1 (a) Consider a physical bookstore, such as in a shopping mall:

i) List three actors that are involved in the design of a checkout system.

Explain the relevance of each actor.

ii) One use case is the purchase of items. List another use case at a comparable level of abstraction. Summarize purpose of each use case with a sentence.

iii) Prepare use case diagram for a physical bookstore checkout system.

7.1 Here are answers for a physical bookstore.

a. Some actors are:

- **Customer**. A person who initiates the purchase of an item.
- **Cashier**. An employee who is authorized to check out purchases at a cash register.
- **Payment verifier**. The remote system that approves use of a credit or debit card.

b. Some use cases are:

- **Purchase items**. A customer brings one or more items to the checkout register and pays for the items.
- **Return items**. The customer brings back items that were previously purchased and gets a refund.

c. Figure A7.1 shows a use case diagram.

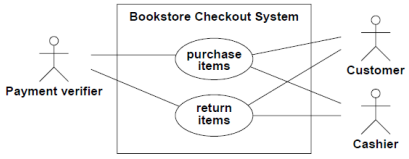


Figure A7.1 Use case diagram for a physical bookstore checkout system

(b) Write scenarios for the following situations:

Moving a bag of corn, a goose and a fox across a river in a boat. Only one thing may be carried in the boat at a time. If the goose is left alone with the corn, the corn will be eaten. If the goose is left alone with the fox, the goose will be eaten. Prepare 2 scenarios, one in which something gets eaten and one in which everything is safely transported across the river.

a. Assume that everything starts out on the east side and is to be moved to the west side. A scenario in which nothing gets eaten:
(Farmer, fox, goose, corn all on W.)
Farmer takes goose to E.
Farmer returns alone to W.
Farmer takes fox to E.
Farmer takes goose to W.
Farmer takes corn to E.
Farmer returns alone to W.
Farmer takes goose to E.
(Farmer, fox, goose, corn all on E.)

A scenario in which something gets eaten:
(Farmer, fox, goose, corn all on W.)
Farmer takes goose to E.
Farmer returns alone to W.
Farmer takes corn to E.
Farmer returns alone to W.
Goose eats corn.
Farmer takes fox to E.
(Farmer, fox, goose on E. Corn is gone.)

2 (a) How are global resources handled while designing a system and how are boundary conditions handled?

Handling Global Resources

- The system designer must identify global resources and determine *mechanisms for controlling access* to them.
- Kinds of global resources:

– *Physical units*
Processors, tape drivers…
- *Spaces*
Disk spaces, workstation screen…
- *Logical name*
Object ID, filename, class name
- *Access to shared data*
Database

Handling Global Resources

ATM example

- Bank codes and account numbers are global resources.
- Bank codes must unique within the context of a consortium.
- Account codes must be unique within the context of a bank.

Handling Boundary Conditions

- Most of system design is concerned with steady-state behaviour, but boundary conditions are also important
- Boundary conditions are

– Initialization
– Termination
– Failure

Handling Boundary Conditions

Initialization

The system must initialize constant data, parameters, global variables, …

Termination

Release any external resources that it had reserved.

Failure

Unplanned termination of a system. The good system designer plans for orderly failure

3 (a) With suitable examples, explain different use case relationships.

Use Case Relationships

Complex use cases can be built from smaller pieces with the include, extend and generalization relationships.

1. Include Relationship:

The include relationship incorporates one use case within the behavior sequence of another use case .

The UML notation for an include relationship is a dashed arrow from source (including) to target (included) use case.

The keyword <<include>> annotates the arrow.

Figure shows an example from an online stock brokerage system.

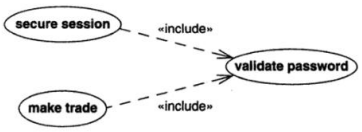


Figure 8.1 Use case inclusion. The *include* relationship lets a base use case incorporate behavior from another use case.

2. Extend Relationship

- The extend relationship is like an include relationship looked at from the opposite direction, in which the extension adds itself to the base rather than the base explicitly incorporating the extension.
- The UML notation is dashed arrow from extension use case to the base case. The keyword <<extend>> annotates the arrow.

Figure 8.2 Use case extension. The *extend* relationship is like an *include* relationship looked at from the opposite direction. The extension adds itself to the base.

3.Generaliztion: A parent use case represents a general behavior and child use cases add variations analogous to generalization among classes

Figure 8.3 Use case generalization. A parent use case has common behavior and child use cases add variations, analogous to generalization among classes.

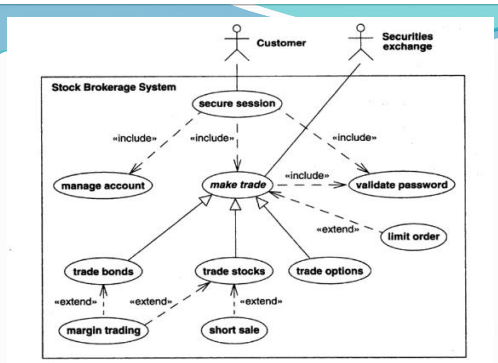


Figure 8.4 Use case relationships. A single use case diagram may combine several kinds of relationships.

(b) What are the guidelines for sequence models?

Guidelines

- Prepare at least one scenario per use case
- Abstract the scenarios into sequence diagrams
- Divide complex interactions
- Prepare a sequence diagram for each error condition

4 (a) Write short notes on:

i) Recurring downwards.

4. Recursing Downward

- To organize operations as layers.
  - Operations in higher layers invoke operations in lower layers.
  - Design process generally works top down- you start with the higher-level operations and proceed to define lower-level operations.
- Two ways of downward recursion:
  - By functionality
  - By mechanism
- Any large system mixes functionality layers and mechanism layers.

29

Functionality Layers

- Functionality recursion means that you take the required high-level functionality and break it into lesser operations.
- Make sure you combine similar operations and attach the operations to classes.
- An operation should be coherent, meaningful, and not an arbitrary portion of code.
- ATM eg., **use case** decomposed into **responsibilities** (see sec 15.3). Resulting **operations** are assigned to classes (see sec 15.4.4). If it is not satisfied rework them

30

Mechanism Layers

- Mechanism recursion means that you build the system out of layers of needed support mechanisms.
- In providing functionality, you need various mechanisms to store information, sequence control, coordinate objects, transmit information, perform computations and other kinds computing infrastructure
- These mechanisms don't show up explicitly in the high-level responsibilities of a system, but they are needed to make it all work.
- E.g. Computing architecture includes
  - Data structures, algorithms, and control patterns.

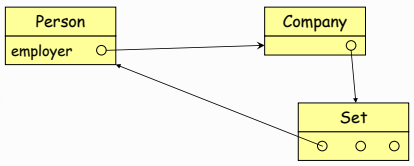
32

ii) Two-way associations.

Two-way Association

- Many associations are traversed in both directions, not usually with equal frequencies
- Three approaches for implementation
  - Implement one-way
  - Implement two-way
  - Implement with an association object

Two-way Association



iii) Reverse engineering vs forward engineering.

Forward Engineering	Reverse Engineering
Given Requirements – develop an application.	Given an application – deduce tentative requirement.
More certain – developer develops according to requirement.	Less certain – can yield different requirements.
Prescriptive – developers are told how to work	Adaptive – find out what the developers actually did.
More mature – skilled staffs available.	Less mature – skilled staff are not present
Time consuming	Less time consumption.
Model will fail if not developed correctly	Salvaging partial information is still useful.

iv) Wrapping.

5. Wrapping:

• Some applications – written long ago, missing documentation, lack guidance and poorly understood.

•Changes can threaten their viability and risk introducing bugs. So need to limit changes to these type of applications/system.

•Solution – Isolate the code and build a wrapper around it.

•Wrapping – a collection of interfaces that control access to the system/application.

•It consists of a set of boundary classes that provide the interface and it calls the existing system/application.

•Advantage – adding few functionality is easy.

5 (a) What do you mean by concurrency? Explain Aggregation concurrency and the different types of concurrency among objects.

Aggregation Concurrency

- A state diagram for an assembly is a collection of state diagrams one for each part.
- The aggregate state corresponds to the combined states of all the parts.
- The aggregate state is one state from the first diagram, *and* a state from the second diagram, *and* a state from each other diagram.
- Transition for one object can depend on another object being in a given state.

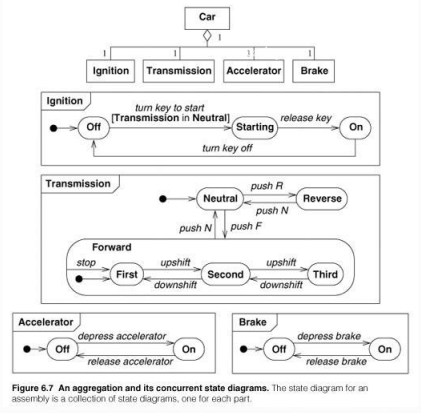


Figure 6.7 An aggregation and its concurrent state diagrams. The state diagram for an assembly is a collection of state diagrams, one for each part.

Concurrency within an Object

- Some objects can be partitioned into subsets of attributes or links.
- Each of the partitioned subset has its own subdiagram.
- The state of the object comprises one state from each subdiagram.
- The subdiagrams need not be independent; the same event can cause transitions in more than on subdiagram.

Designing Algorithms- steps

i. Choose *algorithms* that minimize the cost of implementing operations.
ii. Select *data structures* appropriate to the algorithms
iii. Define new internal classes and operations as necessary.
iv. Assign operations to appropriate classes.

18

Concurrency within an Object

- The play of a bridge rubber

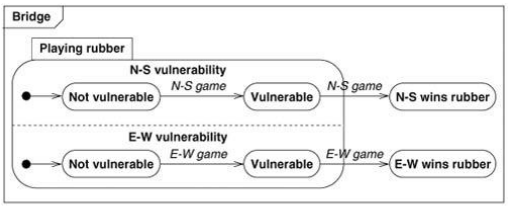


Figure 6.8 Bridge game with concurrent states. You can partition some objects into subsets of attributes or links, each of which has its own subdiagram.

i. Choosing algorithms

(Choose algorithms that minimize the cost of implementing operations)

- When efficiency is not an issue, you should use simple algorithms.
- Typically, 20% of the operations consume 80% of execution time.

19

6. List and explain the steps involved in design of algorithms.

3.Designing Algorithms

- Formulate an ***algorithm*** for each operation
- The analysis specification tells *what* the operation does for its clients
- The algorithm show *how* it is done

17

ATM Example

- Interactions between the consortium computer and bank computers could be complex.
- Considerations:
  - Distributed computing
  - The scale of consortium computer (scalability)
  - The inevitable conversions and compromises in coordinating the various data formats.
- All these issues make the choice of algorithms for coordinating the consortium and the banks important

21

ii. Choosing Data Structures

(select data structures appropriate to the algorithm)

- Algorithms require data structures on which to work.
- They organize information in a form convenient for algorithms.
- Many of these data structures are instances of ***container classes***.
- Such as arrays, lists, queues, stacks, set...etc.

23

iii. Defining New Internal Classes and Operations

- To invent new, low-level operations during the decomposition of high-level operations.
- The expansion of algorithms may lead you to create new classes of objects to hold intermediate results.
- ATM Example:
  - *Process transaction* uses case involves a customer receipt.
  - A *Receipt* class is added.

24

iv. Assigning Operations to Classes

(assign operations to appropriate classes)

- How do you decide what class owns an operation?
  - Receiver of action
    - To associate the operation with the *target* of operation, rather than the *initiator*.
  - Query vs. update
    - The object that is changed is the target of the operation
  - Focal class
    - Class centrally located in a star is the operation's target
  - Analogy to real world

25

Process transaction

ATM Example

- *Process transaction* includes:
  - Withdrawal includes responsibilities:
    - Get amount from customer, verify that amount is covered by the account balance, verify that amount is within the bank's policies, verify that ATM has sufficient cash, ...
    - A database transaction ensures all-or-nothing behavior.
  - Deposit
  - Transfer
- Customer.getAccount(), account.verifyAmount(amount), bank.verifyAmount(amount), ATM.verifyAmount(amount)

26

(b) What are the inputs and outputs of reverse engineering?

Inputs to reverse Engineering:

- **Programming code** – helps to understand the flow of control and the data structure. Comments, variables and functions – deeper understanding.
- **Database Structure** – Specifies the data structure and many constraints.
- **Data** – discover much of the data structure.
- **Documentation** – user manuals, data dictionaries – entities and definitions.
- **Application understanding** – for better understanding of interfaces.
- **Test cases** – normal flow of control and unusual situations.

Outputs from Reverse Engineering:

- **Models** – Provides a basis for understanding the original software and building any successor software.
- **Mappings** – Bind programming code to state and interaction models.
- **Logs** – record their observations and pending questions.

7 (a) For an ATM example:

1. Analyze the different use cases by designing a use case diagram.

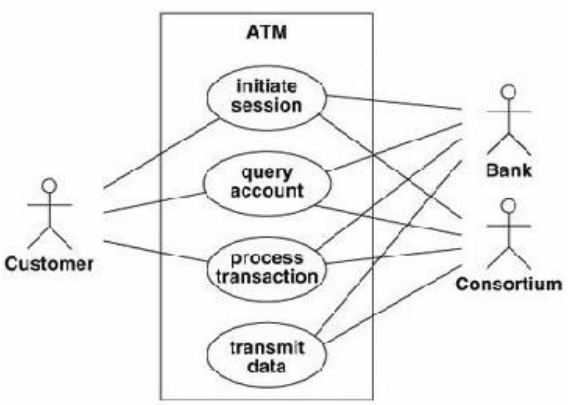


Figure 13.1 Use case diagram for the ATM. Use cases partition the

2. Explain process transaction scenario.

Process transaction Scenario

ATM displays a menu of accounts and commands.
User selects an account withdrawal.
ATM asks for the amount of cash.
User enters 10000.
ATM verifies that the withdrawal satisfies its policy limits.
ATM contacts the consortium and bank and verifies that the account has sufficient funds.
ATM dispenses the cash and asks the user to take it.
User takes the cash.
ATM displays a menu of accounts and commands.

3. Bring out initial and final events for each of the use cases.

4. Find initial and final events

ATM example
- *Initial session*
  - Initial event
    The customer's insertion of a cash card.
  - final event
    The system keeps the cash card, or
    The system returns the cash card.

ATM example
- *Query account*
  - Initial event
    A customer's request for account data.
  - final event
    The system's delivery of account data to the customer.

4. Find initial and final events

ATM example
- *Process transaction*
  - Initial event
    The customer's initiation of a transaction.
  - final event
    Committing or
    Aborting the transaction

ATM example
- *Transmit data*
  - Initial event
    Triggered by a customer's request for account data, or
    Recovery from a network, power, or another kind of failure.
  - final event
    Successful transmission of data.

4. With the help of activity diagram, show the possible responses for the verification of card inserted by the user at the ATM.

ATM Example

Activity diagram for card verification

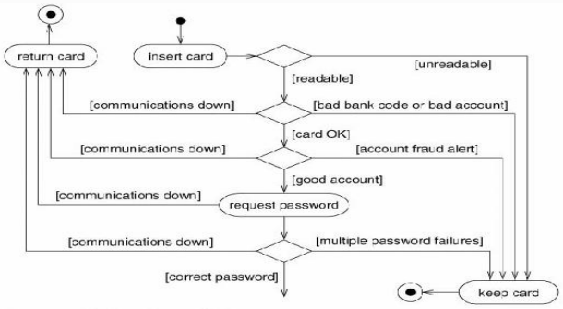


Figure 13.5 Activity diagram for card verification. You can use activity diagrams to document business logic but do not use them as an excuse to begin premature implementation.

8 (a) Explain the different tasks involved in design optimization.

6. Design Optimization

- To design a system is to first get the logic correct and then optimize it. (because it is difficult to optimize a design at the same time as you create it)
- Often a small part of the code is responsible for most of the time or space costs.
- It is better to focus optimization on the **critical areas**, than to spread effort evenly.

37

Design Optimization

- Optimized system is **more obscure** and less **likely to be reusable**.
- You must strike an appropriate **balance between efficiency and clarity.**
- **Tasks** to optimization:
  i. Provide efficient access paths.
  ii. Rearrange the computation for greater efficiency.
  iii. Save intermediate results to avoid recomputation.

38

i. Adding Redundant Associations for Efficient Access

- Rearrange the associations to optimize critical aspects of the system.
- Consider employee skills database

Figure 15.5 Analysis model for person skills. Derived data is undesirable during analysis because it does not add information.

- *Company.findSkill*( ) returns a set of persons in the company with a given skill.
- Suppose the company has 1000 employees, Each of whom has 10 skills on average. A simple nested loop would traverse Employs 1000 times and HasSkill 10,000 times

39

Adding Redundant Associations for Efficient Access

- In case where the number of hits from a query is low because few objects satisfy the test, an *index* can improve access to frequently retrieved objects.

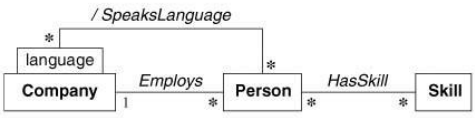


Figure 15.6 Design model for person skills. Derived data is acceptable during design for operations that are significant performance bottlenecks.

40

ii. Rearranging Execution order for Efficiency

-eliminate dead paths

iii. Saving Derived Values to Avoid Recomputation

You must update the cache if any of the objects on which it depends are changed. There are three ways to handle updates:

- Explicit update: the designer inserts code into the update operation of source attributes to explicitly update the derived attributes that depend on it.
- Periodic Recomputation
- Active values: An active value is a value that is automatically kept consistent with its source values.