Object Oriented Concepts with UML (203105207)

Prof. Shaleen Shukla, Assistant Professor
Information Technology

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-4

Analysis and Design

SOFTWARE DEVELOPMENT STAGES

Software Development Stages

- System conception
- Analysis
- System design
- Class design
- Implementation
- Testing
- Training
- Deployment
- Maintenance

Software Development Life Cycle

•SDLC is a process used by the software industry to design, develop and test high quality software's.

•Two Types

•Waterfall Development

•Iterative Development

Model: Model is an abstraction of something for the purpose of understanding it, before building.

Abstract: Thought, idea

SOFTWARE DEVELOPMENT LIFE CYCLE

Waterfall Development

•The developers perform the software development stages in a rigid linear sequence
•No backtracking
•First capture requirement
•Then perform a system design
•Then prepare a class design
•Then implementation
•Testing
•Deployment..

Waterfall Development

•Each stage is completed in its entirety before the next stage is begun

•It is suitable for well-understood applications with predictable outputs from analysis and design

•Too many organizations attempt to follow a waterfall when requirements are fluid

•This leads to familiar situation where developers complain about changing requirement, the business complains about inflexible software development

Waterfall Development

•It is not a useful system until completion
•This makes it difficult to assess progress and correct a project that has gone awry

Examples of Waterfall Model

In the olden days, Waterfall model was used to develop enterprise applications like Customer Relationship Management (CRM) systems, Human Resource Management Systems (HRMS), Supply Chain Management Systems, Inventory Management Systems, Point of Sales (POS) systems for Retail chains etc.

Waterfall Development

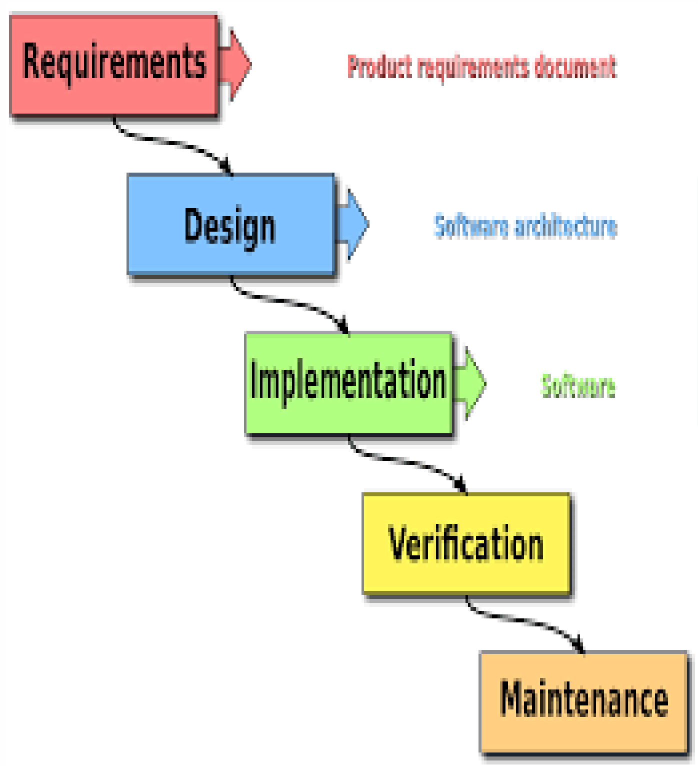


Figure: 4.1 Waterfall Development Image source : en.wikipedia.org

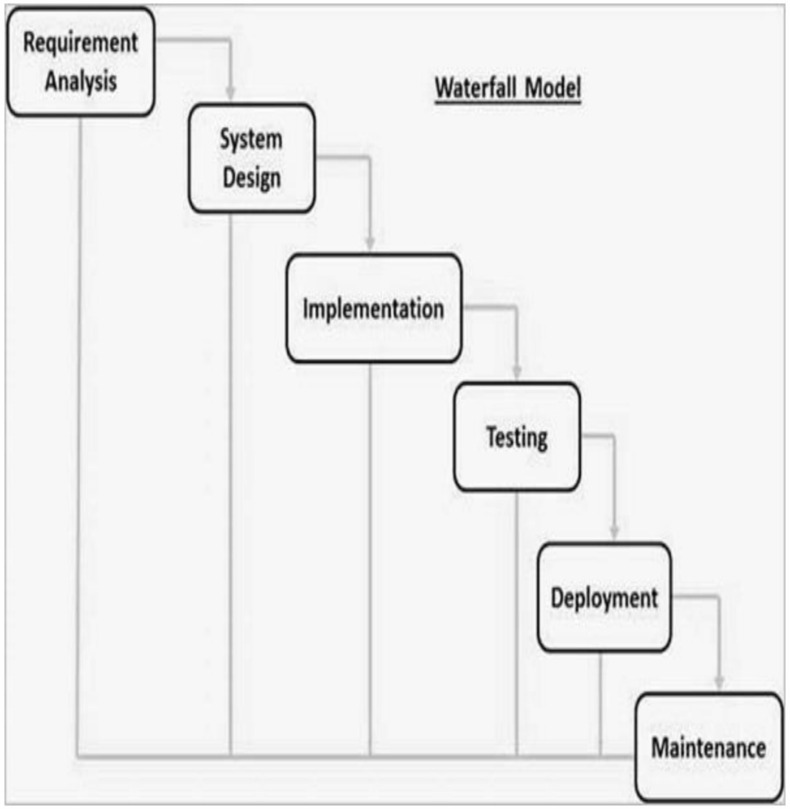


Figure: 4.2 Waterfall Development Image source : tutorialspoint.om

Customer Relationship Management System

Customer relationship management (**CRM**) is an approach to **managing** a company's interaction with current and potential **customers**. It uses data analysis about **customers**' history with a company to improve business **relationships** with **customers**, specifically focusing on **customer** retention and ultimately driving sales growth.

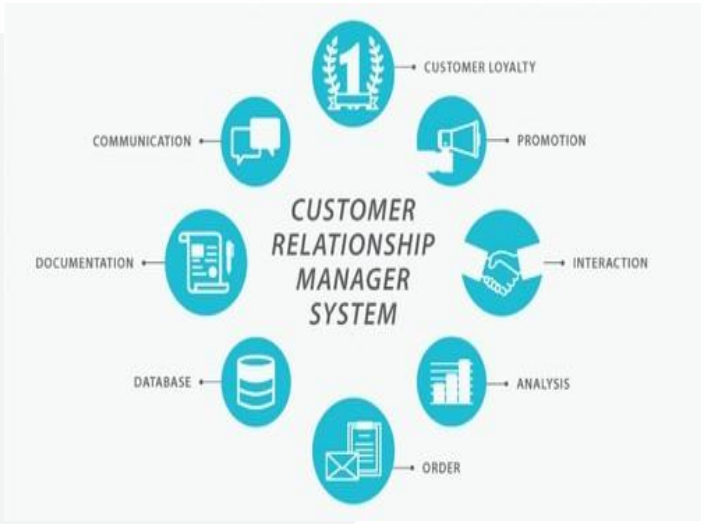


Figure: 4.3 Customer Relationship Management System

Image source : indiamart.com

Iterative Development

- It is more flexible
- First you develop the nucleus system
    - Analyzing
    - Designing
    - Implementing
    - Delivering working code
- Then you grow scope of the system, adding properties and behavior to existing objects
- There are multiple iterations as the system evolves to the final deliverable
- Each iteration includes full complement of stages
- Unlike strict sequence of the waterfall model, iterative development can interleave the different stages
- need not construct the entire system in lock step
- Some parts may be completed early

Iterative Development

•Other crucial parts are completed later

•Each iteration ultimately yields an executable system that can be integrated and tested

•If any problem, you can move backward to earlier stage for rework

•It gracefully responds to change and minimizes risks of failure

•Examples of Iterative Model

•User Interfaces. A requirement of a film company needs a custom system for equipment tracking, costumes, and props. ...

•Graphic Design. The advertising agencies creative department has 2 weeks to produce an ad print to a customer. ...

•Architecture. ...

•Marketing. ...

•Urban Design.

Iterative Process/Development Model

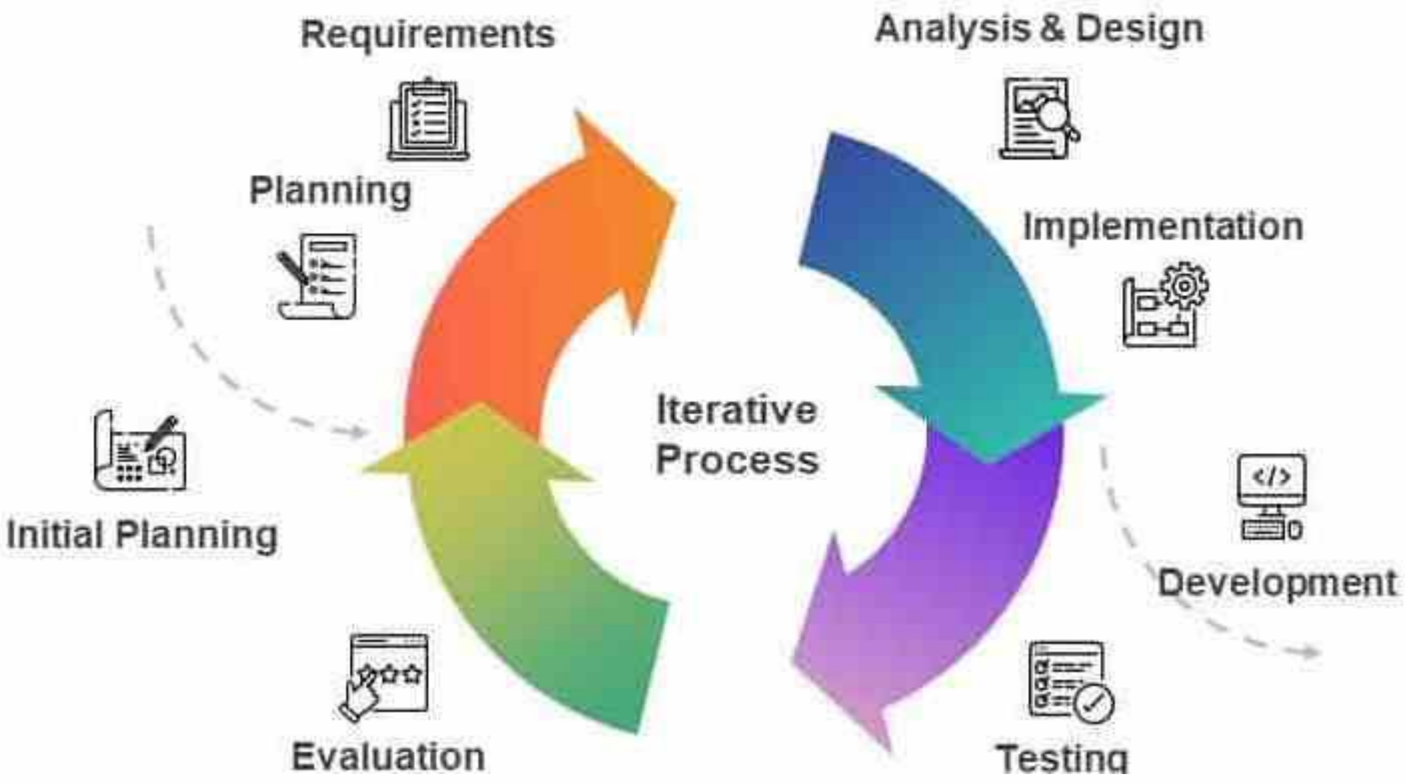


Figure: 4.4 Iterative /Development Model

Image source : slidebazzar.com

DOMAIN ANALYSIS

Domain Analysis

INTRODUCTION:

•Domain analysis is concerned with devising a precise, concise, understandable and correct model of the real world

•During analysis, we build models and begin to understand the requirements deeply

•To build a domain model,

- •Interview business experts
- •Examine requirements statements
- •Restate them rigorously

•The successful analysis model states

- •What must be done, without restricting how it is done
- •Avoids implementation decisions

Domain Analysis

OVERVIEW OF ANALYSIS:

•Analysis begins with a problem statement generated during system conception

•The statement may be incomplete or informal

•Analysis makes it more precise and exposes ambiguities and inconsistencies

•Statements in natural language are often ambiguous, incomplete and inconsistent

•The analysis model is concise representation of the problem that permits answering questions and building a solution

Parul® University

DIGITAL LEARNING CONTENT

Domain Analysis

OVERVIEW OF ANALYSIS:

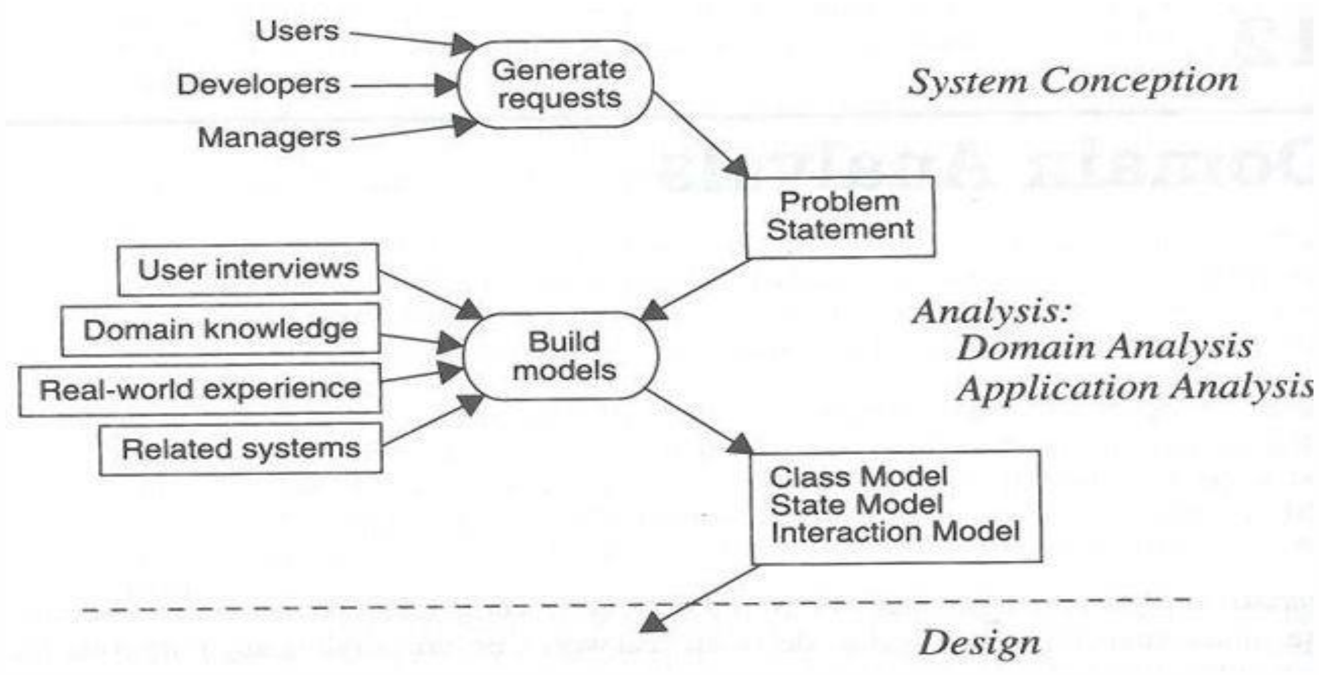


Figure: 4.5 Overview of Analysis

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Analysis

OVERVIEW OF ANALYSIS:

•Subsequent design steps refer to analysis model rather than the original vague problem statement

•The analysis model addresses the three aspects of objects

- •Static structure of objects
- •Interactions among objects
- •Lifecycle histories of objects

Domain Class Model

•The domain model shows the static structure of the real world system and organizes it into workable pieces

•It describes real world classes and its relationships to each other

•Make sure you consider all information that is available and do not rely on a single source

Domain Class Model

Steps to be performed to construct a domain class model

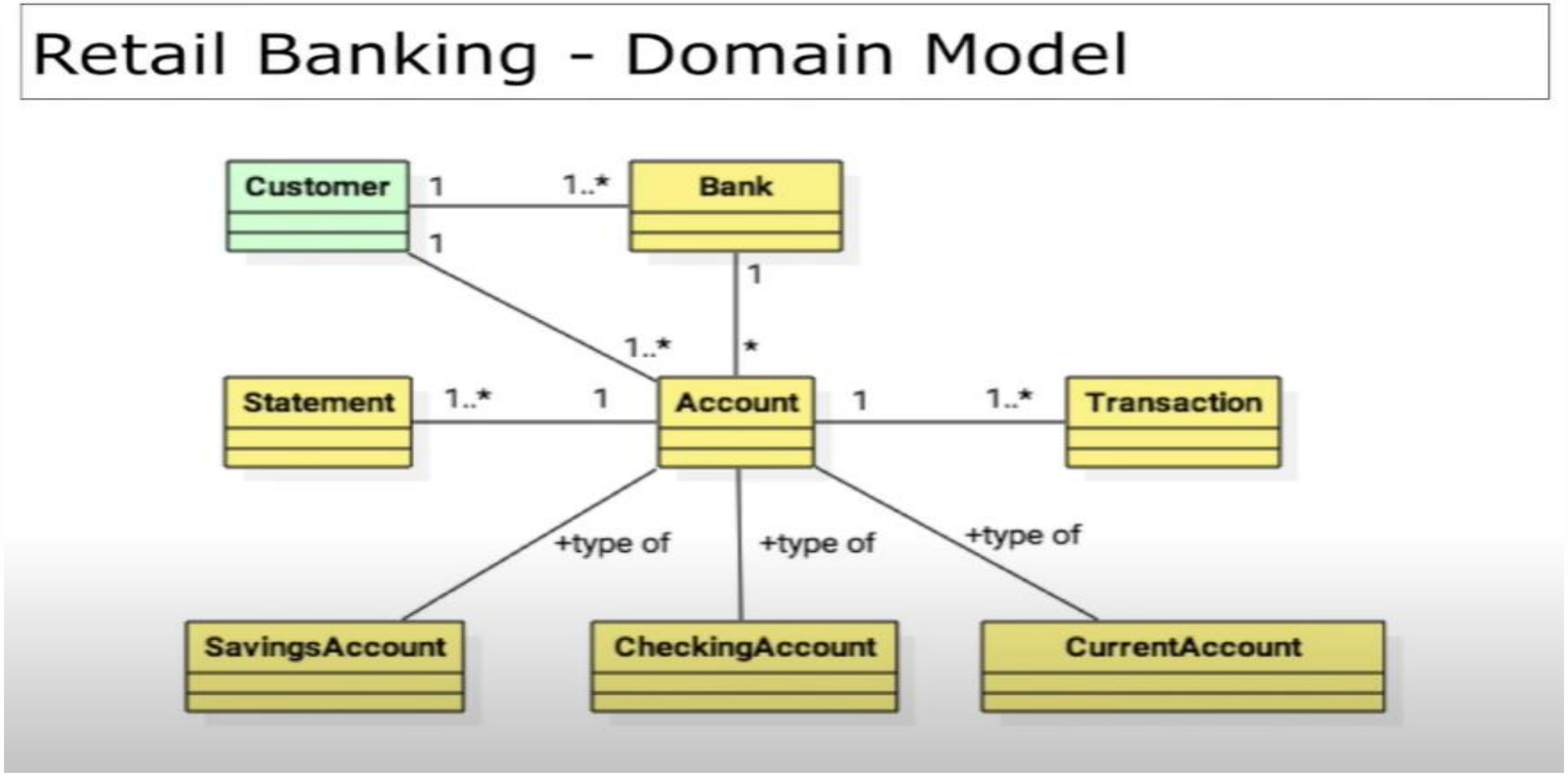
- Find classes
- Prepare a data dictionary
- Find associations
- Find attributes of objects and links
- Organize and simplify classes using inheritance
- Verify that access paths exists for likely queries
- Iterate and refine the model
- Reconsider the level of abstraction
- Group classes into packages

Domain Class Model

FIND CLASSES

•Find relevant classes for objects from the application domain
•Object includes physical entities as well as concepts
•Avoid computer implementation constructs such as linked lists and subroutines
•Classes often correspond to nouns
•The idea is to capture concepts

Domain Class Model



Domain Class Model

FIND CLASSES

- For e.g. "a reservation system to sell tickets to performances at various theaters"
- Tentative classes: Reservation, System, Tickets, Performance, and Theatre.
- Don't worry much about inheritance or high-level classes

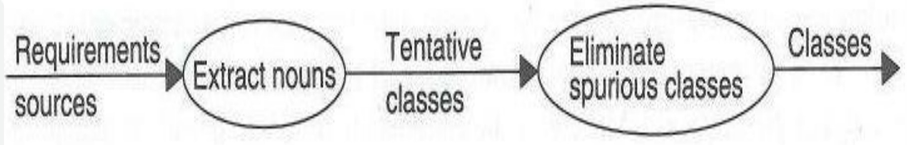


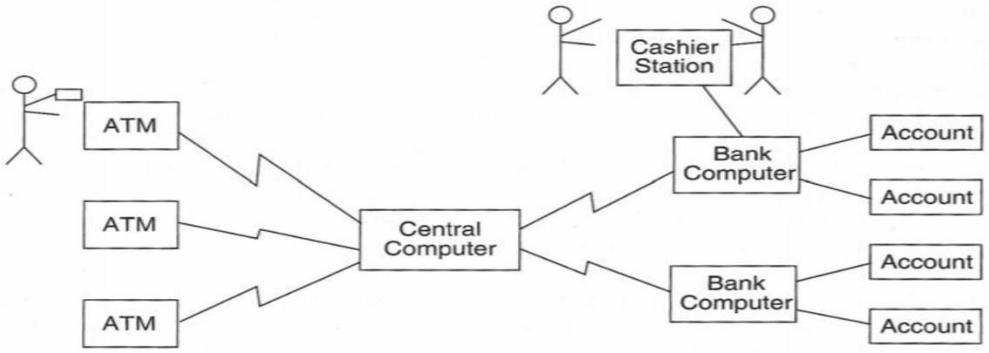
Figure: 4.6 Finding Classes

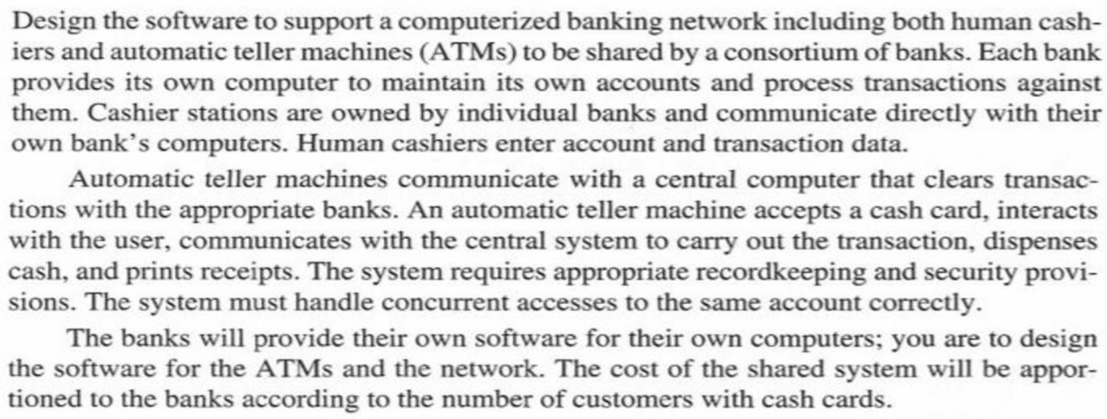
Image source : Book: OOMD with UML by Blaha/Rumbaugh

Parul® University

DIGITAL LEARNING CONTENT

ATM Case Study




Design the software to support a computerized banking network including both human cashiers and automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank's computers. Human cashiers enter account and transaction data.

Automatic teller machines communicate with a central computer that clears transactions with the appropriate banks. An automatic teller machine accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispenses cash, and prints receipts. The system requires appropriate recordkeeping and security provisions. The system must handle concurrent accesses to the same account correctly.

The banks will provide their own software for their own computers; you are to design the software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.

Figure: 4.7 ATM Network . Case Study Throughout the Unit

Image source : Book: OOMD with UML by Blaha/Rumbaugh

ATM Example: Tentative Classes

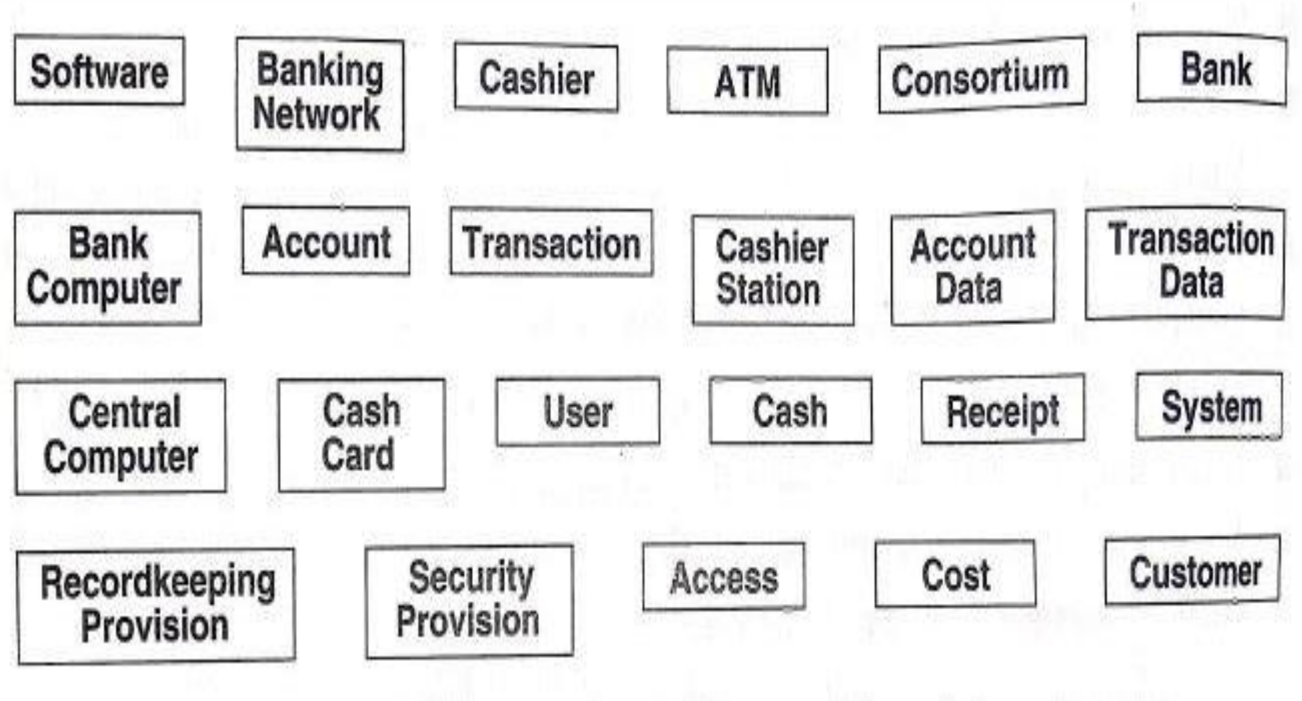


Figure: 4.8 ATM Classes Extracted from problem statement nouns

Image source : Book: OOMD with UML by Blaha/Rumbaugh

ATM Example: Additional Classes

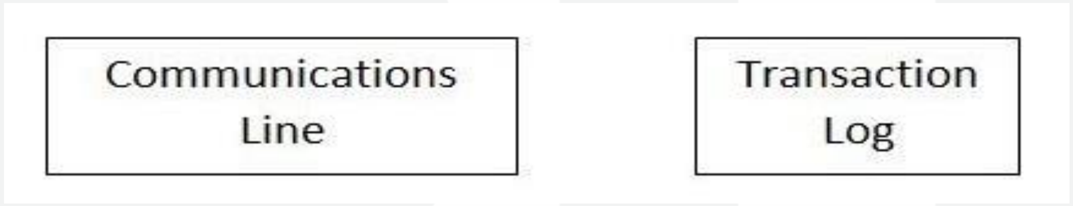


Figure: 4.9 ATM Classes Identified from knowledge of problem domain

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

KEEPING THE RIGHT CLASSES

•Discard unnecessary and incorrect classes according to following criteria

•Redundant classes

  •two classes express the same concept, keep the most descriptive name

•Irrelevant class

  •Class having nothing to do with the problem, eliminate it

•Vague

  •A class should be specific

  •Some tentative classes may have ill-defined boundaries or be too broad in scope

Domain Class Model

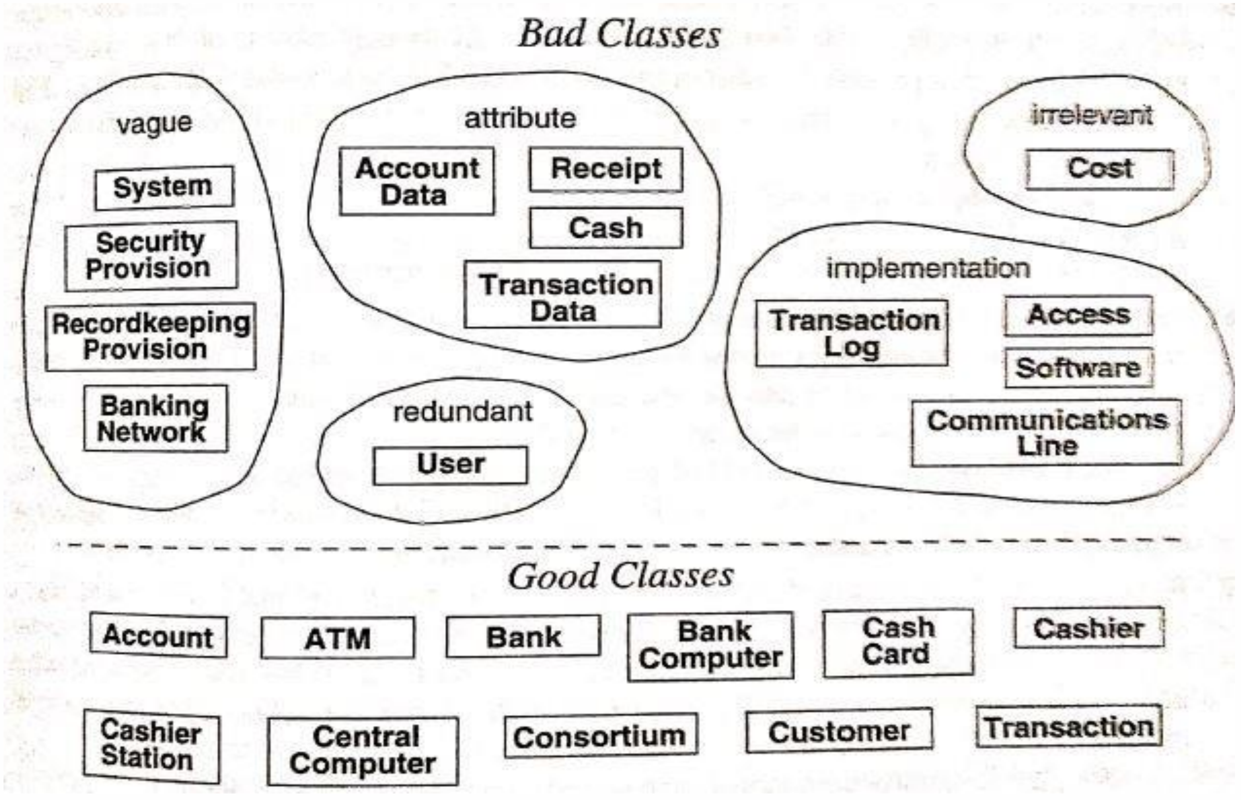


Figure: 4.10 Eliminating unnecessary classes from ATM problem

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

- Attributes
    - Names that primarily describe individual objects should be restated as attributes
    - Ex. Name, birth date, etc.
    - If the independent existence of a property is important, make it a class
- Operations
    - If name describes an operation that is applied to objects, and not manipulated in its own right, then it is not a class
    - An operations has features of its own should be modeled as a class
- Roles
    - The name of the class should reflect its intrinsic nature, Not a role that it plays in an association
    - Ex. Owner would be a poor name for a class in a car manufacturer’s database, Person would be suitable

Domain Class Model

- Implementation constructs
  - Eliminate constructs from the analysis model that are extraneous to the real world
  - You may need them during design
  - Ex. CPU, subroutine, algorithms, etc,.
- Derived classes
  - Omit the classes that can be derived from other classes
  - If a derived class is important, you can include it
  - Mark derived classes with a preceding slash ('/') in the class name

Domain Class Model

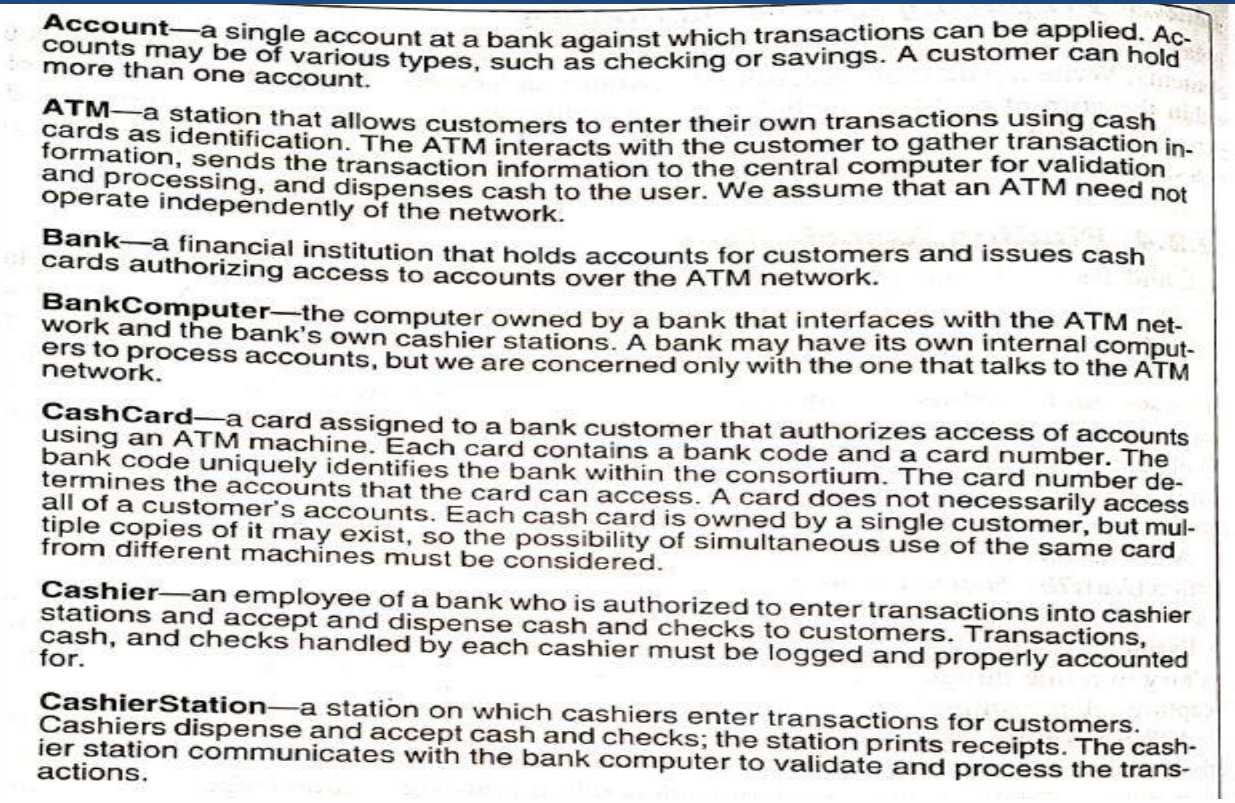
PREPARING A DATA DICTIONARY

- •Isolated words have too many interpretations
- •So prepare a data dictionary for all modeling elements
- •Write a paragraph precisely describing each class
- •Describe the scope of the class within the current problem
- •Data dictionary also describes associations, attributes, operations and enumerations

Parul® University

DIGITAL LEARNING CONTENT

Domain Class Model



Account—a single account at a bank against which transactions can be applied. Accounts may be of various types, such as checking or savings. A customer can hold more than one account.

ATM—a station that allows customers to enter their own transactions using cash cards as identification. The ATM interacts with the customer to gather transaction information, sends the transaction information to the central computer for validation and processing, and dispenses cash to the user. We assume that an ATM need not operate independently of the network.

Bank—a financial institution that holds accounts for customers and issues cash cards authorizing access to accounts over the ATM network.

BankComputer—the computer owned by a bank that interfaces with the ATM network and the bank's own cashier stations. A bank may have its own internal computers to process accounts, but we are concerned only with the one that talks to the ATM network.

CashCard—a card assigned to a bank customer that authorizes access of accounts using an ATM machine. Each card contains a bank code and a card number. The bank code uniquely identifies the bank within the consortium. The card number determines the accounts that the card can access. A card does not necessarily access all of a customer's accounts. Each cash card is owned by a single customer, but multiple copies of it may exist, so the possibility of simultaneous use of the same card from different machines must be considered.

Cashier—an employee of a bank who is authorized to enter transactions into cashier stations and accept and dispense cash and checks to customers. Transactions, cash, and checks handled by each cashier must be logged and properly accounted for.

CashierStation—a station on which cashiers enter transactions for customers. Cashiers dispense and accept cash and checks; the station prints receipts. The cashier station communicates with the bank computer to validate and process the transactions.

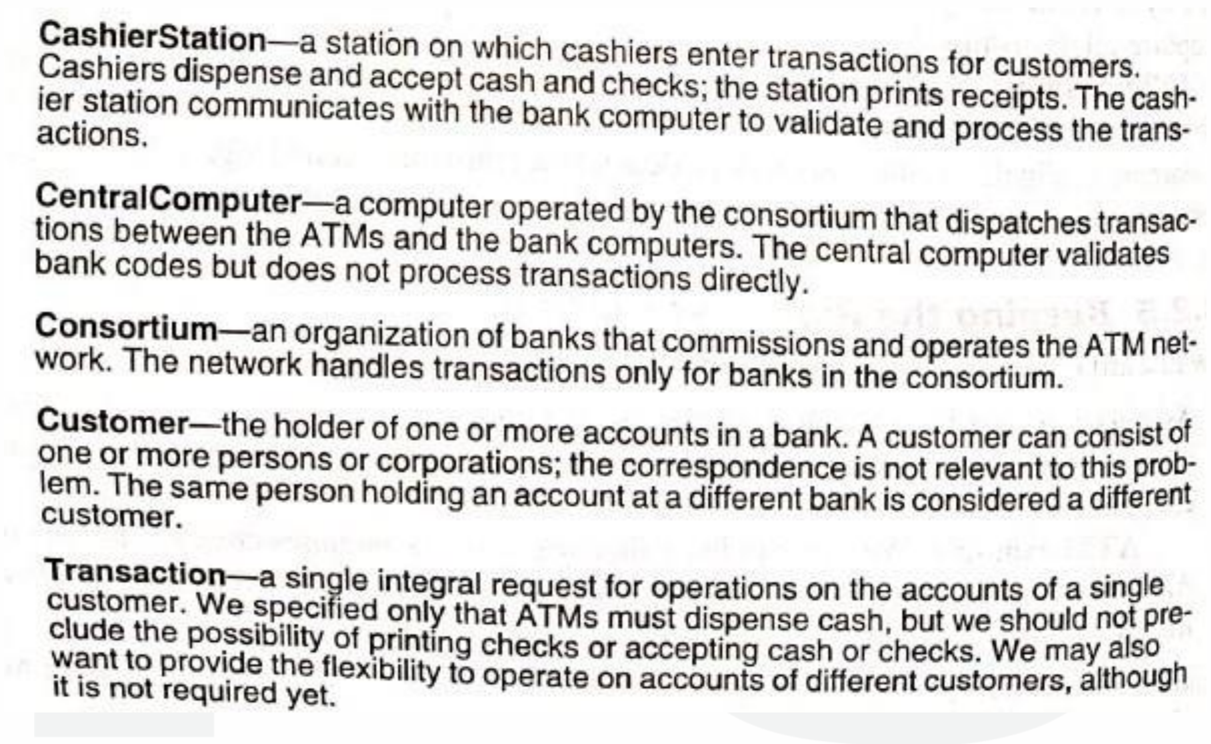
Figure: 4.11.1 Data Dictionary for ATM Classes

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Parul® University

DIGITAL LEARNING CONTENT

Domain Class Model



CashierStation—a station on which cashiers enter transactions for customers. Cashiers dispense and accept cash and checks; the station prints receipts. The cashier station communicates with the bank computer to validate and process the transactions.

CentralComputer—a computer operated by the consortium that dispatches transactions between the ATMs and the bank computers. The central computer validates bank codes but does not process transactions directly.

Consortium—an organization of banks that commissions and operates the ATM network. The network handles transactions only for banks in the consortium.

Customer—the holder of one or more accounts in a bank. A customer can consist of one or more persons or corporations; the correspondence is not relevant to this problem. The same person holding an account at a different bank is considered a different customer.

Transaction—a single integral request for operations on the accounts of a single customer. We specified only that ATMs must dispense cash, but we should not preclude the possibility of printing checks or accepting cash or checks. We may also want to provide the flexibility to operate on accounts of different customers, although it is not required yet.

Figure: 4.11.2 Data Dictionary for ATM Classes

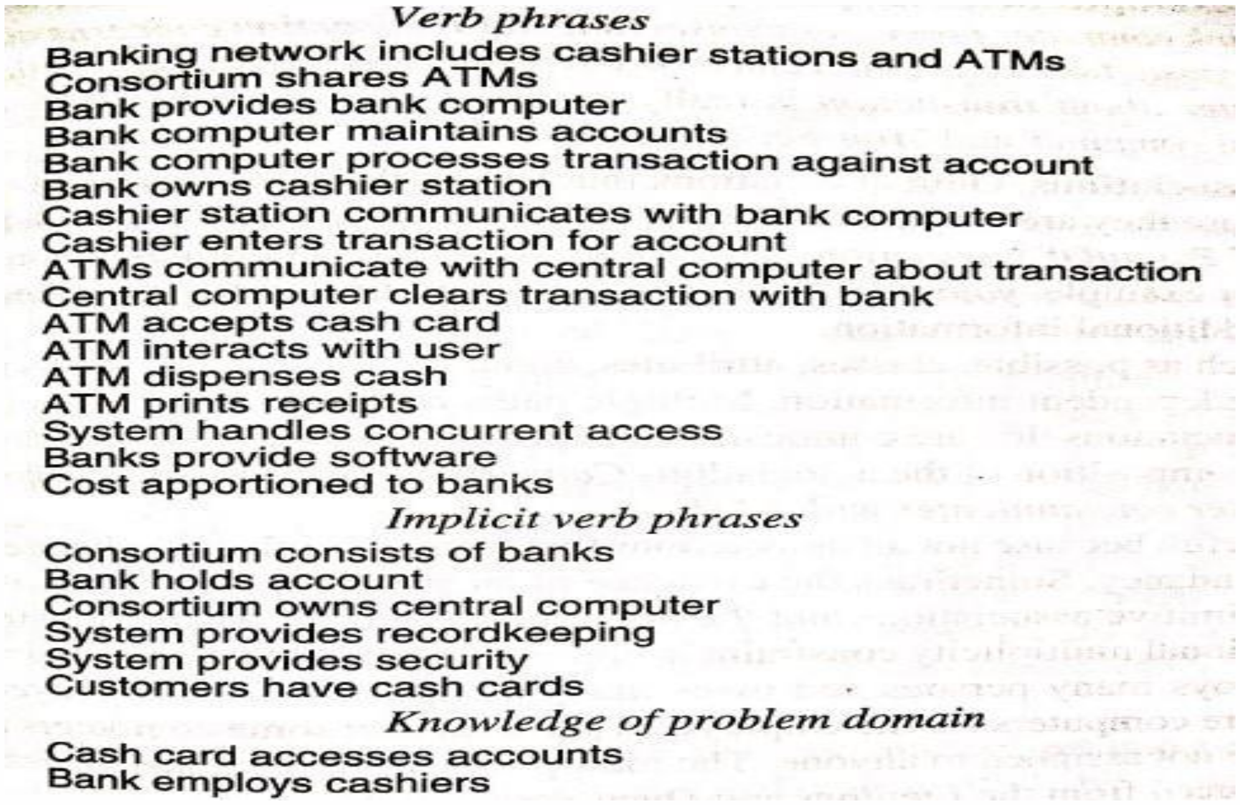
Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

FINDING ASSOCIATIONS

•A structural relationship between two or more classes is an association.
• A reference from one class to another is an association.
• Idea here is to capture relationshipsThey include
  - •Physical location (Next To, PartOf, ContainedIn)
  - •Directed actions (Drives)
  - •Communication (TalksTo)
  - •Ownership (Has, PartOf)
  - •Etc…

Domain Class Model



Verb phrases

Banking network includes cashier stations and ATMs
Consortium shares ATMs
Bank provides bank computer
Bank computer maintains accounts
Bank computer processes transaction against account
Bank owns cashier station
Cashier station communicates with bank computer
Cashier enters transaction for account
ATMs communicate with central computer about transaction
Central computer clears transaction with bank
ATM accepts cash card
ATM interacts with user
ATM dispenses cash
ATM prints receipts
System handles concurrent access
Banks provide software
Cost apportioned to banks

Implicit verb phrases

Consortium consists of banks
Bank holds account
Consortium owns central computer
System provides recordkeeping
System provides security
Customers have cash cards

Knowledge of problem domain

Cash card accesses accounts
Bank employs cashiers

Figure: 4.12 Association from ATM problem staement

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

KEEPING THE RIGHT ASSOCIATIONS

- •Associations between eliminated classes
  - •Eliminate associations between eliminated classes
- •Irrelevant or implementation associations
  - •Eliminate any associations that are outside the problem domain or deal with implementations constructs
- •Actions
  - •An association should describe a structural property of the application domain, not a transient event
  - •Ex. *ATM accepts cash cards* describes part of the interaction cycle between an ATM and a customer
  - •Not a permanent relationship between ATM and cash card

Domain Class Model

- Ternary associations
    - Decompose ternary associations into binary associations
- Derived associations
    - Omit associations that can be defined in terms of other associations, they are redundant
    - Ex. *GrandparentOf* can be defined in terms of *ParentOf* associations
    - Omit the associations defined by conditions on attributes
    - Ex. *YoungerThan*

Domain Class Model

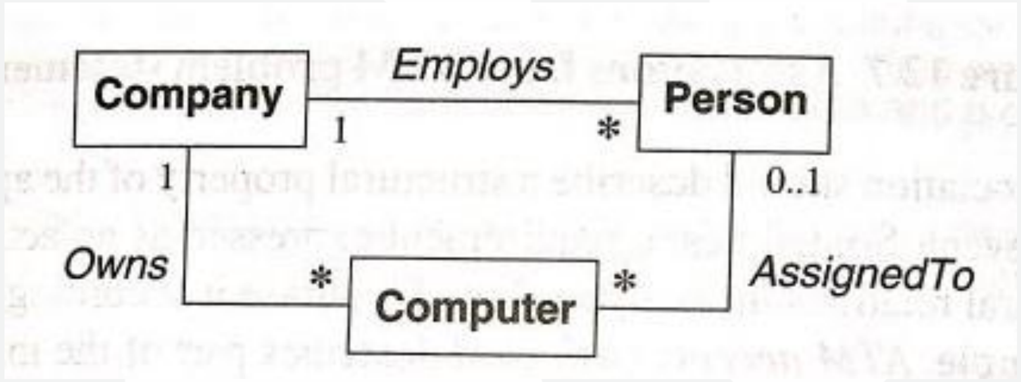


Figure: 4.13 No redundant associations

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

- Misnamed associations
    - Names are important to understanding
    - Should be chosen with great care
- Association end names
    - Add association end names where appropriate
- Qualified associations
    - Usually a name identifies an object within some context
    - Most names are not globally unique
    - The context combines with name to uniquely identify an object
    - A qualifier distinguishes object on “many” side of associations

Domain Class Model

- Multiplicity
  - Don’t put too much effort into getting it right
  - It often changes during analysis
- Missing associations
  - Add any missing associations that are discovered
- Aggregation
  - For some applications, aggregation is relatively minor
  - It can be unclear whether to use aggregation or ordinary association
  - So don’t spend much time trying to distinguish between them

Domain Class Model

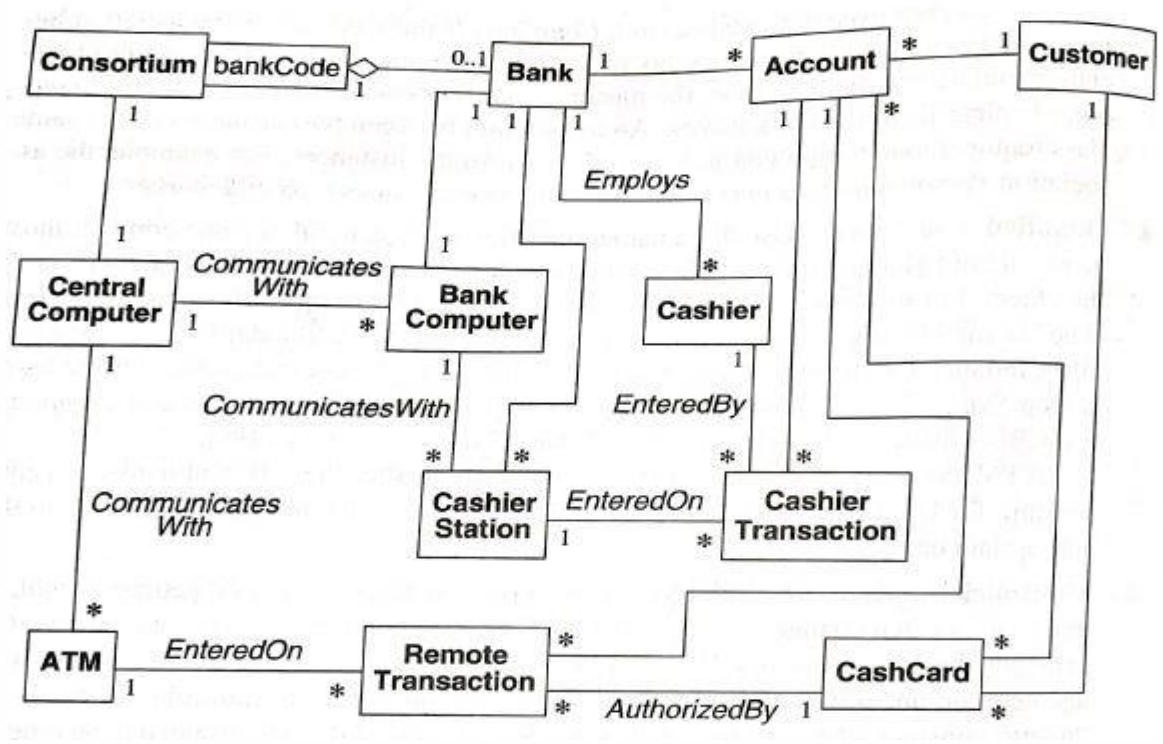


Figure: 4.14 Initial Class Diagram for ATM System

- ATM Example: We decide that a bank is a consortium and indicate the relationship with aggregation
- ATM Example: Figure 4.14 shows a class diagram with the remaining associations. **Only significant association names are included**

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

FINDING ATTRIBUTES

- They are data properties of individual objects
- attribute values should not be objects
- Use an association to show any relationship between two objects
- Attributes are less likely to be defined in the problem statement
- Draw your knowledge of the application domain and the real world to find them
- Only consider attributes directly relevant to the system

Domain Class Model

KEEPING RIGHT ATTRIBUTES

- Eliminate unnecessary and incorrect attributes with the following criteria
- Objects
    - If the independent existence of an element is important, it is an object
    - The distinction often depends on the application
    - Ex. In mailing list *city* might be considered an attribute
    - While in census *City* would be a class with many attributes

Domain Class Model

- Qualifier
  - If the value of an attribute depends in a particular context, then consider restating the attribute as a qualifier
- Name
  - A name is an attribute when its use does not depend on context
  - Especially when it need not be unique within some set

Domain Class Model

- Identifiers
  - Do not include an attribute whose only purpose is to identify an object
  - Object identifiers are implicit in class model
- Attributes on association
  - If a value requires the presence of a link, the property is an attribute of the association
  - Not of a related class

Domain Class Model

- Internal values
  - If an attribute describes the internal state of an object that is invisible outside the object, then eliminate it from the analysis
- Fine detail
  - Omit minor attributes that are unlikely to affect most operations
- Discordant attributes
  - Unrelated attribute to all other attributes may indicate a class that should be split into two distinct classes.
- Boolean attributes
  - Reconsider all boolean attributes.

Domain Class Model

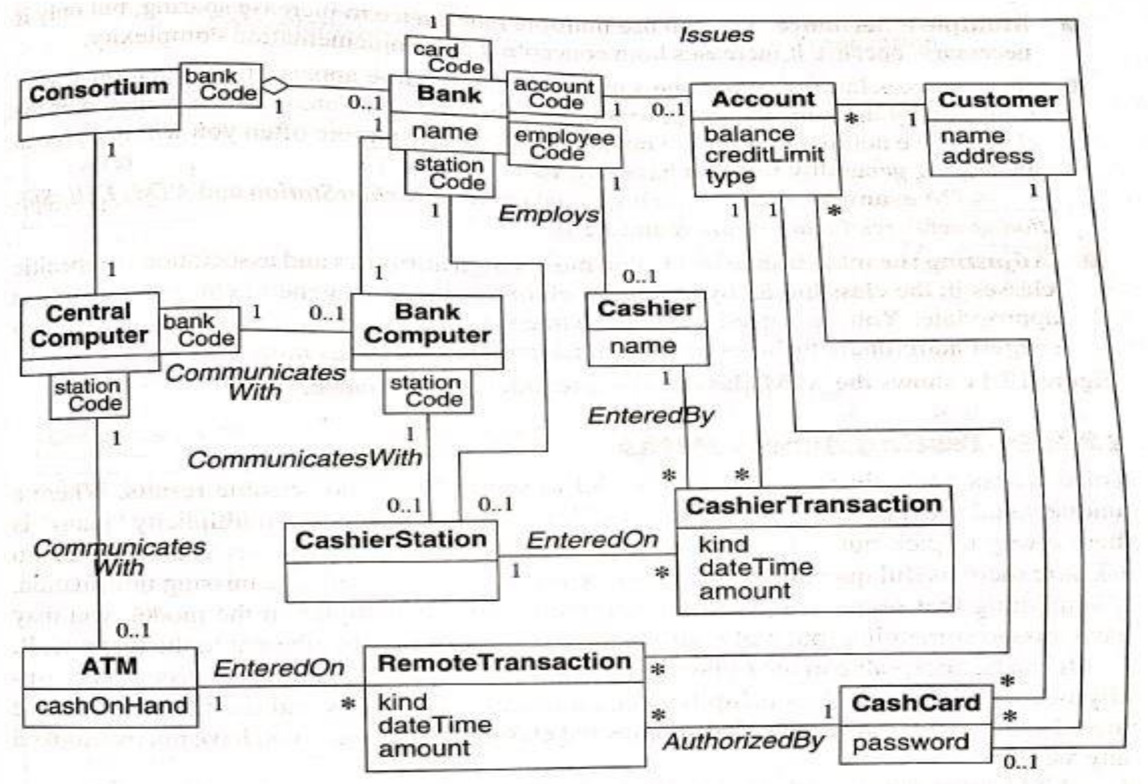


Figure: 4.15 ATM class model with attributes

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Parul® University

DIGITAL LEARNING CONTENT

Domain Class Model

REFINING WITH INHERITANCE

- The next step to organize classes by using inheritance to share common structure
- It can be added in two directions:
  - By generalizing common aspects of existing classes into a super class(bottom up)
  - By specializing existing classes into multiple subclasses (top down)

Domain Class Model

- Bottom-up
  - Discover inheritance from the bottom up by searching for classes with similar attributes, associations, and operations
  - For each generalization, define a superclass to share common features
- Top-down
  - Apparent from the application domain
  - Look for the noun phrases composed of various adjectives on the class name
  - Ex. Fluorescent lamp, incandescent lamp, fixed men, pop-up menu

Parul® University

DIGITAL LEARNING CONTENT

Domain Class Model

- •Multiple inheritance
  - •You can use multiple inheritance to increase sharing, but only if necessary
- •Similar associations
  - •When the same association name appears more than once with substantially the same meaning, try to generalize the associated class
- •Adjusting the inheritance level
  - •Must assign attributes and associations to specific classes in the class hierarchy
  - •Assign each one to the most general class for which it is appropriate

Domain Class Model

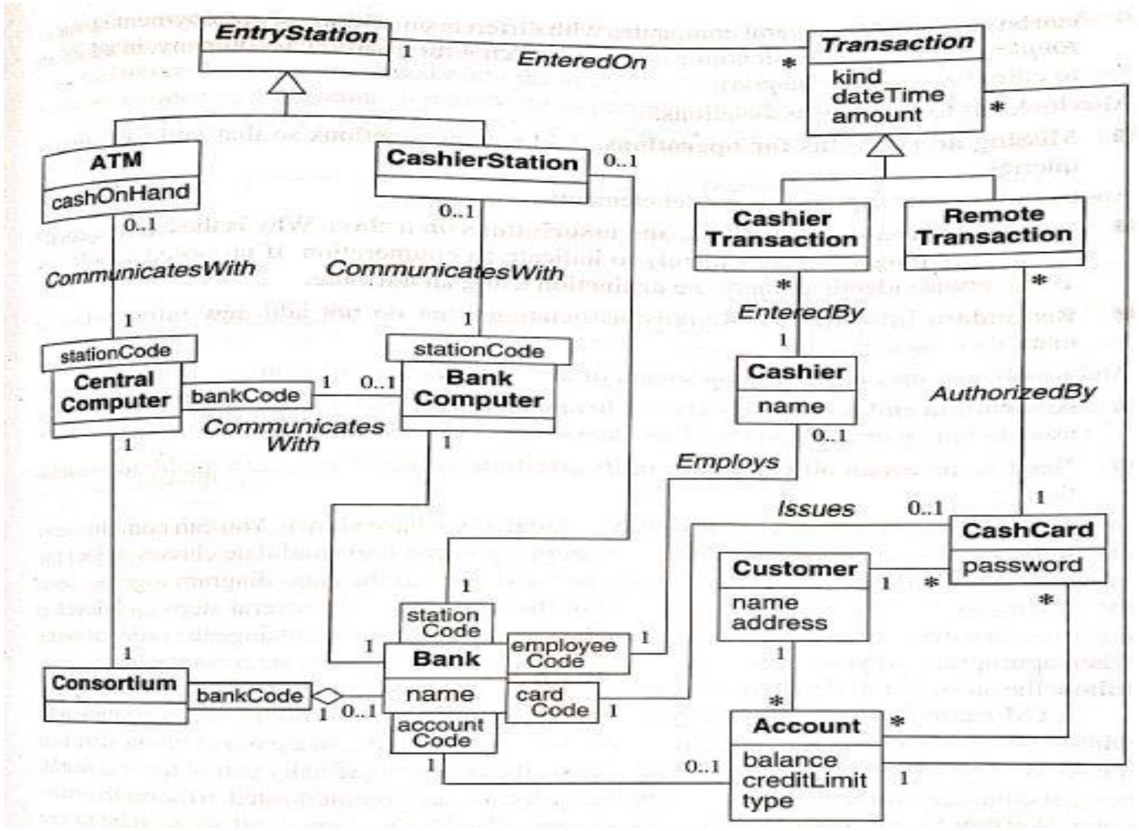


Figure: 4.16 ATM class model with attributes and inheritance

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

TESTING ACCESS PATHS

- Trace the access path through the class model to see if they yield sensible result
- For example,
- Where a unique value is expected, is there a path yielding a unique value???
- Think of the questions you might like to ask
- Are there useful questions that cannot be answered?
- They indicate missing information

Domain Class Model

ITERATING A CLASS MODEL

•A class model is rarely correct after a single pass
•The entire software development process is one of the continual iteration
•Different parts of a model are often at different stages of completion
•If you find any deficiency, go back to an earlier stage if necessary to correct it
•Some refinements can come only after completing the state and the interaction models

Domain Class Model

- Several signs of missing classes
    - Asymmetries in associations and generalizations
    - Difficulty in generalizing clearly
    - Duplicate associations with the same name and purpose
    - Missing access path for operations
    - Lack of attributes, operations, and associations on a class
    - Redundant information, etc…

Domain Class Model

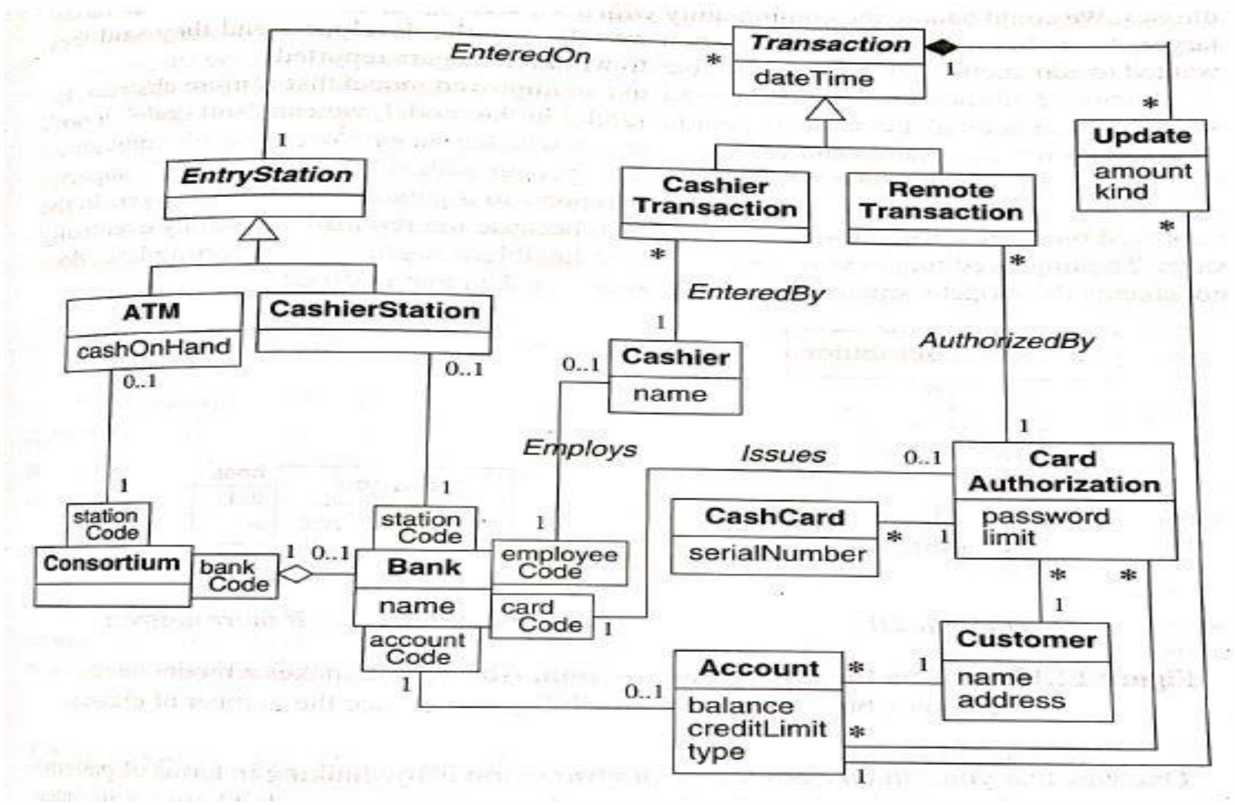


Figure: 4.17 ATM class model after further revision

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

SHIFTING THE LEVEL OF ABSTRACTION

•Previous analysis we done is not always suffice
•Sometimes you must raise level of abstraction to solve a problem
•For example,

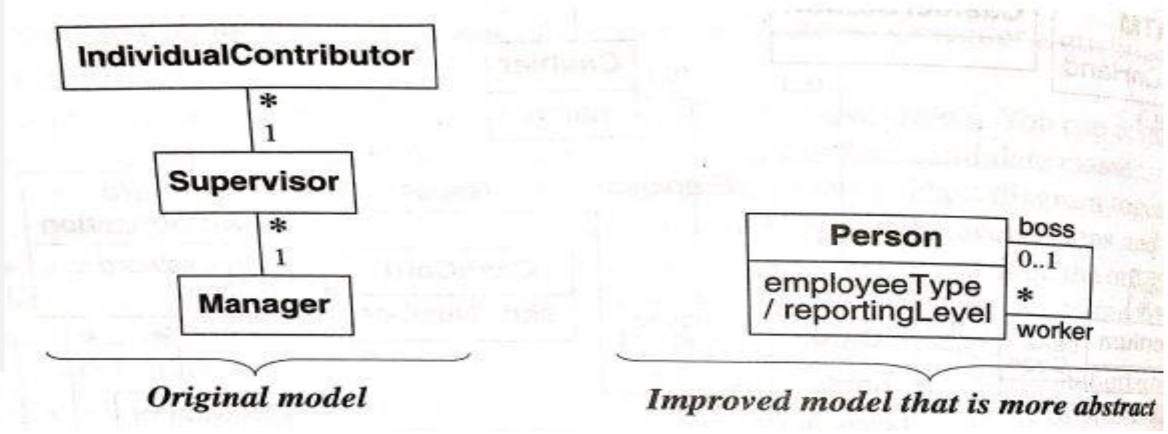


Figure: 4.18 Shifting the level of abstraction

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain Class Model

GROUPING CLASSES IN PACKAGES

•A package is a group of elements with a common theme
•Elements may be classes, associations, generalizations and lesser packages
•Packages organize a model for convenience in drawing, printing and viewing
•Classes in the same package are more closely related than classes in different packages

Domain State Model

•Some domain objects pass through qualitatively distinct states during their lifetime

•There may be different constraints on attribute values, different associations or multiplicities in the various states

•Different operations that may be invoked

•Different behavior of the operations

Domain State Model

•The state diagram describes the various states the object can assume
•The properties and constraints of the objects in various states
•And events that take an object from one state to another state
•Most domain classes do not require state diagrams
•For minority of classes that do exhibit distinct states, a state model can help in understanding their behavior

Domain State Model

Steps performed in constructing a domain state model

- Identify domain classes with states
- Find states
- Find events
- Build state diagrams
- Evaluate state diagrams

Domain State Model

IDENTIFYING CLASSES WITH STATES

•Examine list of domain classes which have distinct life cycle
•Look for the classes that can be characterized by a progressive history
•Identify the significant states in the life cycle of the object

Domain State Model

FINDING STATES

•List the states for each class
•Characterize the objects in each class
•Give each state a meaningful name
•It is necessary to determine all the states before examining events
•By looking at events and considering transitions among states, missing states will become clear

Domain State Model

FINDING EVENTS

•Find events that cause transitions among states
•Think about the stimuli that cause a state to change
•In many cases, you can regard an event as completing a do-activity

Domain State Model

BUILDING STATE DIAGRAMS

•Note the states to which each event applies
•Add transitions to show the change in states caused by occurrence of an event
•If an event terminates a state, it will usually have a single transition from the state to another state
•If an event initiates a target state, then consider where it occur,and add transitions from those states to the target states

Domain State Model

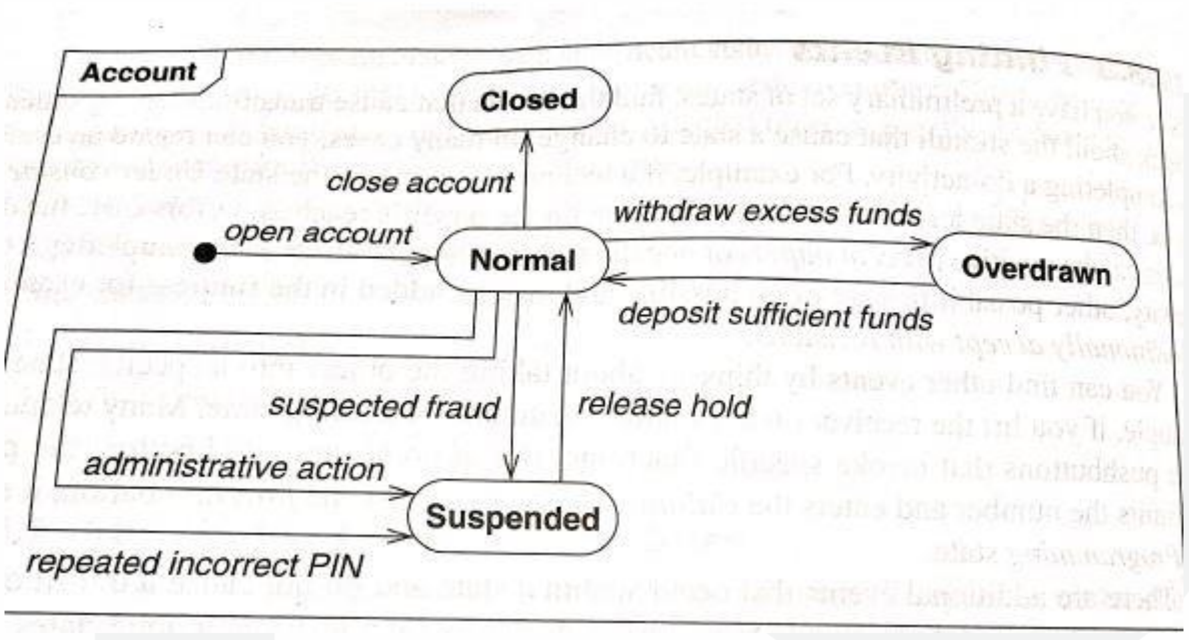


Figure 4.19: Domain state model for accounts class

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Domain State Model

EVALUATING STATE DIAGRAMS

- Examine each state model
- Are all the states connected???
- Pay particular attention to paths through it
- If it represents a progressive class, is there a path from the initial state to the final state??
- Are the expected variations present??
- Etc…

Domain Interaction Model

- The interaction model is seldom important for domain analysis
- During domain analysis the emphasis is on key concepts and deep structural relationships
- Not the user’s view of them
- The interaction model is an important aspect of application modeling

Domain Interaction Model

ITERATING THE ANALYSIS

•Most analysis models require more than one pass to complete
•Problem statements often contain circularities
•Most applications cannot be approached in completely linear way
•To understand a problem with all implications, you must attack the analysis iteratively

Domain Interaction Model

STEPS:

Refining the analysis model
Restating the requirements
Analysis and design

Parul® University

DIGITAL LEARNING CONTENT

APPLICATION ANALYSIS

Application Interaction Model

•After completing the domain model, we shift out attention to the details of an application and consider interaction

Application Interaction Model

- **Steps for constructing application interaction model**
    - Determine the system boundary
    - Find actors
    - Find use cases
    - Find initial and final events
    - Prepare normal scenarios
    - Add variation and exception scenarios
    - Find external events
    - Prepare activity diagrams for complex use cases
    - Organize actors and use cases
    - Check against the domain class model

Application Interaction Model

DETERMINING THE SYSTEM BOUNDARY

•You must know the scope of the application in order to specify functionality
•You must decide what the system includes and what it omits
•During analysis, you determine the purpose of the system and the view that it presents to its actors

Application Interaction Model

FINDING ACTORS

•After defining the system boundary, you must identify the external objects that directly interact with the system

•Actors are not under control of the application

•Each actor represents idealized user that exercises some subset of the system functionalities

Application Interaction Model

FINDING USE CASES

•For each actor, list the fundamentally different ways in which the actor uses the system

•Each of these ways is a use case

•They partition the functionality of a system into small number of discrete units

Application Interaction Model

Initiate Session: The ATM establishes the identity of the user and makes available a list of accounts and actions

Query Account: The system provides general data for an account, such as current balance, date of last transaction, and date of mailing for last statement

Process Transaction: The ATM system performs an action that affects ad account's balance , such as deposit, withdraw, and transfer. The ATM ensures that all completed transactions are ultimately written to the bank's database

Transmit Data: The ATM uses the consortium's facilities to communicate with the appropriate bank computers.

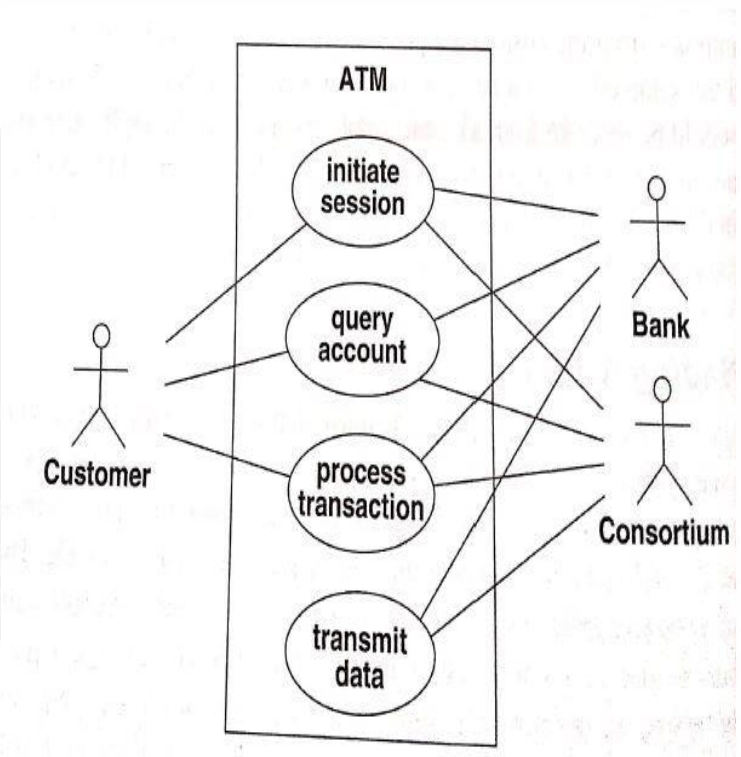


Figure 4.20 Use case diagram for the ATM. Bullets summarize themes.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

FINDING INITIAL AND FINAL STATES

•Use cases do not show the behavior clearly

•To understand it, you must understand the execution sequence

•You can start by finding the events that initiate each use cases that cover each use case

•Determine which actor will initiate the use case

•Define the event that it sends to the system

•You should also determine the final event or events and how to include in each use case

Application Interaction Model

- ATM Example. Here are initial and final events for each use case
    - **Initiate Session:**
        - Initial Event: Customer insertion of a card.
        - Final Event: system keeps the card or returns the card
    - **Query Account:**
        - Initial Event: customer's request for account data
        - Final Event: system's delivery of final data
    - **Process Transaction:**
        - Initial Event: customer's initiation of a transaction
        - Final Event: committing or aborting the transaction
    - **Transmit Data:**
        - Initial Event: customers request for account data or recovery from network, power or another kind of failure

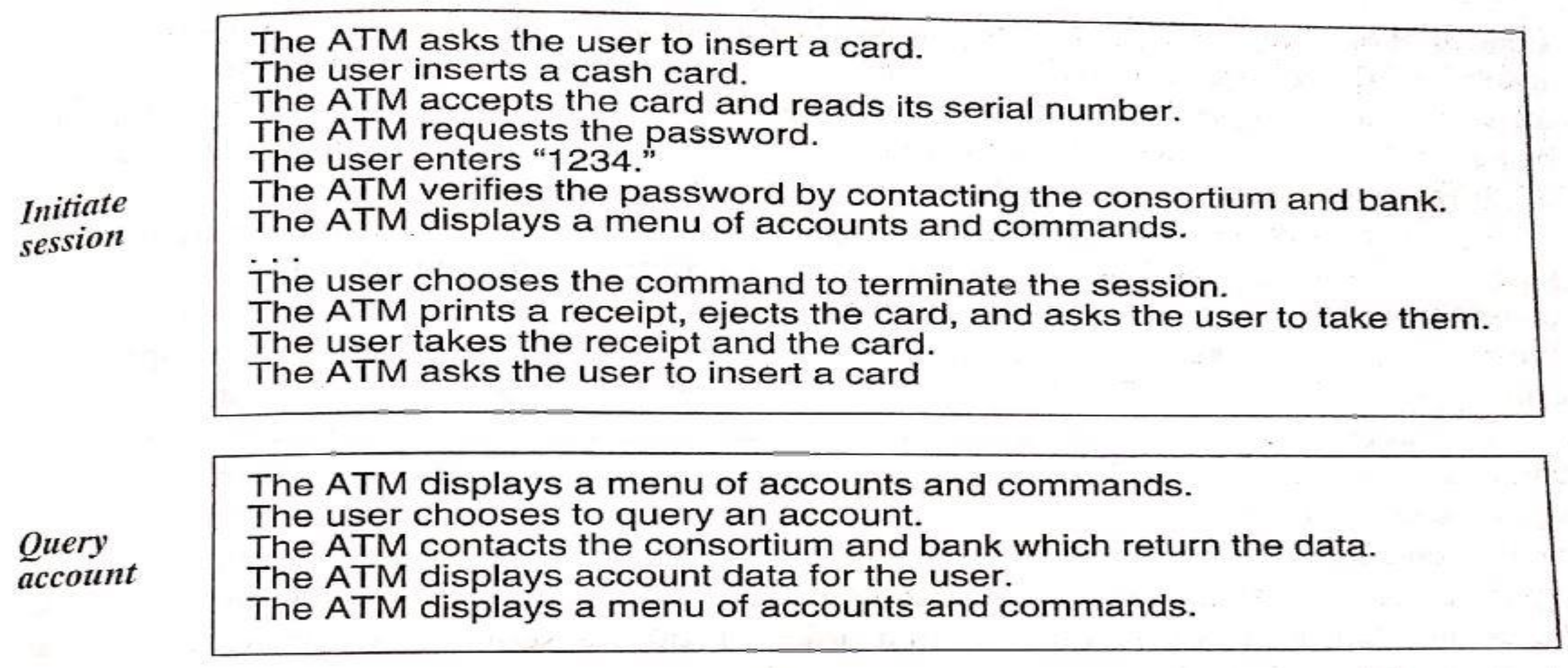
          Final Event: successful transmission of data.

Application Interaction Model

PREPARING NORMAL SCENARIOS

•A scenario is a sequence of events among a set of interacting objects
•Think in terms of sample interactions, rather than trying to write down general case directly

Application Interaction Model



Initiate session

The ATM asks the user to insert a card.
The user inserts a cash card.
The ATM accepts the card and reads its serial number.
The ATM requests the password.
The user enters "1234."
The ATM verifies the password by contacting the consortium and bank.
The ATM displays a menu of accounts and commands.
. . .
The user chooses the command to terminate the session.
The ATM prints a receipt, ejects the card, and asks the user to take them.
The user takes the receipt and the card.
The ATM asks the user to insert a card

Query account

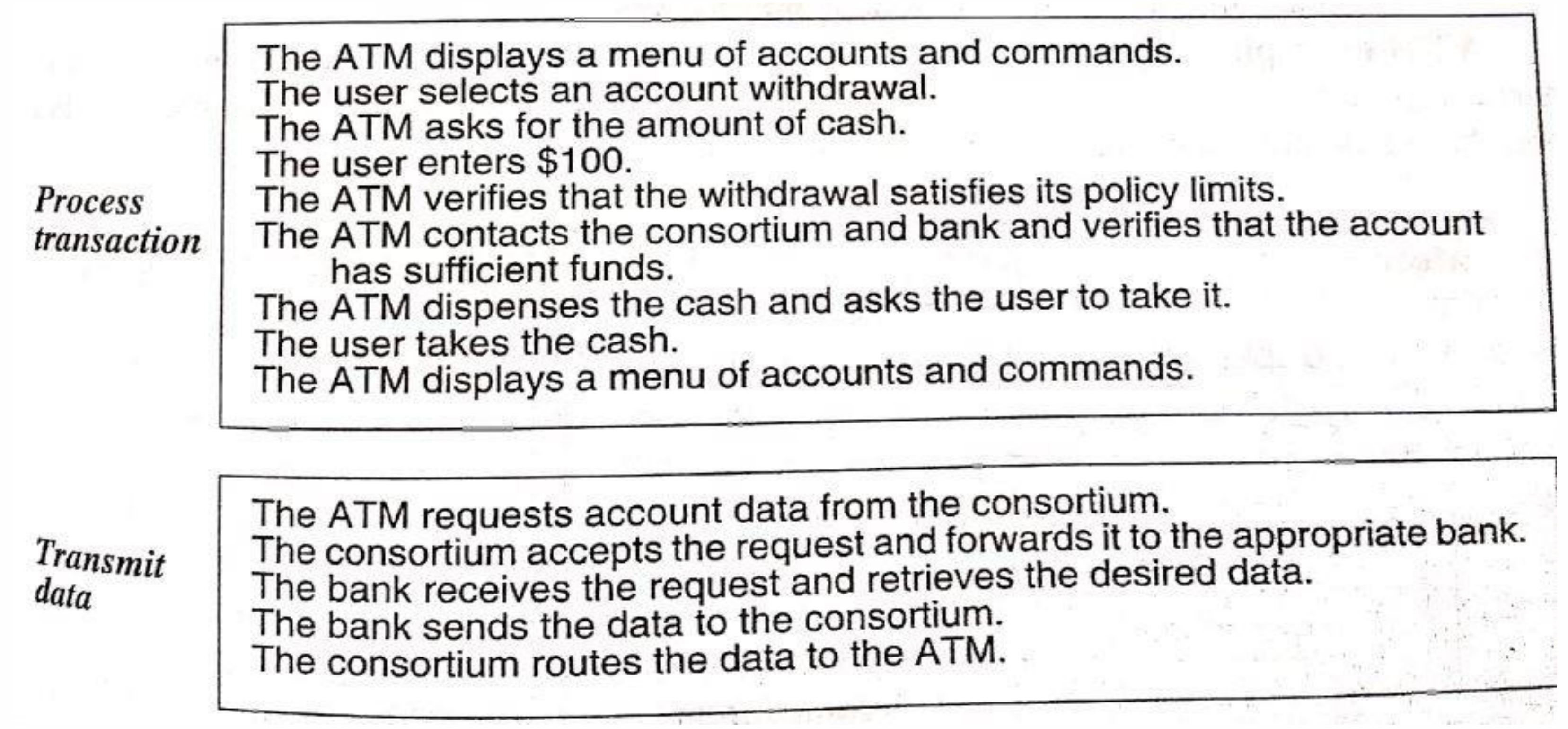
The ATM displays a menu of accounts and commands.
The user chooses to query an account.
The ATM contacts the consortium and bank which return the data.
The ATM displays account data for the user.
The ATM displays a menu of accounts and commands.

Figure 4.21.1: Normal ATM Scenarios.

*Prepare one or more scenarios for each use case.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model



Process transaction

The ATM displays a menu of accounts and commands.
The user selects an account withdrawal.
The ATM asks for the amount of cash.
The user enters $100.
The ATM verifies that the withdrawal satisfies its policy limits.
The ATM contacts the consortium and bank and verifies that the account has sufficient funds.
The ATM dispenses the cash and asks the user to take it.
The user takes the cash.
The ATM displays a menu of accounts and commands.

Transmit data

The ATM requests account data from the consortium.
The consortium accepts the request and forwards it to the appropriate bank.
The bank receives the request and retrieves the desired data.
The bank sends the data to the consortium.
The consortium routes the data to the ATM.

Figure 4.21.2: Normal ATM Scenarios.

*Prepare one or more scenarios for each use case.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

ADDING VARIATION AND EXECPTION SCENERIO

- After preparing the typical scenarios, consider special cases
- Such as omitted input, maximum and minimum values and repeated values
- Then consider error cases, including invalid values and failure to respond
    - ATM Example:
        - ATM Can't read the card
        - Card has expired
        - ATM times out
        - Amount is invalid
        - Machine is out of cash
        - Communication lines are down
        - Transaction is rejected because of suspicious pattern of card usage

Application Interaction Model

FINDING EXTERNAL EVENTS

•Examine the scenarios to find all external events
•Include all inputs, decisions, interrupts and interactions to or from users or external devices
•An event can trigger effects for a target object
•Internal computation steps are not events, except for computations that interact with the external world
•A transmittal to an object is an event
•Group together under a same name events that have the same effect on flow of control

Application Interaction Model

PREPARING SEQUENCE DIAGRAM FOR EACH SCENARIOS

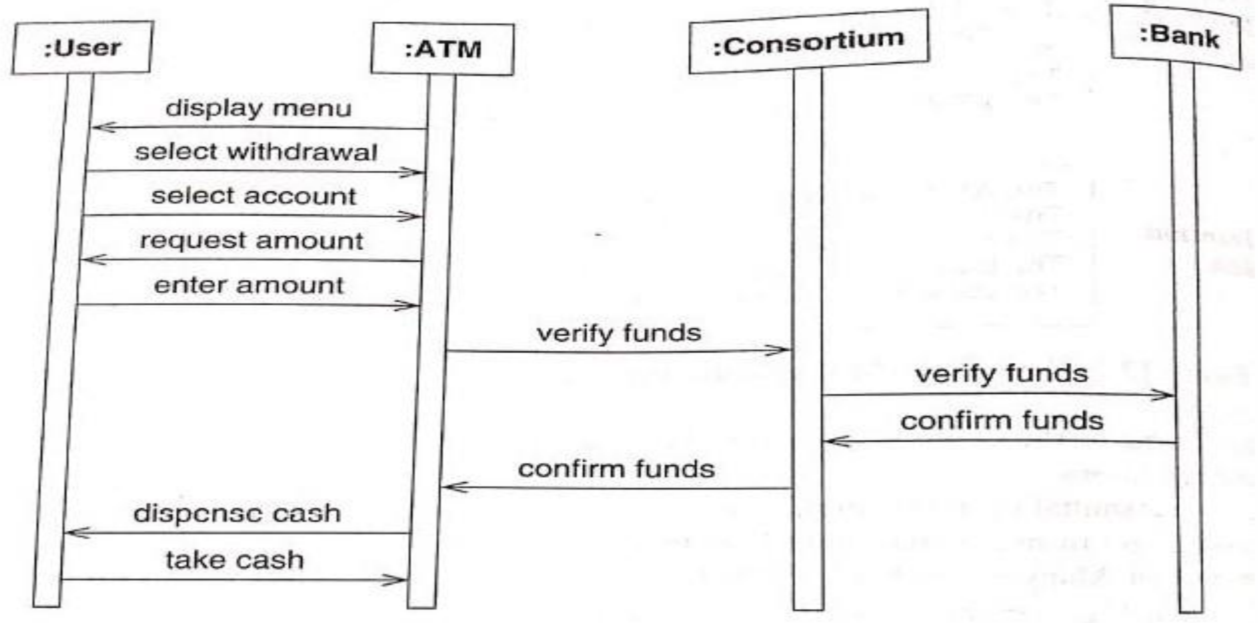


Figure: 4.22 Sequence diagram for the process transaction scenario.
* A sequence diagram clearly shows the sender and receiver of each event

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

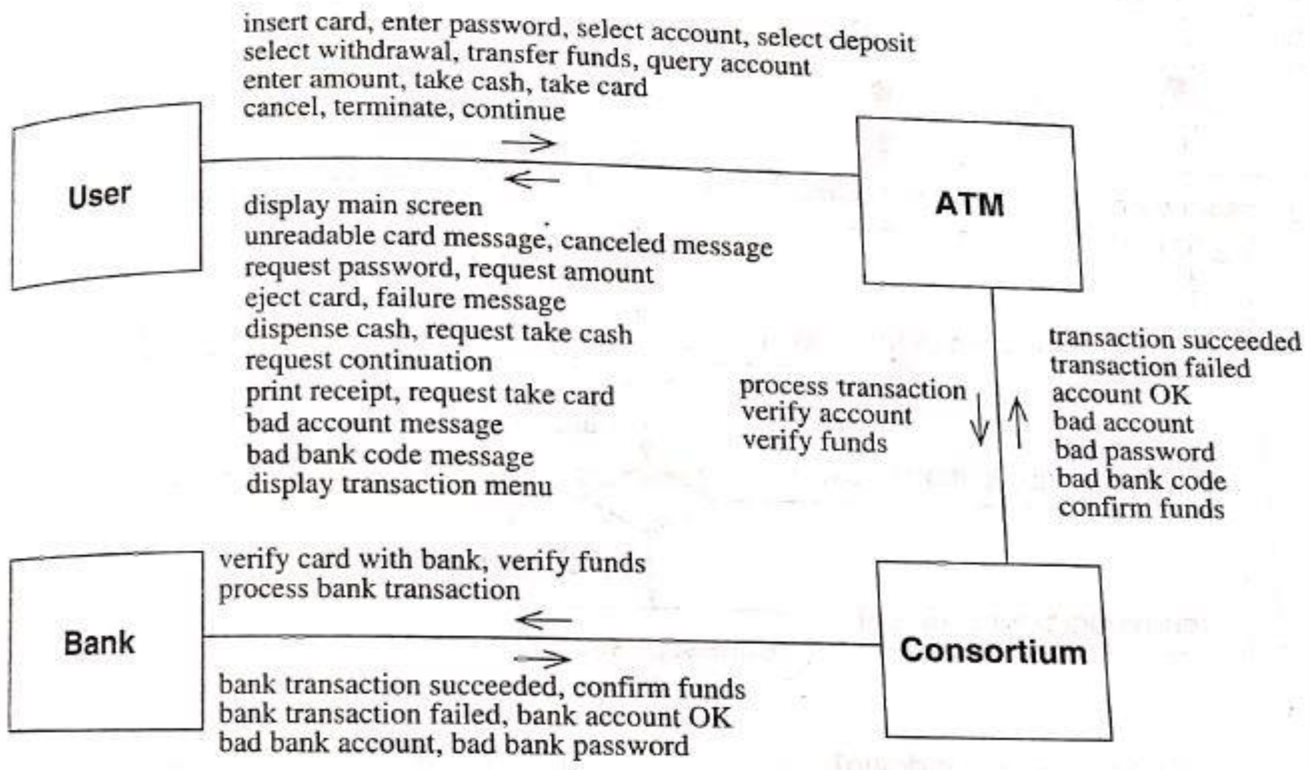

insert card, enter password, select account, select deposit
select withdrawal, transfer funds, query account
enter amount, take cash, take card
cancel, terminate, continue
User
ATM
display main screen
unreadable card message, canceled message
request password, request amount
eject card, failure message
dispense cash, request take cash
request continuation
print receipt, request take card
bad account message
bad bank code message
display transaction menu
process transaction
verify account
verify funds
transaction succeeded
transaction failed
account OK
bad account
bad password
bad bank code
confirm funds
verify card with bank, verify funds
process bank transaction
Bank
Consortium
bank transaction succeeded, confirm funds
bank transaction failed, bank account OK
bad bank account, bad bank password

Figure: 4.23 Events for the ATM case study.

* Tally the events in the scenario and note the classes that send and receive each event

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

PREPARING ACTIVITY DIAGRAMS FOR COMPLEX USE CASES

•Sequence diagrams capture dialog and interplay between the actors
•They do not clearly shows the decisions and alternatives
•Activity diagrams let you consolidate all this behavior by documenting forks and merges in the control flow

Application Interaction Model

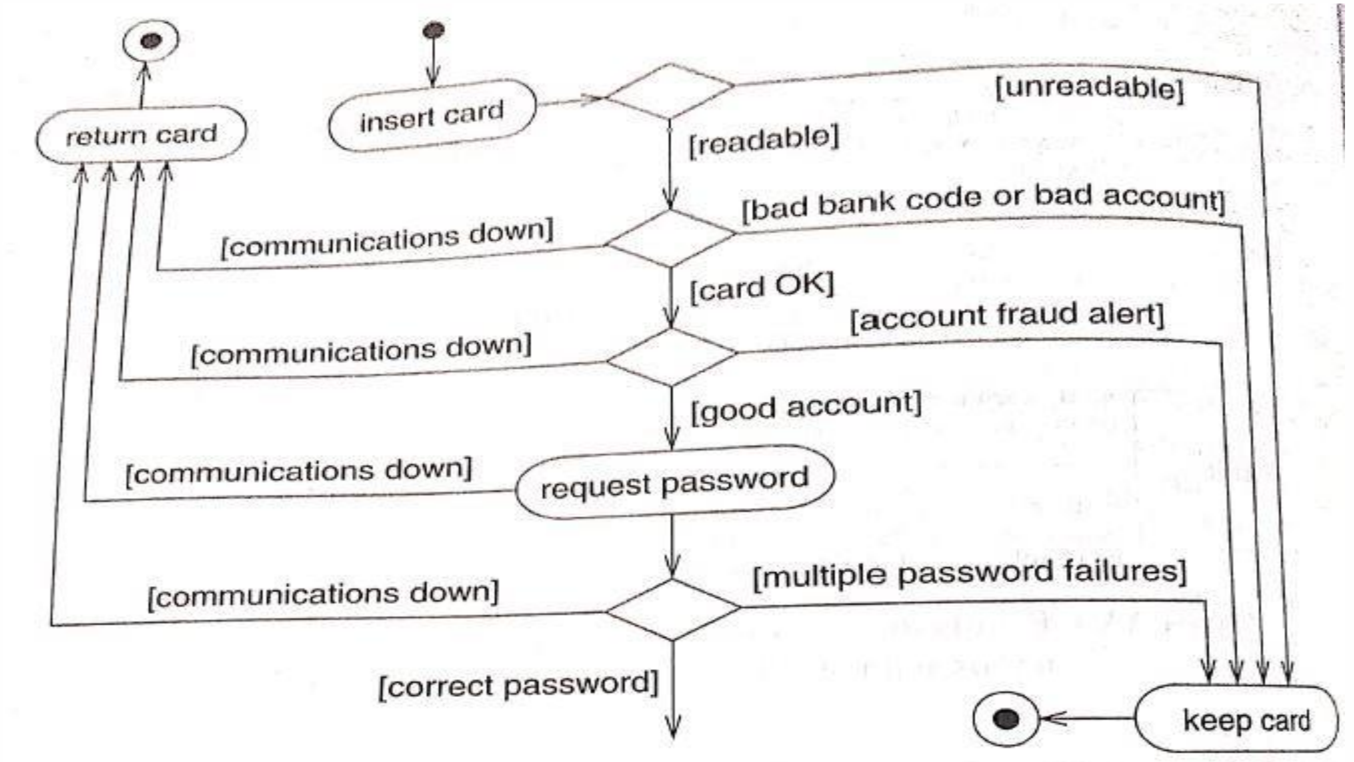


Figure 4.24 Activity diagram for card verification.

* You can use activity diagrams to document business logic, but do not use them as an excuse to begin premature implementation

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

ORGANIZING ACTORS AND USE CASES

•The next step is to organize use cases with relationships (include, extend and generalization)

•This is especially helpful for large and complex systems

Parul® University

DIGITAL LEARNING CONTENT

Application Interaction Model

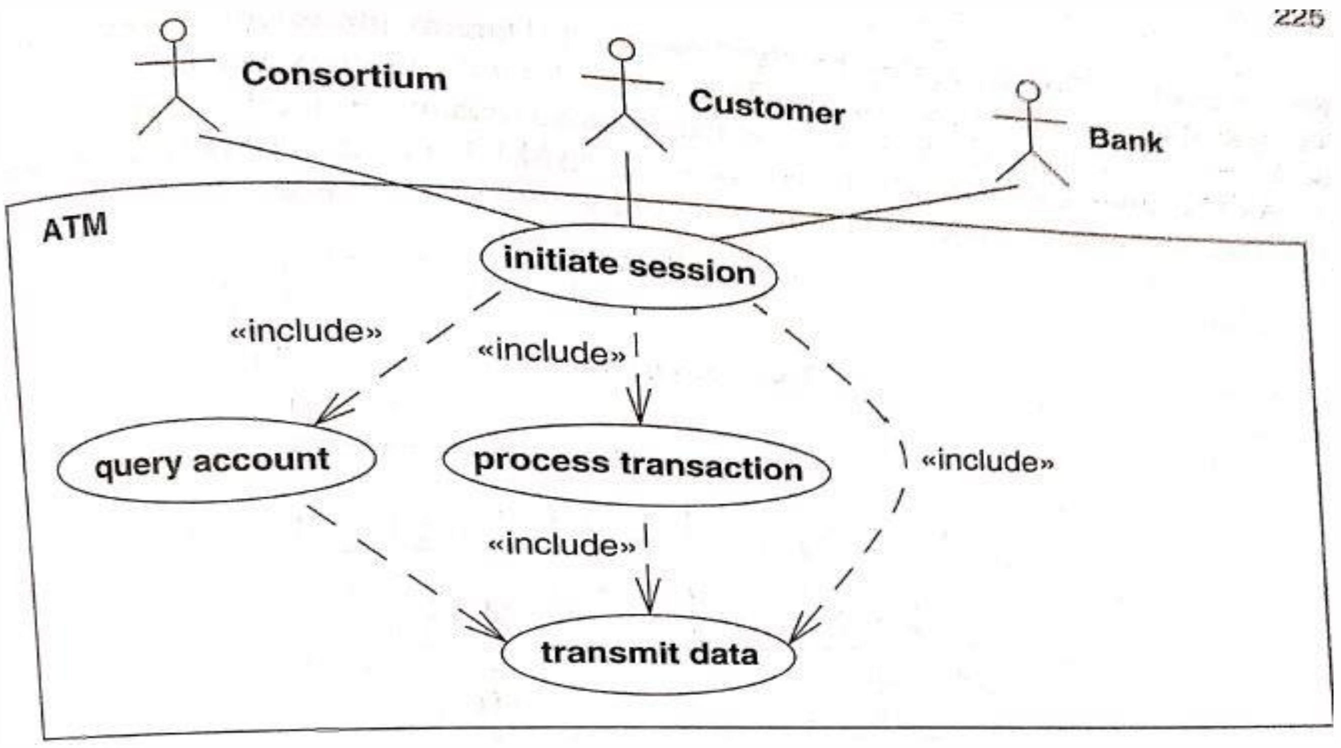


Figure: 4.25 Organizing use cases.

* Once the basic use cases are identified, you can organize them with relationships.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Interaction Model

CHECKING AGAINST THE DOMAIN CLASS MODEL

•The application and domain models should be mostly consistent
•The actors, use cases and scenarios are all based on classes and concepts from the domain model
•Cross check the application and domain models to ensure that there are no inconsistencies
•Examine the scenarios and make sure that domain model has all necessary data

Application Class Model

•Application classes define application itself, rather than real world objects that the application acts on

•Most applications classes are computer-oriented and define the way that users perceive the application

Application Class Model

STEPS TO CONSTRUCT APPLICATION CLASS MODEL

- Specify user interfaces
- Define boundary classes
- Determine controllers
- Check against the interaction model

Application Class Model

SPECIFYING USER INTERFACES

- Most interactions can be separated into two ways
    - Application logic
    - User interface
- A user interface is an object or group of objects that provides the user of a system with coherent way to access its domain objects, commands and application options
- During analysis the emphasis is on the information flow and control rather than presentation

Application Class Model

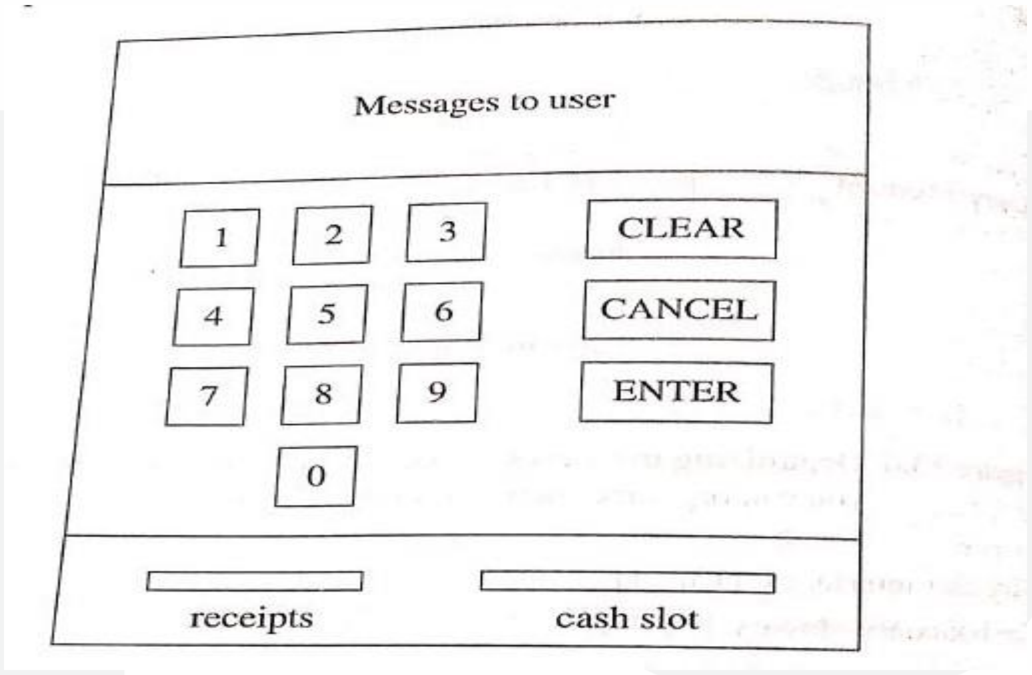


Figure: 4.26 Format of an ATM interface.
* Sometimes a sample interface can help you visualize the operation of an application.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application Class Model

DEFINING BOUNDARY CLASSES

•A system must be able to operate with and accept information from external sources

•But it should have its internal structure dictated by them

•A boundary class is a class that provides a staging area for communications between a system and an external source

•A boundary class understands the format of one or more external sources and converts information for transmission to and from the internal system

Application Class Model

DETERMINING CONTROLLERS

•A controller is an active object that manages control within an application
•It receives signals from the outside world or from objects within the system, reacts to them, invokes operations and send signals to the outside world
•Most of the work in designing a controller is in modeling its state diagram
•In application class model, you should capture the existence of the controller in a system, the control information that each one maintains, the associations from the controllers to the other objects

Application Class Model

CHECKING AGAINST THE INTERACTION MODEL

•As you build the application class model, go over the use cases and think about how they would work

Application Class Model

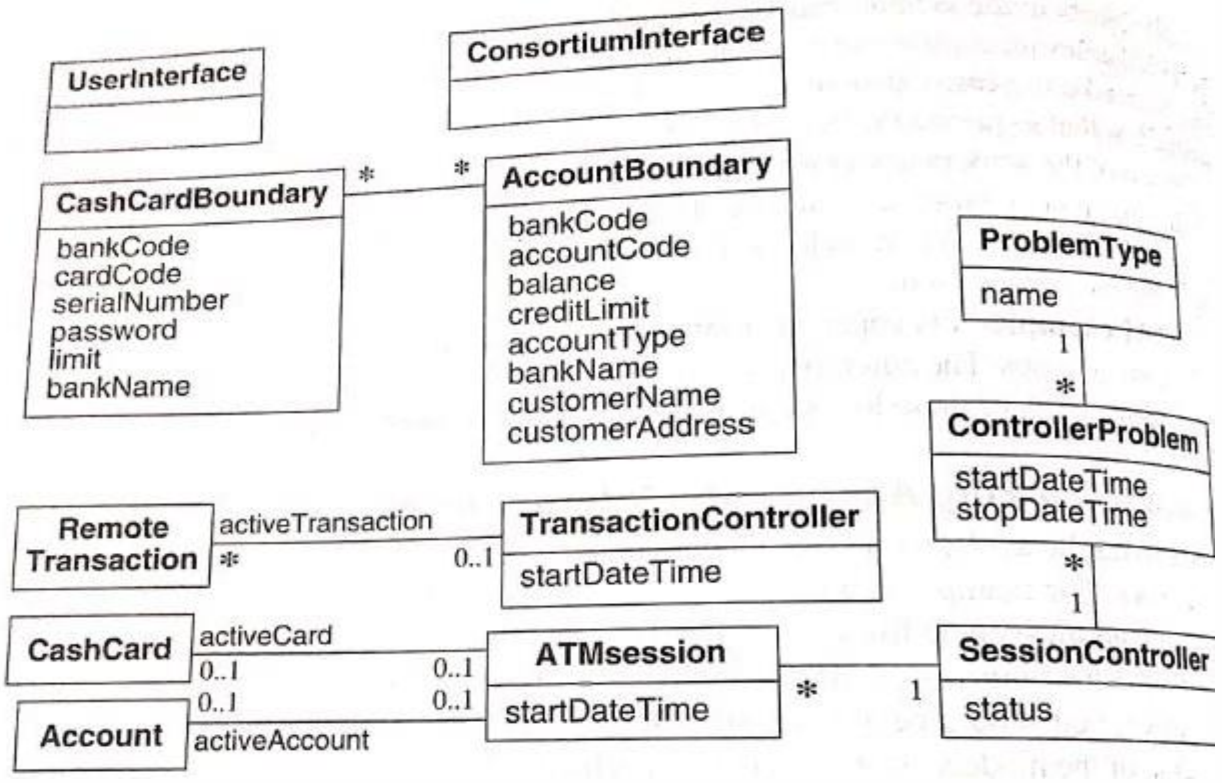


Figure: 4.27 ATM application class model.

* Application classes augment the domain classes and are necessary for development

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application State Model

- Focuses on application classes and arguments the domain state model
- Application classes are more likely to have important temporal behavior than domain classes
- Steps for constructing application state model:
    - Determine application classes with states
    - Find events

Application State Model

•Build state diagrams
•Check against other state diagrams
•Check against the class model
•Check against the interaction model

Application State Model

DETERMINING APPLICATION CLASSES WITH STATES

•The application class model adds computer-oriented classes that are important prominent to users
•Also important to the operation of an application
•Consider each class and determine which class has multiple states
•User interface classes and controller classes are good candidates for state models

Application State Model

FIDING EVENTS

•Study the scenario, prepared previously, and extract events
•They will highlight the major events
•With domain state model, first we find states and then we find events
•Because the domain model focuses on data
•Significant grouping of data form states that are subject to events

Parul® University

DIGITAL LEARNING CONTENT

Application State Model

•With the application state model, first we find events and then we determine states

•Because the focus is on behavior

•Use cases are elaborated with scenarios that reveal events

Application State Model

BUILDING STATE DIAGRAMS

•Build state diagram for each application class with temporal behavior

Application State Model

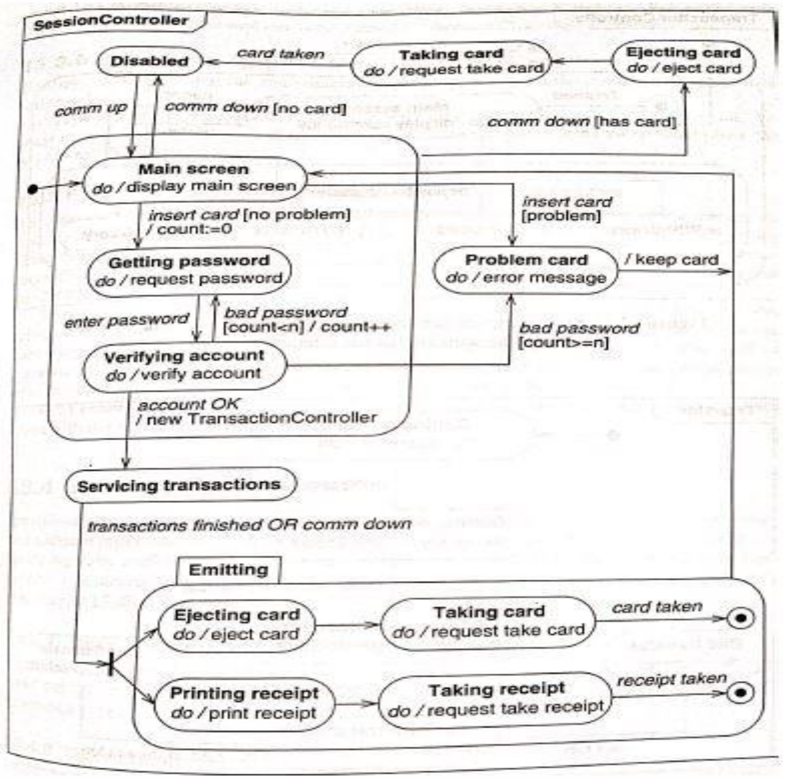


Figure: 4.28 State diagram for SessionController.

* Build a state diagram for each application class with temporal behaviour

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application State Model

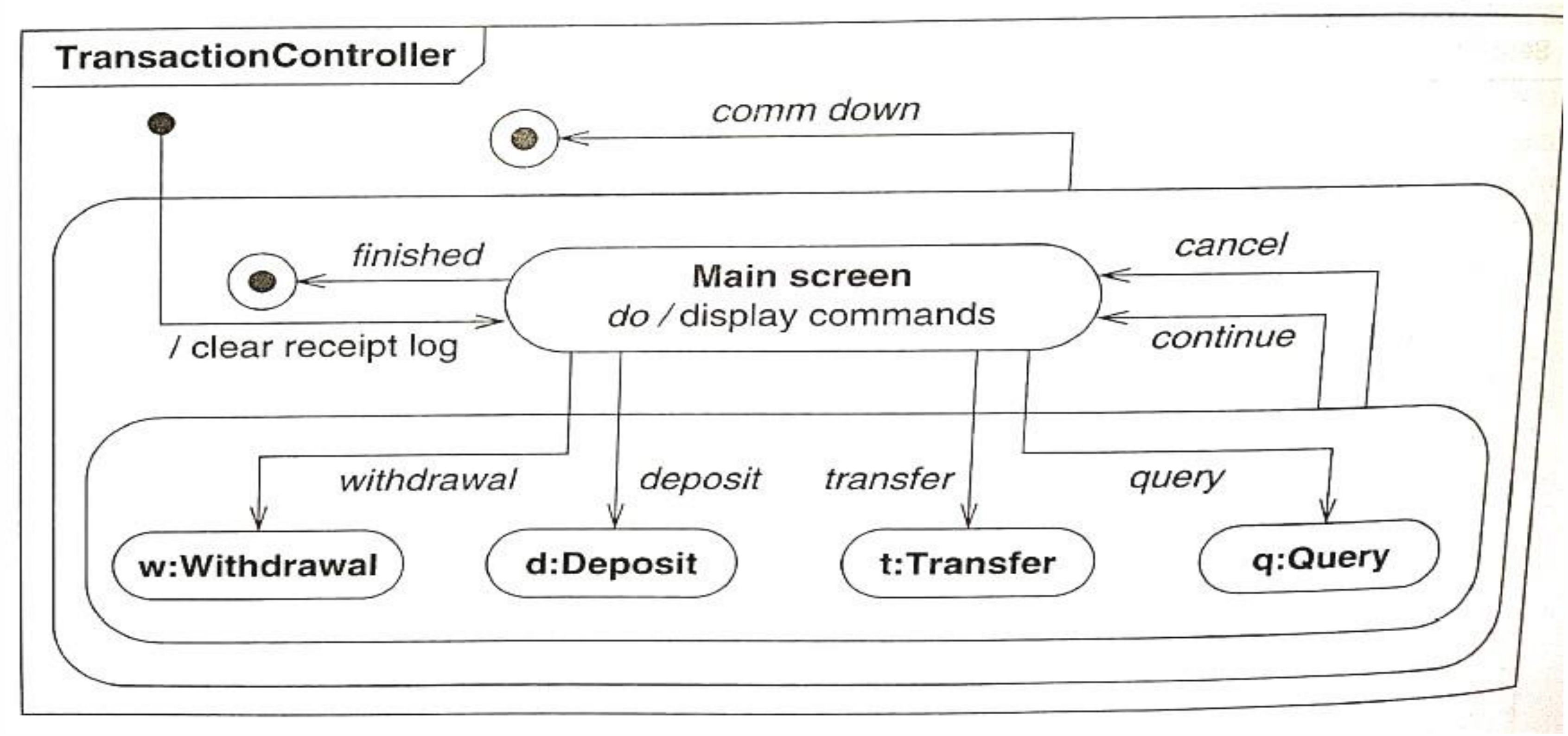


Figure: 4.29 State diagram for TransactionController.

* Obtain information from the scenario of the interaction model.

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application State Model

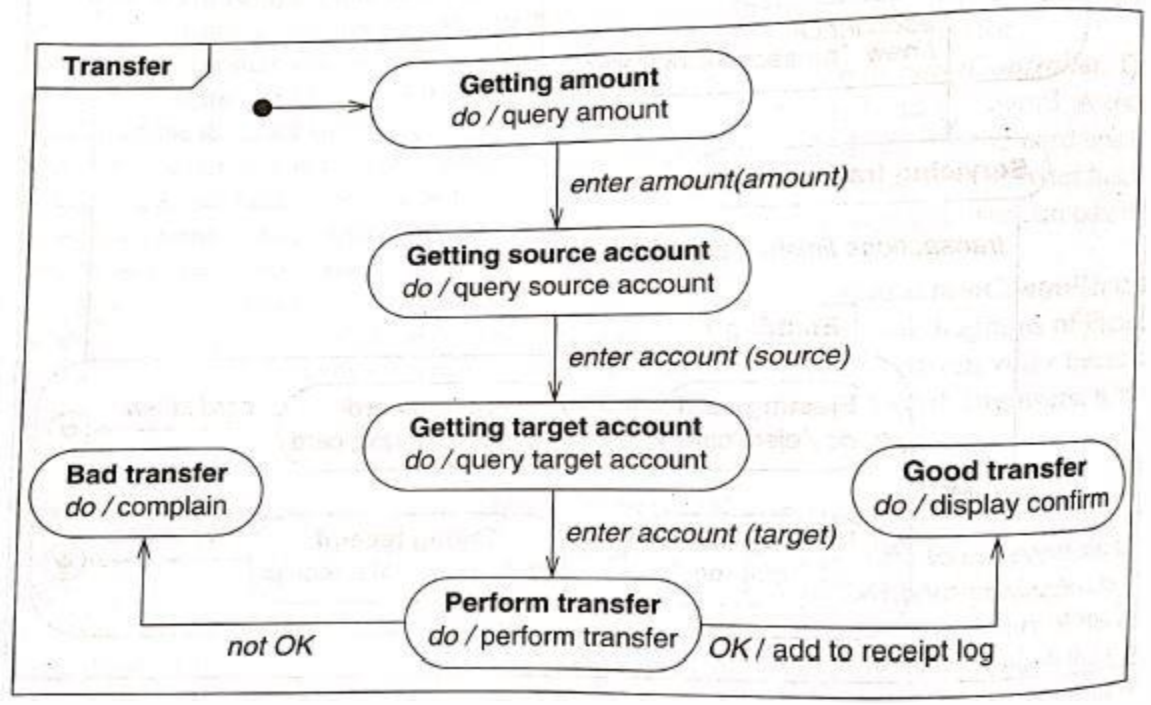


Figure: 4.30 State diagram for Transfer.
* This diagram elaborates the Transfer state in Figure 4.29

Image source : Book: OOMD with UML by Blaha/Rumbaugh

Application State Model

CHECKING AGAINST OTHER STATE DIAGRAMS

- Check the state diagrams for each class for completeness and consistency
  - ATM Example:
    - The SessionController initiates the TransactionController, and the termination of the TransactionController causes the Session Controller to resume

Parul® University

DIGITAL LEARNING CONTENT

Application State Model

CHECKING AGAINST CLASS MODEL

•Make sure that state diagrams are consistent with the domain and application class model

Application State Model

CHECKING AGAINST INTERACTION MODEL

•When state model is ready, go back and check it against the scenarios of the interaction model

•Verify that the state diagram gives the correct behavior

Application State Model

ADDING OPERATIONS

•Operations from the class model
•Operation from use cases
•Shopping-list operations
    •Sometimes real-world behavior of classes suggest operations
    •These operations are not dependent on a particular application but are meaningful in their own weight
    •Ex. Account.close()
    •Bank.createCashcardAuth(customer)

Application State Model

- Simplifying operations
    - Examine the class model for similar operations and variations in form on a single operation
    - Use inheritance where possible to reduce the number of operations
    - Locate each operation at correct level

Parul® University

DIGITAL LEARNING CONTENT

Application State Model

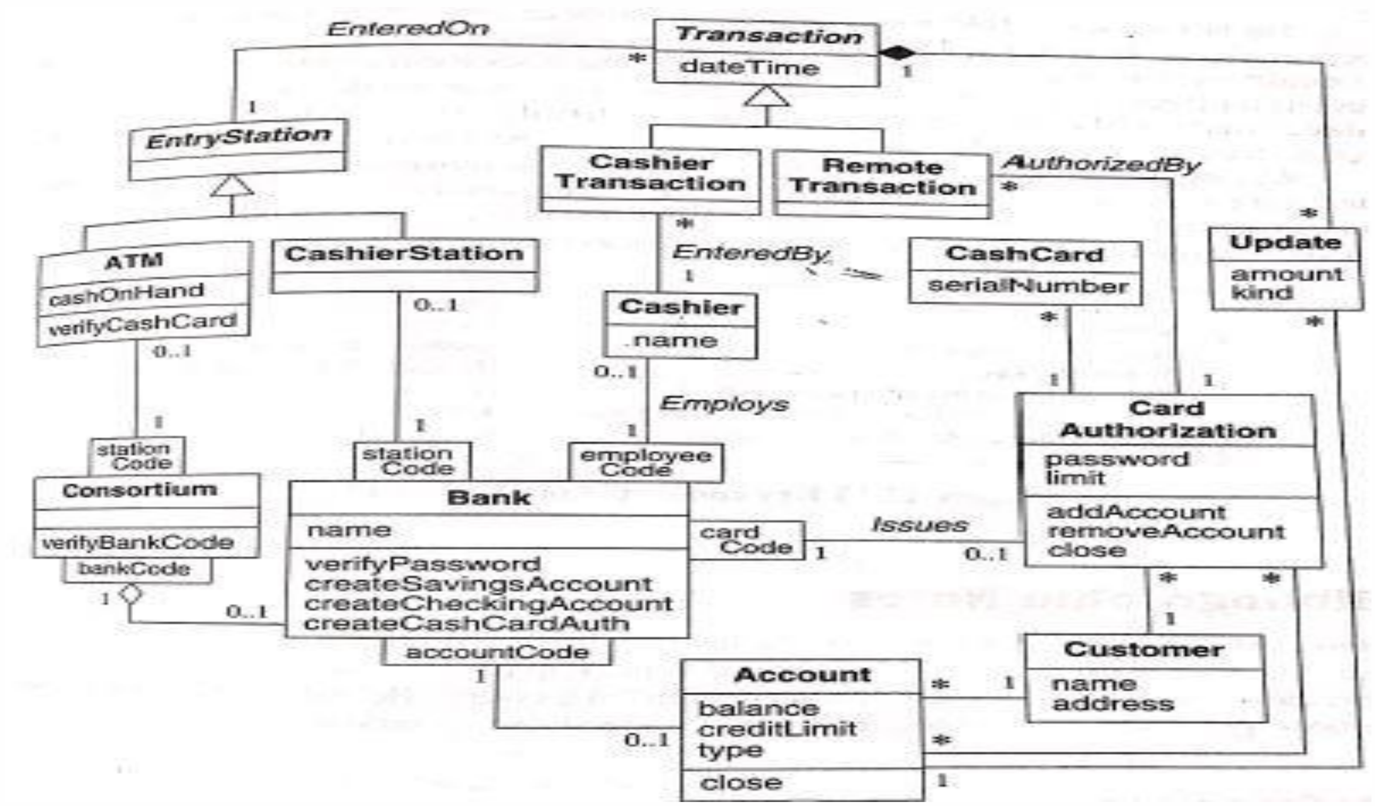


Figure: 4.31 ATM domain class model with some operations

Image source : Book: OOMD with UML by Blaha/Rumbaugh

References

- Object oriented Modeling and Design with UML (TextBook)
  By Michael R Blaha and James R Rambaugh.
- http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/
- https://en.wikipedia.org/wiki/Waterfall_model
- https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm
- https://www.indiamart.com/
- https://slidebazaar.com/

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in