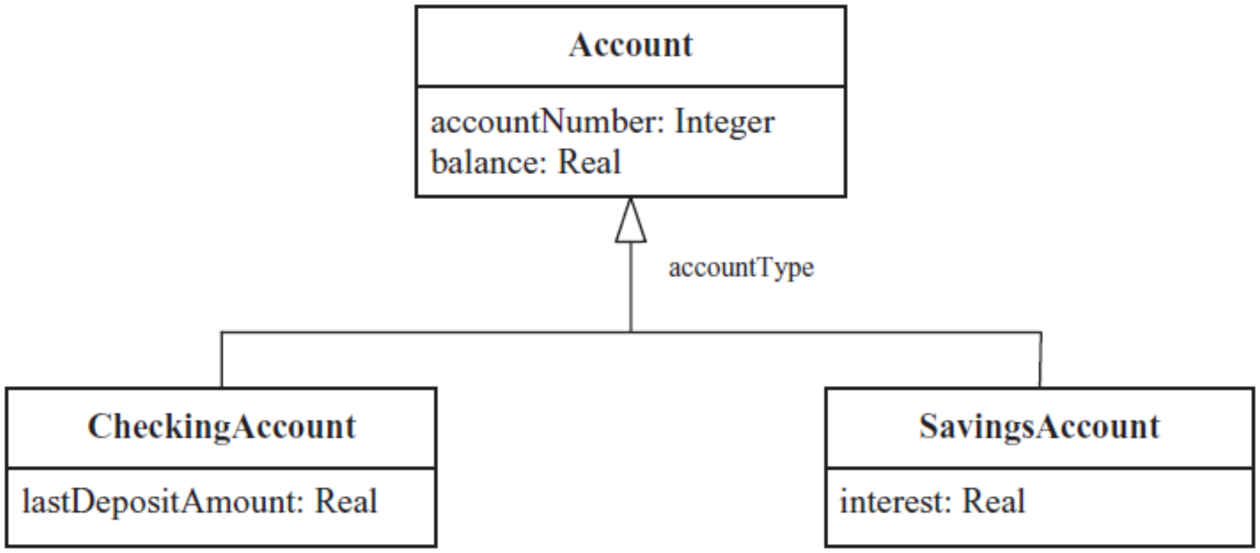
Relationships in Class Diagrams

- Association
- Dependency
- Whole/Part: composition and aggregation
- Generalization/Specialization: inheritance
- Implementation

Generalization = Implementation
> composition > aggregation
> Association > Dependency

GENERALIZATION/SPECIALIZATION

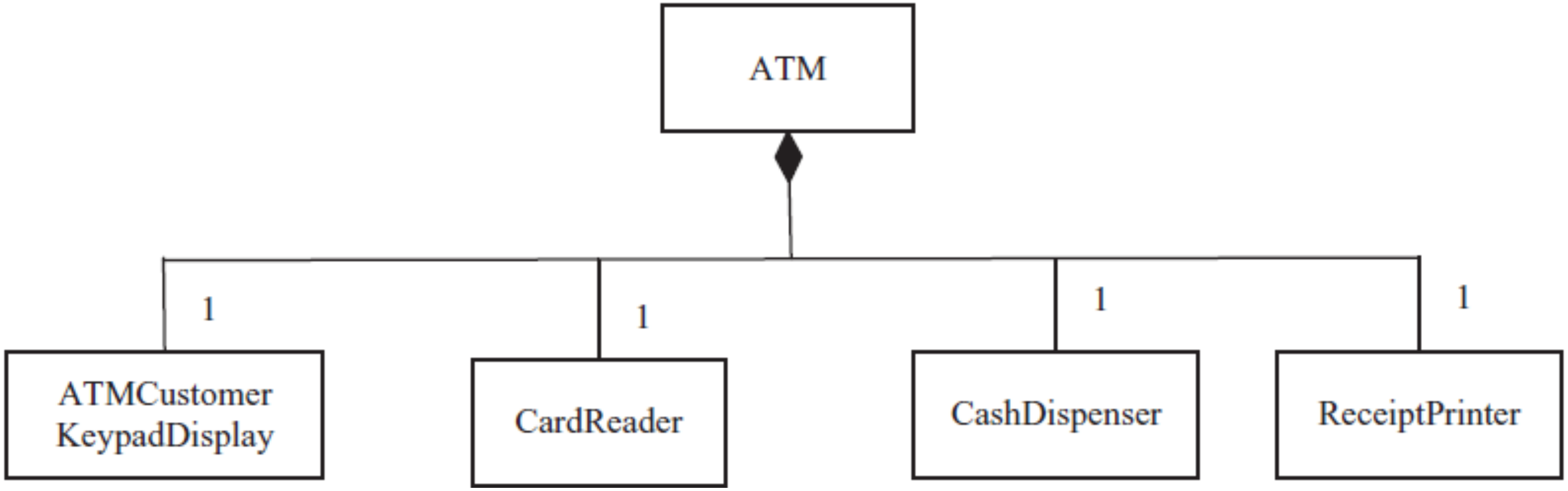
- Common attributes are abstracted into a generalized class (superclass)
- The different attributes are properties of the specialized class (subclass)



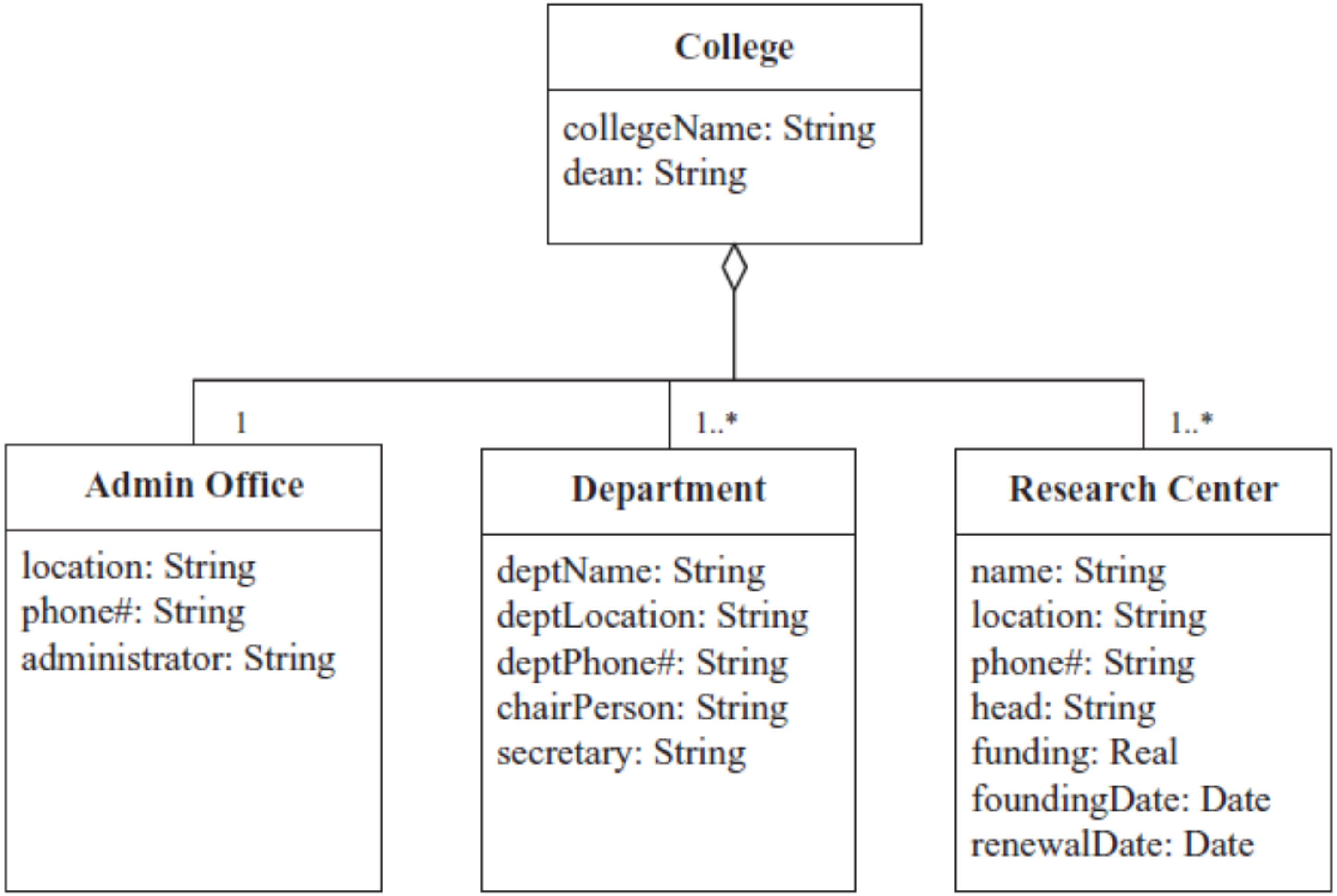
Composition and Aggregation

- Composition and aggregation hierarchies address a class that is made up of other classes
  - A composite class often involves a physical relationship between the whole and the parts
  - In an aggregation, part instances can be added to and removed from the aggregate whole

COMPOSITION (EXAMPLE)



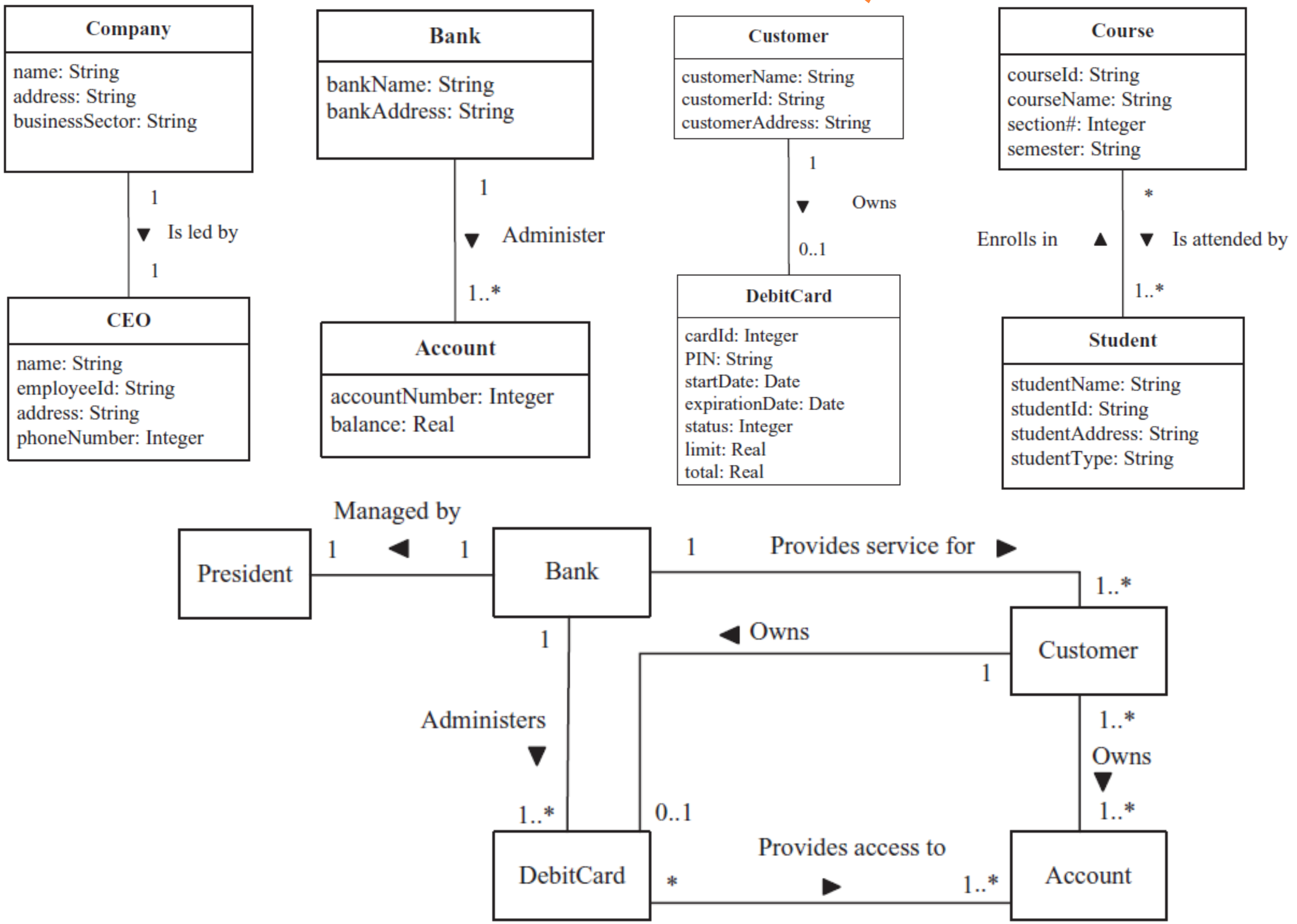
AGGREGATION (EXAMPLE)



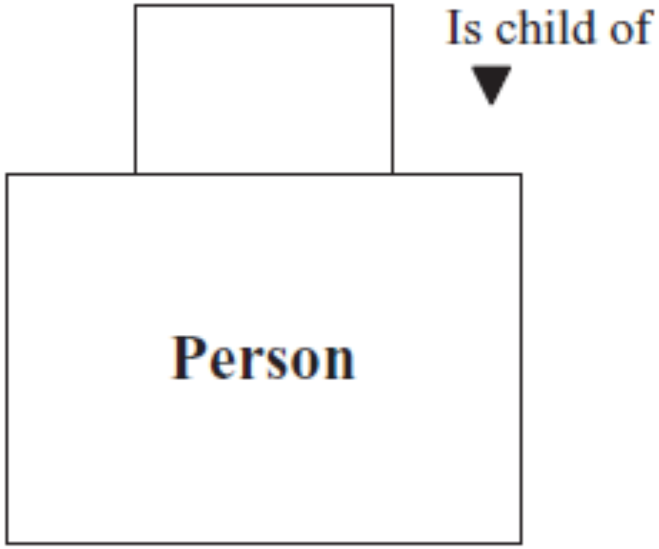
ASSOCIATION

- An association defines a relationship between two or more classes, denoting a static, structural relationship between classes
- Associations are inherently bidirectional
  - The name of the association is in the forward direction
    e.g., Employee *Works in* Department
  - There is also an implied opposite direction of the association (which is often not explicitly stated)
    e.g., Department *Employs* Employee
- In class diagrams, association names usually read from left to right and top to bottom

Association (Example)

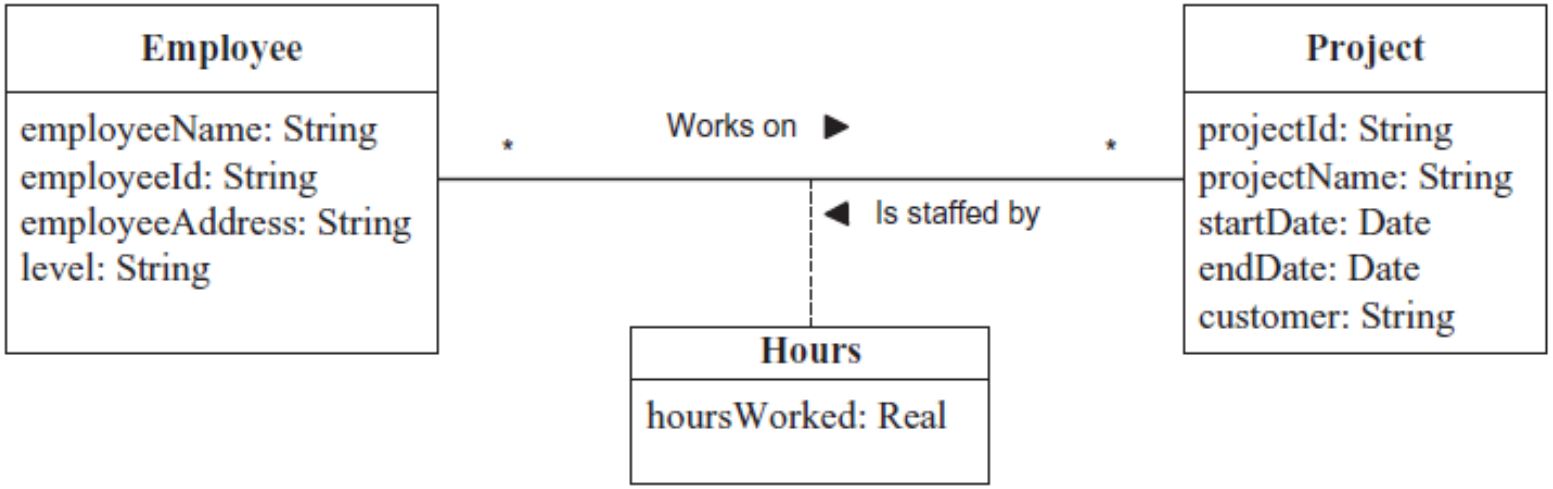


Unary Associations



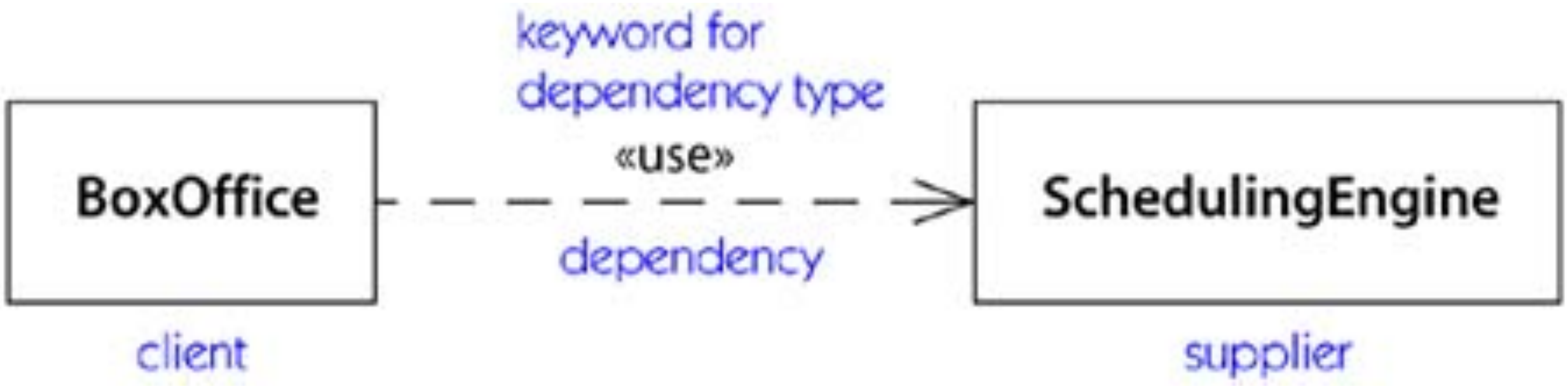
ASSOCIATION CLASSES

- Association Class models an association between two or more classes
- The attributes of an association class are the attributes of the association



DEPENDENCY

- Dependency is a weak association
  - A class depends on another class to collaborate for a task
  - Dependency is a temporary association



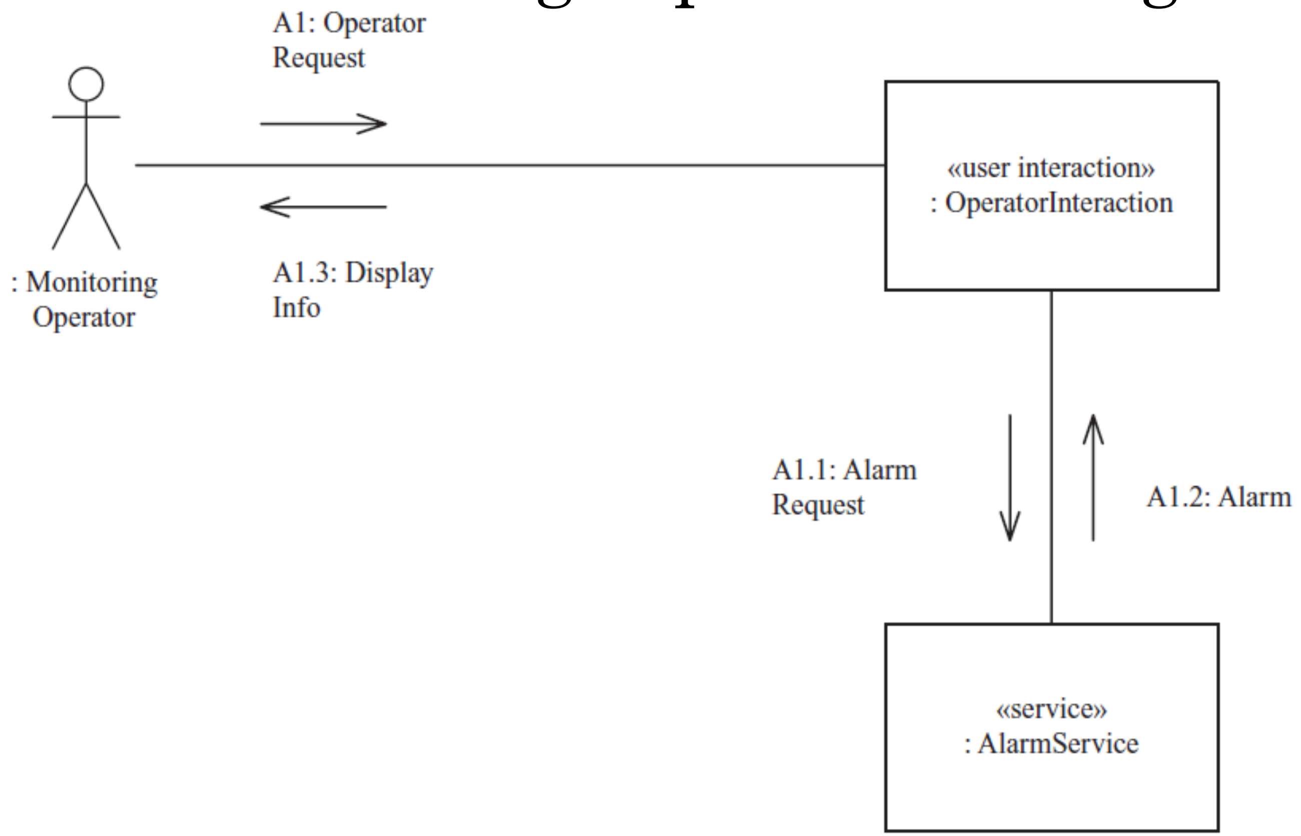
Dynamic Interaction Modeling

DYNAMIC INTERACTION MODELING

- Dynamic Interaction Modeling
  - based on the realization of the use cases developed during use case modeling
  - determine how the objects that participate in a use case dynamically interact with each other
- UML Diagrams
  - Communication Diagram
  - Sequence Diagram

DYNAMIC INTERACTION MODELING

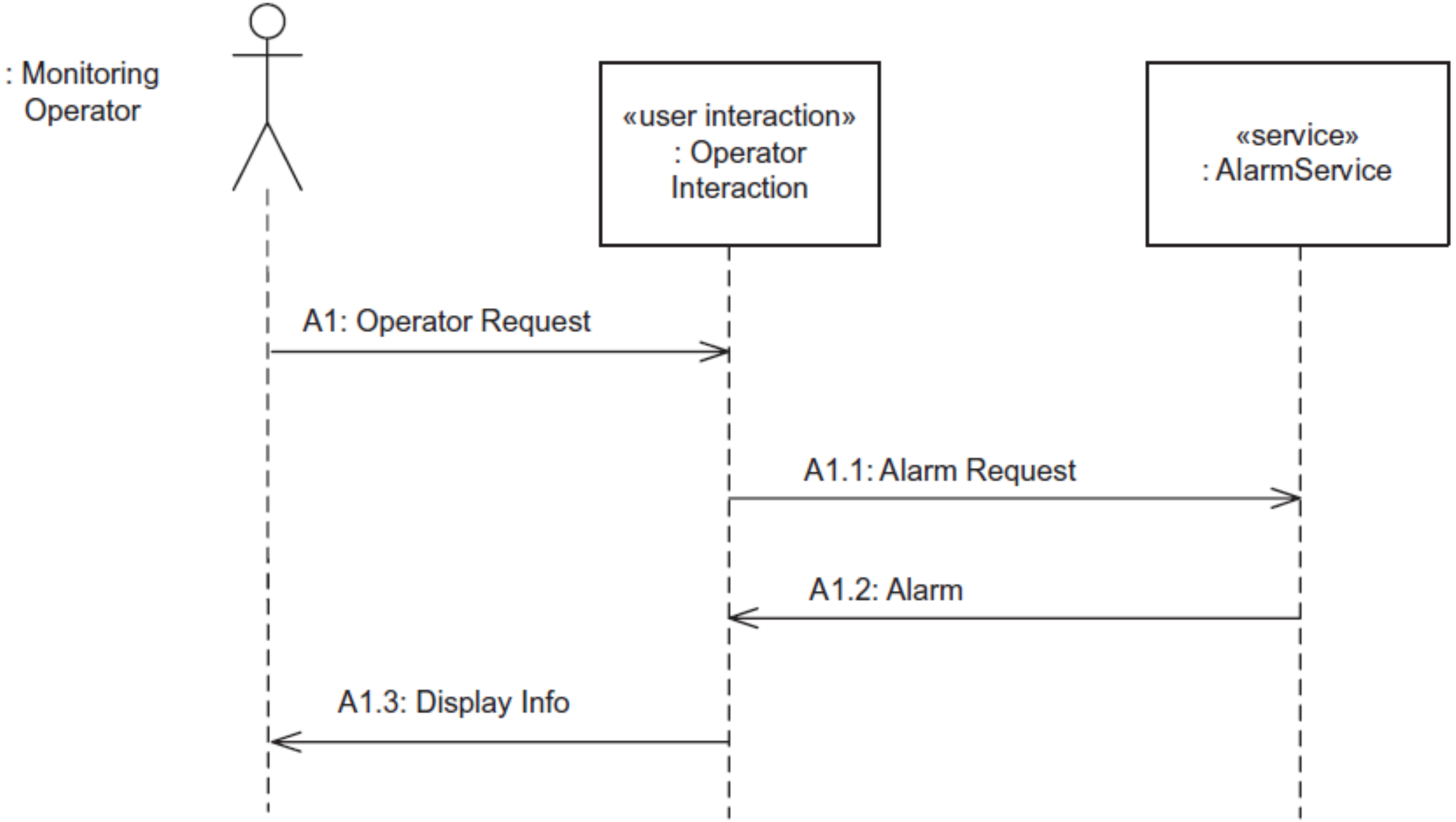
- Depict a dynamic view of a group of objects interacting with each other by showing the sequence of messages passed among them



SEQUENCE DIAGRAM

- Show object interactions arranged in time sequence
- Can also depict loops and iteration
- Sequence diagram and communication diagram depict similar information but in different ways

SEQUENCE DIAGRAM (EXAMPLE)



End of Chapter X-1