Domain Analysis & Application Analysis

Unit - 4

Overview of analysis

- Analysis is divided into Two sub-stages

- Domain Analysis
- Application Analysis

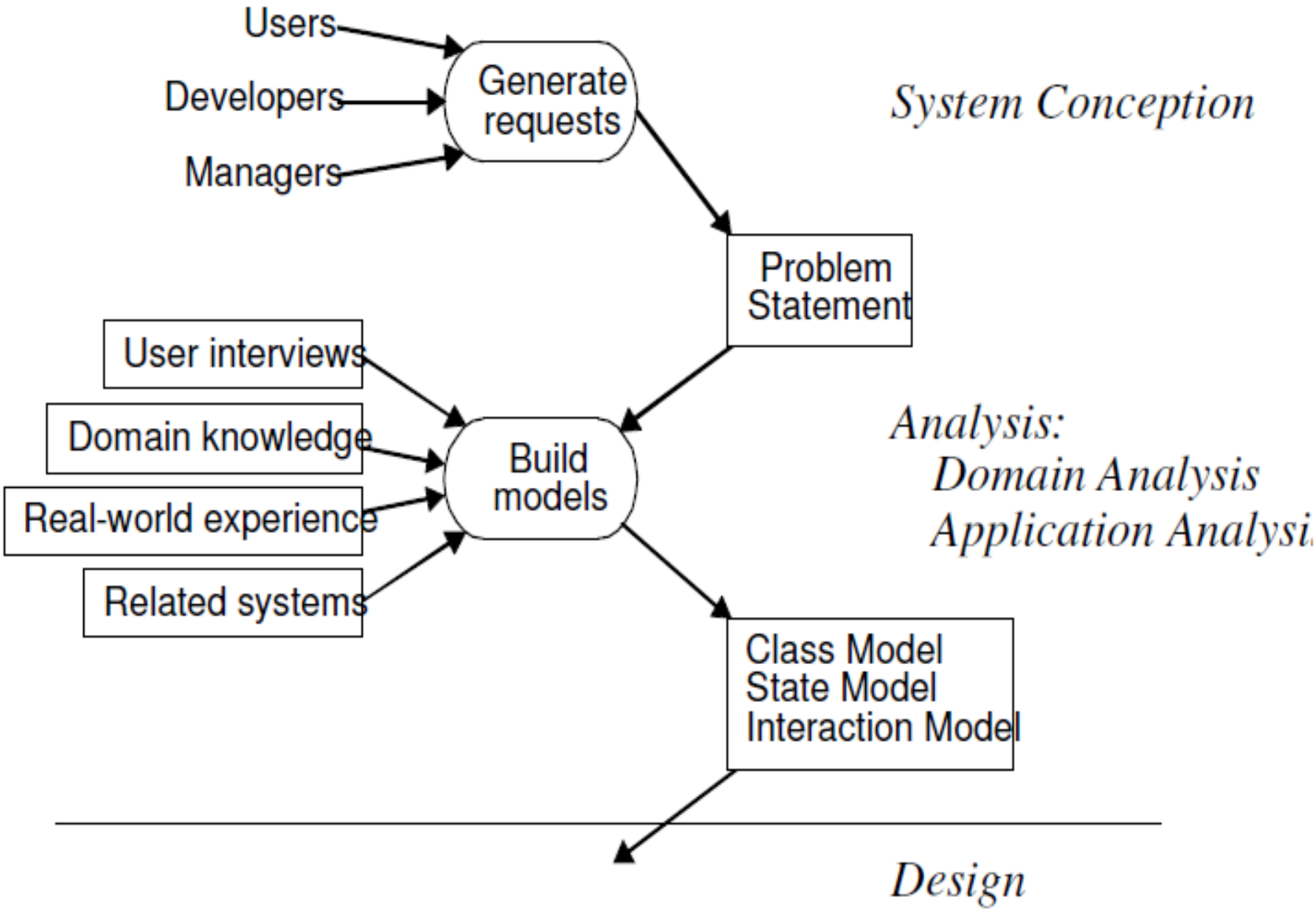


Fig 12.1 Overview of Analysis

- The analysis model addresses the 3 aspects of objects :

- Static structure of objects (class model)
- Interaction among objects (interaction model)
- Life-cycle histories of objects (State model)

- All the 3 models are not equally important in every problem
- Some problems have useful class models derived from real world entities.
- Some problems concerning reactive control and timing such as user interfaces and process control use state model.
- Some problems containing significant computation as well as systems that interact with other systems and different kind of users makes interaction models.

Domain class model

- First step in analysing the requirements is to construct a domain model.

- Domain model shows static structure of the real world system and organizes it into workable pieces.
- It describes real-world classes and their relationships to each other.

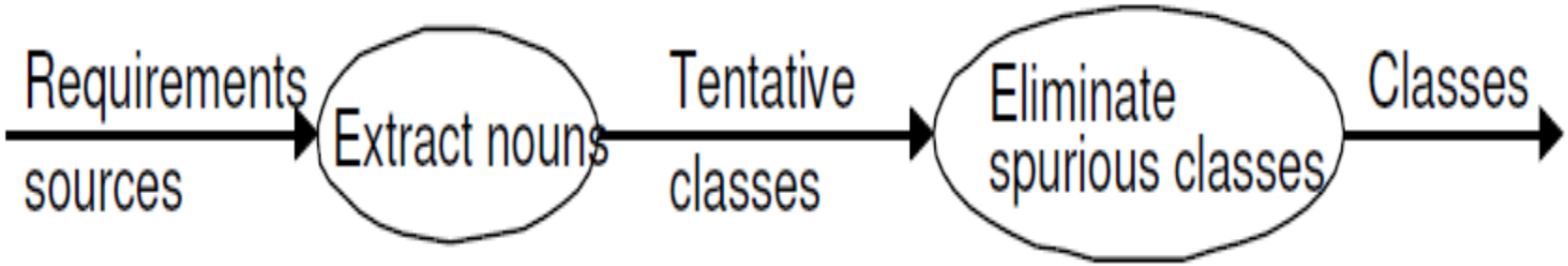
- Information for the domain model comes from problem statement, artifacts from related systems, expert knowledge of application domain and general knowledge of real world.

Steps to be performed to construct a domain class model:

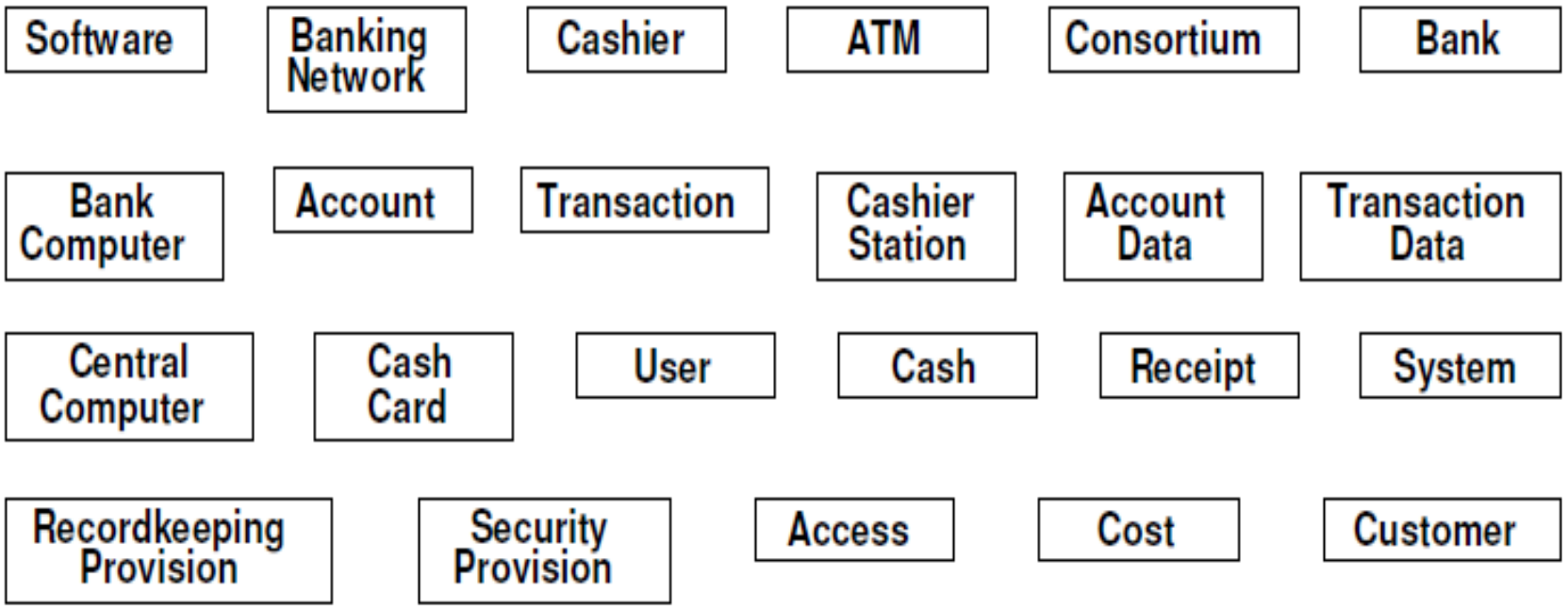
1. Find Classes.
2. Prepare a data dictionary.
3. Find associations.
4. Find attributes of objects and links.
5. Organize and simplify classes using inheritance.
6. Verify that access paths exist for likely queries.
7. Iterate and refine the model.
8. Reconsider the level of abstraction.
9. Group classes into packages

Step1: Finding classes:

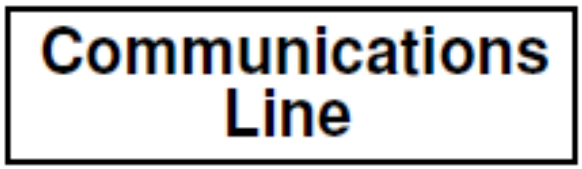
- First step in constructing a class model is to find relevant classes for objects from application domain.
- Classes often correspond to **nouns**

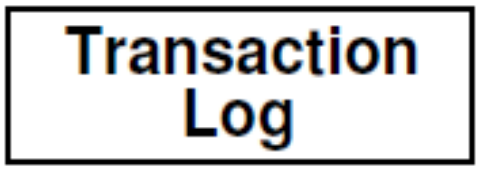


For the case study of the ATM: following are the classes extracted from problem statement nouns.



- The additional classes do not appear directly in the statement but can be identified from our knowledge of the problem domain





Domain Class Model

Keeping the right classes and discard unnecessary and incorrect classes according to the following criteria:

1. **Redundant classes**: If two classes express the same concept, you should keep the most descriptive name

- For ex: Customer and user are redundant;
- We can retain **Customer** because it is more descriptive.

2. Irrelevant classes: apportioning **cost** is outside the scope of the ATM software

3. Vague classes: Class should be specific.

4. Attributes: Names that primarily describe individual objects should be restated as attributes.

5. Operations: if a name describe an operation that is applied to the objects and not manipulated in its own right, then it is not a class.

Ex: if we are simply building telephones, then **call** is part of the state model and not a class.

6. Roles: the name of a class should reflect its intrinsic nature and not a role that it plays in an association.

- Ex: Owner of a car… in a car manufacturing database, not correct as a class. It can be a person (owner, driver, lessee)

7. Implementation constructs: Eliminate constructs from the analysis model that are extraneous to the real world. We may need them during design and not now.

8. Derived classes: As a general rule, omit classes that can be derived from other classes, mark all derived classes with a preceding slash (‘/’) in the class name.

Keeping the right classes:

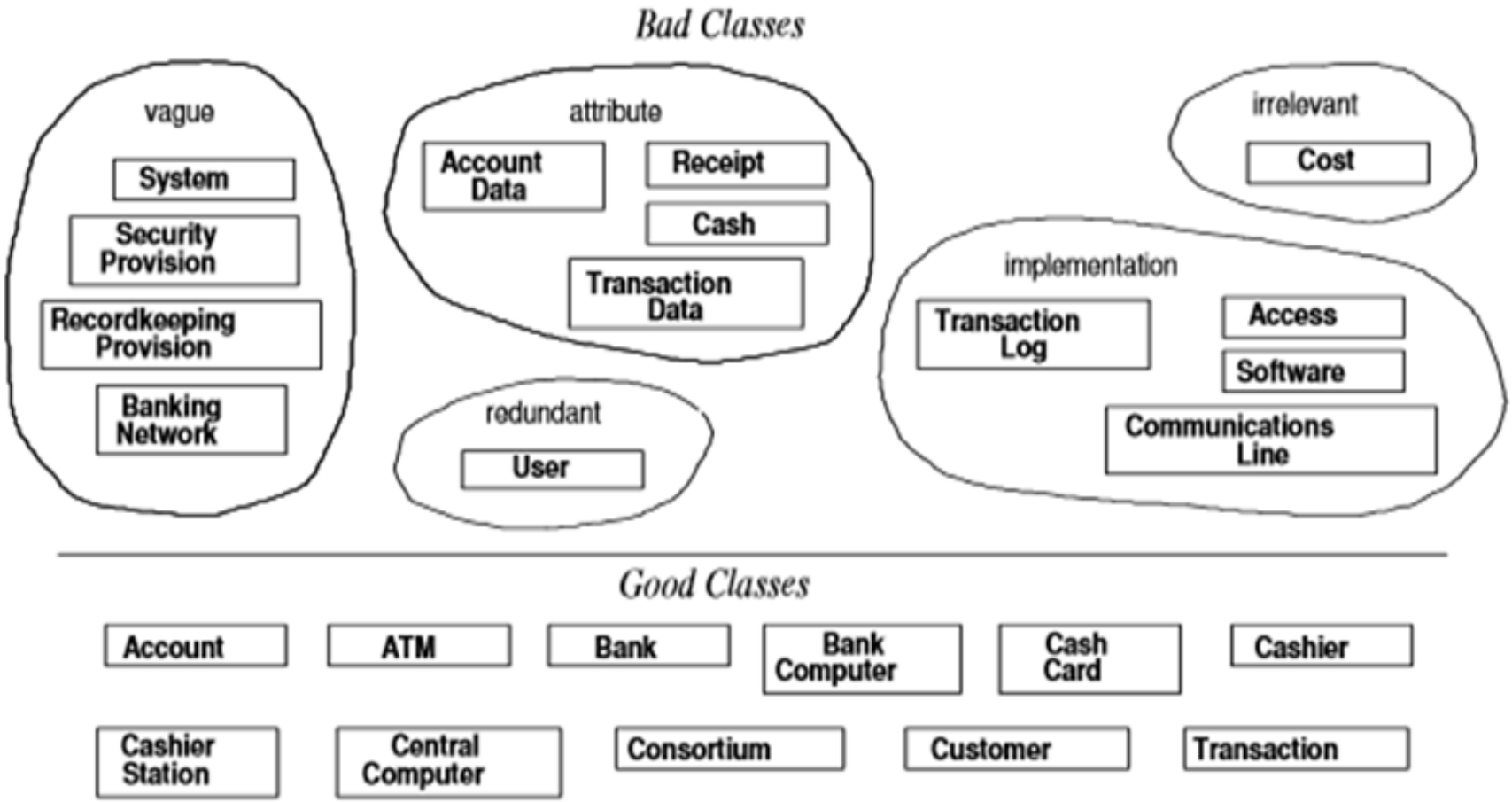


Figure: Eliminating unnecessary classes from ATM Problem.

Step-2: Preparing a data dictionary:

- Isolated words have too many interpretations, so prepare a data dictionary for all modeling elements.

- Describe the scope of the class within the current problem, including all assumptions or restrictions on its use.

- Data dictionary also describes associations, attributes, operations and enumeration values.

Example:
Data Dictionary for the ATM classes

- Account
- ATM
- Bank
- Bank Computer
- Cash Card
- Cashier
- Cashier Station
- Central Computer
- Consortium
- Customer
- Transaction

Step-3: Finding associations

- A structural relationship between two or more classes is an association.
- An association is a reference from one class to another class.
- Associations often correspond to verbs or verb phrases.
- Idea here is to capture relationships.

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

Step-4: Find attributes of objects and links.

- Attributes are data properties of objects
  - eg: weight, velocity, color.

- Attribute values **should not be objects**; use an association to show any relationship between objects.

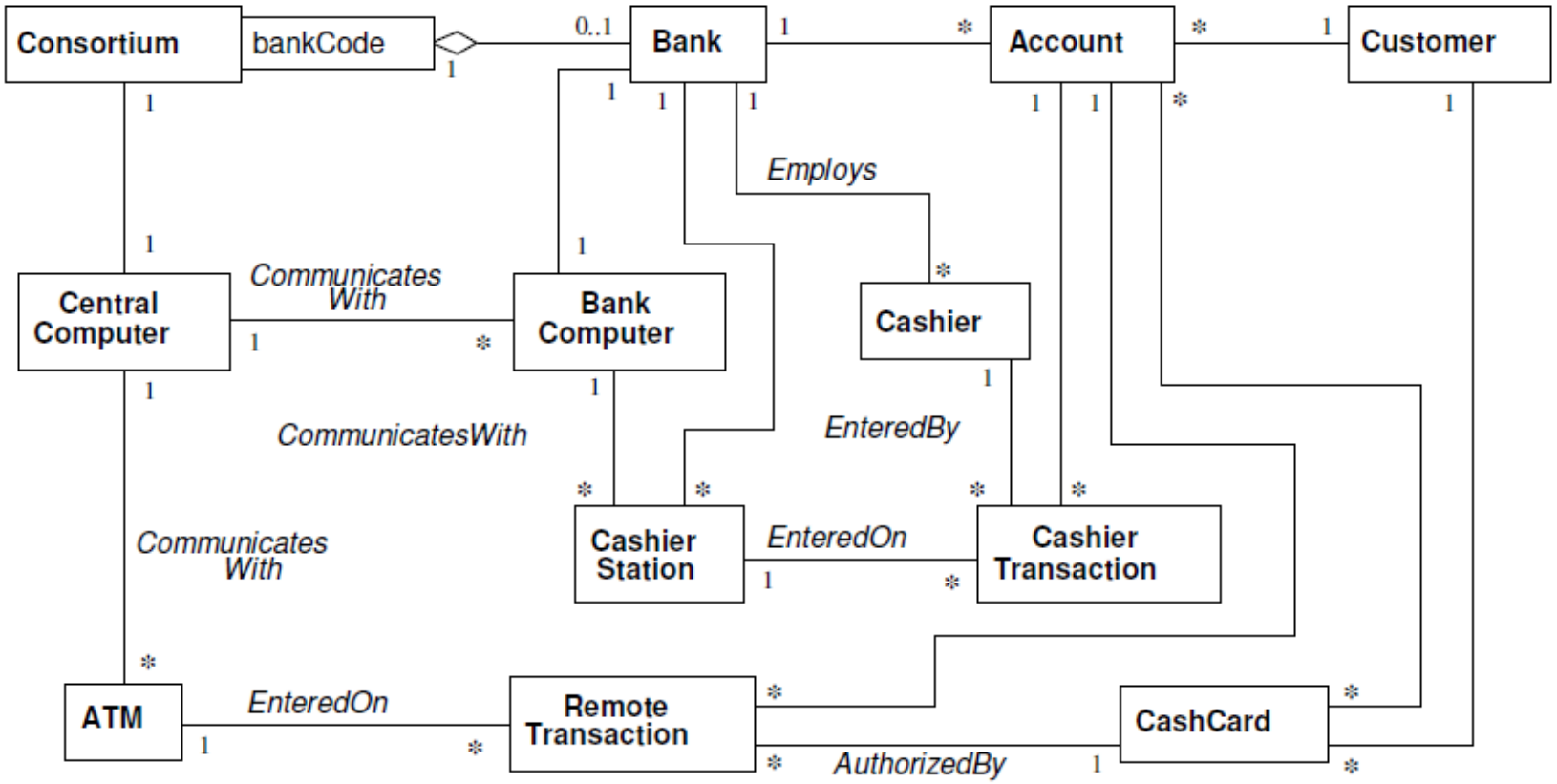
- Attributes usually correspond to nouns followed by possessive phrases, such as
- "the color of the car"

Keeping the right attributes:

Eliminate **unnecessary and incorrect attribute** with the following criteria:

- Objects
- Discordant attributes
- Qualifiers
- Boolean attributes.
- Names
- Identifiers
- Attributes on associations
- Internal values
- Fine detail

ATM example:
we apply these criteria to obtain attributes for each class



Step-5: Refining with inheritance

- Next step is to organize classes by using inheritance to share a common structure.

- Inheritance can be added in **2 directions**:

1. By generalizing common aspects of existing classes into a super classes (bottom up).
2. By specialising existing classes into multiple subclasses (top down)

To Share common features we organize classes using inheritance

- Bottom up generalization.
- Top-down generalization
- Generalization vs. enumeration.
- Multiple inheritance
- Similar associations
- Adjusting the inheritance level.

Step-6: Testing access paths

- **Trace access paths** through the class model to see if they **yield sensible results**.
  - Where a unique value is expected, is there a path yielding a unique result ?
  - They **indicate missing information**, if something that seems simple in real world appears complex in the model, you may have missed something

- Verify that access paths exist for likely queries.

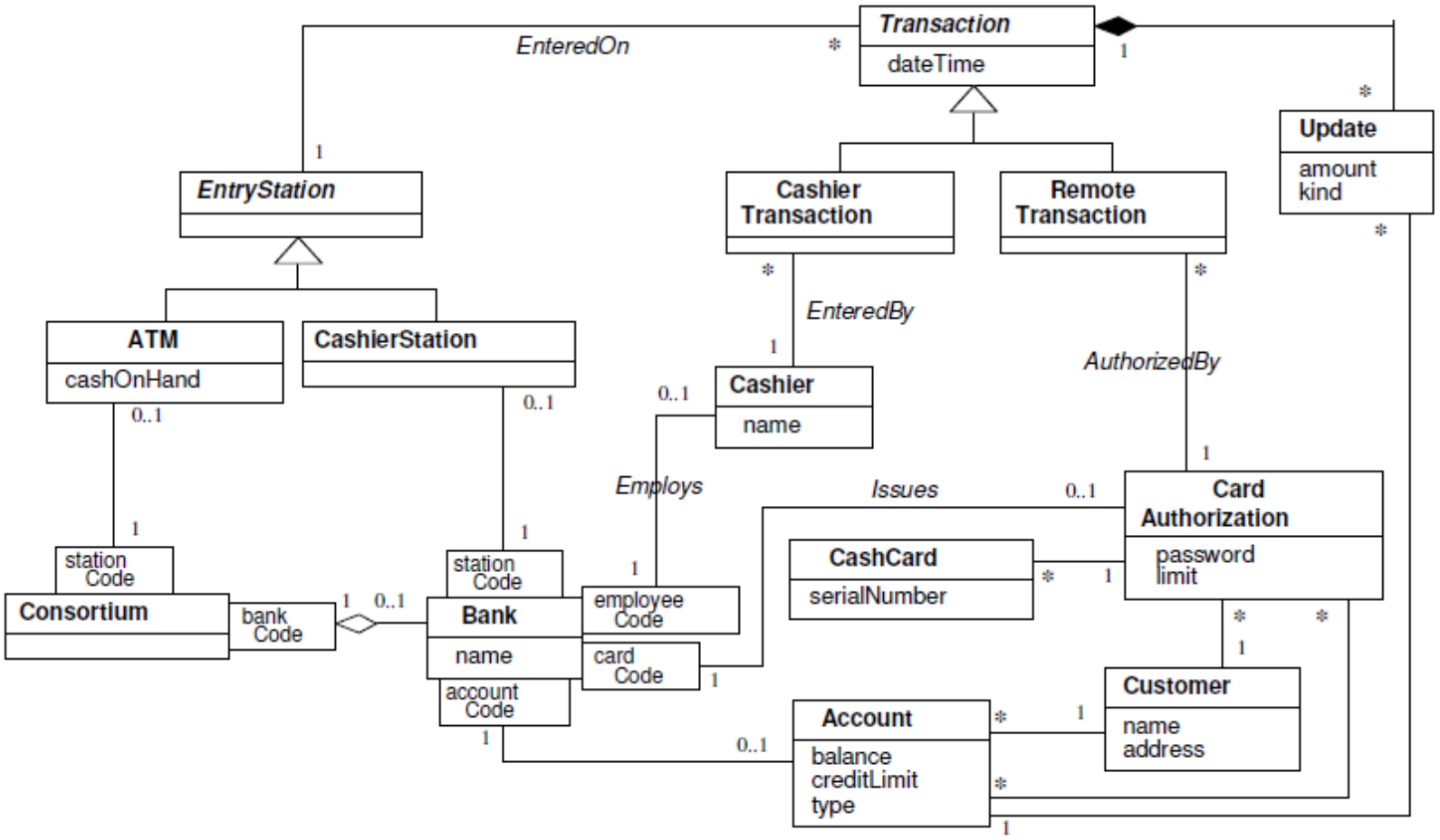
Step-7: Iterate and refine the model.

- A class model is rarely correct after a single pass, so **iterate** and **refine** the model.

- Different parts of a model are often at different stages of completion. If you find a deficiency, **go back to an earlier stage if necessary to correct it**.

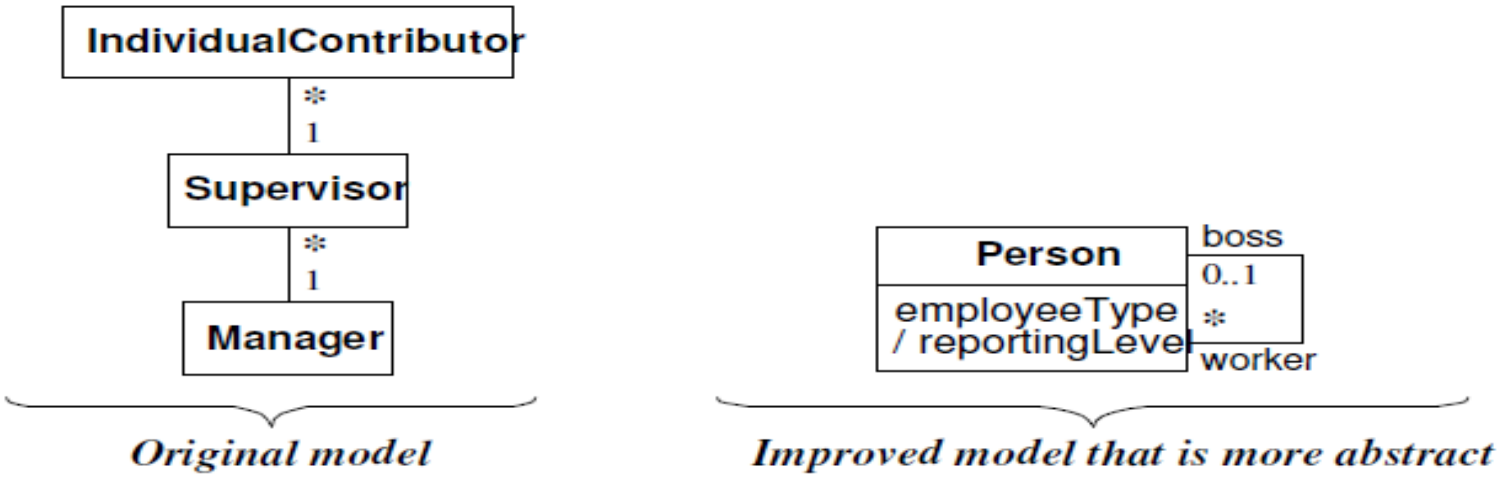
- Some refinement can come only after completing the state and interaction models.

Revised class diagram that is simpler and cleaner.



Step-8: Reconsider the level of abstraction.

- Abstraction makes a model more complex but can increase flexibility and
- **reduce the number of classes**.



Step-9: Group classes into packages.

- The last step of class modelling is to group classes into **packages**.

- A package is a **group of elements** (classes, association, generalizations and lesser packages) with common theme.

Domain State Model

Domain State Model

- The Following steps are performed in constructing a domain state model

1. Identifying classes with states
2. Finding states
3. Finding Events
4. Building state diagrams
5. Evaluating state diagrams

Step-1: Identifying classes with states

- Examine the list of domain classes for those that have a **distinct life cycle.**

- Identify the **significant states** in the life cycle of an object.

- Example:

1. A scientific paper for journal goes from being written to under consideration to accepted or rejected .
   - There can be some cycles for eg: reviewers ask for some revisions
2. An airplane owned by an airline cycles, the states of maintenance, loading, flying and unloading.

Step-2: Finding states

- List the states of each class
- Characterize the objects in each class, attribute value, associations they participate, their multiplicities and so on.
- Give each state a meaningful name .
- States should be based on qualitative differences in behaviour like attributes or associations but not on quantitative differences such as small medium and large.
- By examining events can determine all the states. By looking at events and considering transitions among states, missing states will become clear.

Step-3: Finding Events

- Once you have a set of states, find the event that cause transitions among states.

- Think about the stimuli that cause a state to change, you can regard an event as completing a do-activity.

  - Example: if a technical paper is in the state under consideration can be positive or negative(reject paper)
  - Also other possibilities are often possible and may be added in future .
  - Example: Conditionally accept the paper with revisions.

- There are additional events that occur within a state and **do not** cause a transition.

- For domain state model you **should focus on events that cause transitions among states**

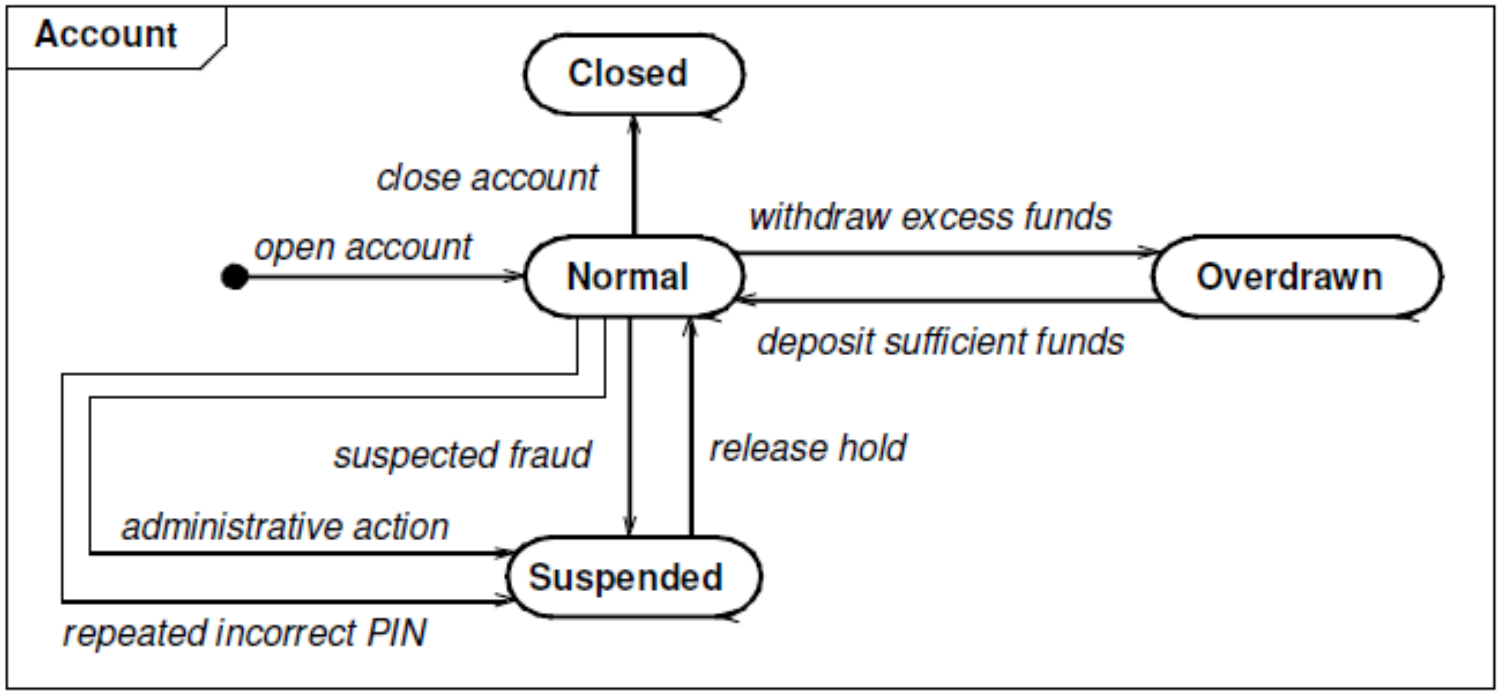
For example:
ATM important events include:

- Close account
- Withdraw excess funds
- Repeated incorrect PIN
- Suspected fraud
- Administrative action

Step-4: Building state diagram

- Note the states to which each event applies.
- Add transitions to show change in state caused by occurrence of events
- If the object in the class performs activities on transition add them to positive state diagram

ATM example:
domain state model for the account class



Step-5: Evaluating state diagrams

Examine each state model :

- Are all states connected?
- If it represents a progressive class, is there a path from initial state to final state?
- Are the expected variations present?
- If it represent a cyclic class, is the main loop present?
- Are there any dead states that terminates cycle?

Domain Interaction Model

Domain Interaction Model

- The Interaction model is seldom (infrequently or rarely) important for domain analysis.

- The Interaction model is an important aspect of application modelling.

Application Analysis

Application Interaction Model

- The focus of domain modelling is on building model of intrinsic concepts.
- After completing the domain model then we shift our attention to the details of an application and **consider interaction**.
- Begin the interaction modelling by determining the overall boundary of the system.
- Then identify use cases and flesh them out with scenarios and sequence diagram.
- Finally check against the domain class model to ensure that there are no inconsistencies.

To Construct an application interaction model requires following steps:

1. Determine the system boundary.
2. Find actors
3. Find use cases
4. Find initial and final events
5. Prepare normal scenarios
6. Add variation and exception scenarios.
7. Find external event.
8. Prepare activity diagram for complex use case.
9. Organize actors and use cases.
10. Check against the domain class model

Step-1: Determine the system boundary

- We must know the scope of application-boundary of system, in order to specify functionality.

- It means what system includes and **implicitly what it omits.**

Step-2: Finding actors

- Once you determine system boundary, you must identify the external objects that interact directly with the systems.
- And these are its actors

- Actors include humans, external devices and other software systems.

- Important thing is actors are not under control of applications.

Cont…

- Examine each external object to see if it has several distinct faces.
- An actor is a coherent face presented to system, and an external object may have more than one actor.
- It is also possible for different kinds of external objects to play the part of the same actor.
- Example: A particular person may be both a bank employee and bank customer at the same bank

Cont…

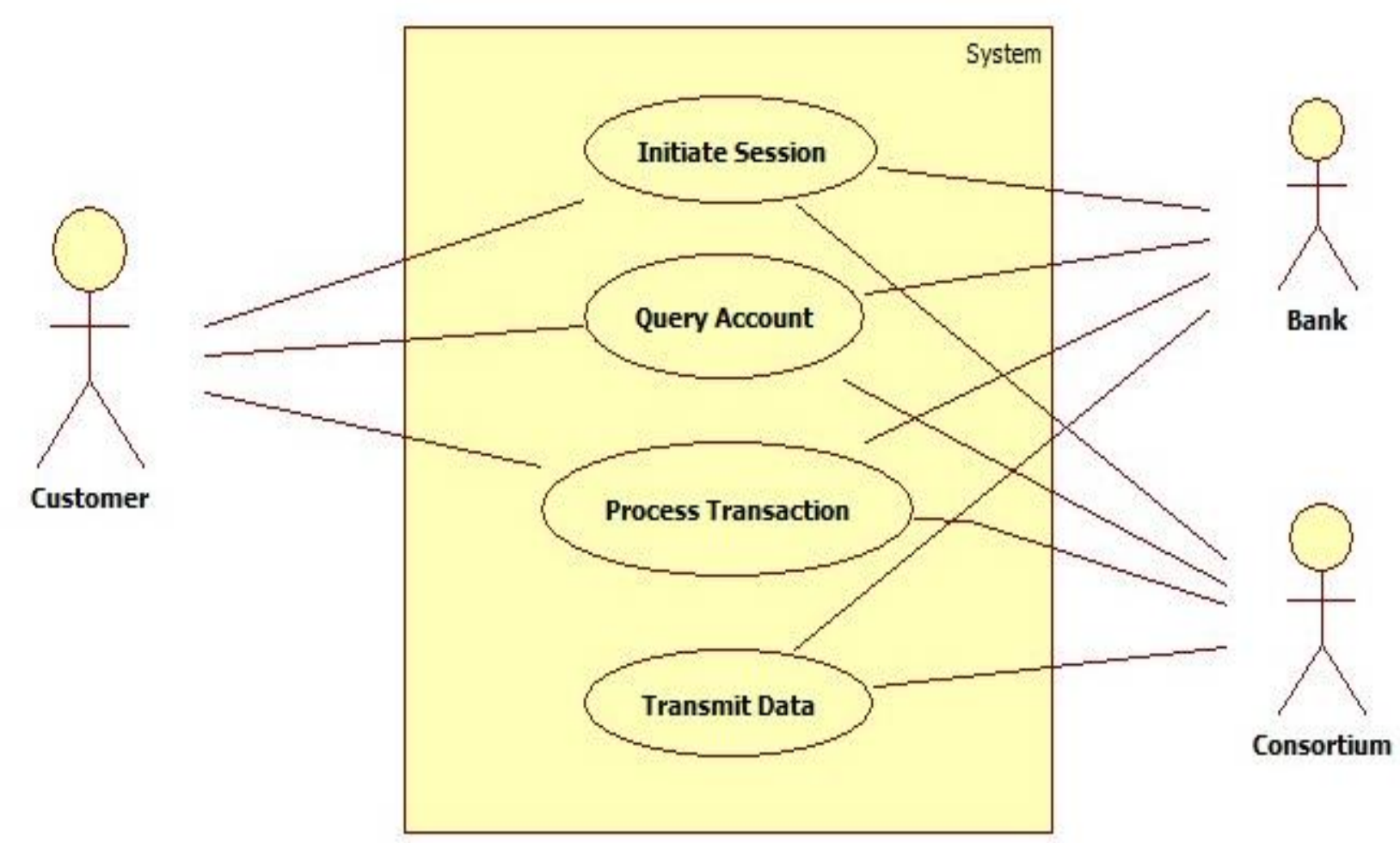
For ATM applications the actors are

- customer
- Bank
- Consortium.

Step-3. 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.
- Each use case should represent a kind of service that the system provides.
- Example: A preliminary use case diagram show the actors and use cases, connect actor to use cases.

Finding Use case in ATM:



Cont…

- Use cases partition the functionality of a system into small number of discrete units that cover its behaviour.

Initiate session:

- The ATM establishes the identity of user and makes available a list of accounts and actions.

Cont…

Query account:

- The system provides general data for account such as the current balance, date of last transaction and date of mailing for last statement.

Process Transaction:

- The ATM performs an action that affects an account's balance such as deposits, withdraw and transfer.

Transmit data:

- The ATM uses the consortium facilities to communicate with the appropriate bank computers.

Step-4. Finding Initial and final event

- Use cases partition system functionality into discrete pieces and show the actors that are involved with each piece, but they do not show the behaviour correctly.

- To understand behaviour you must understand the execution sequence that cover each use cases.

- You can start by finding the events that initiates each use case, Determine which actor initiates the use case and define the events it sends to the system.

- You must also determine the final event or events and how much to include in each use case.

Cont…

- Here are Initial and Final events for each use case.

Initiate Session :

- Initial event → is Customer's insertion of cash card.
- There are two final events → System keeps the cash card or System returns the cash card.

Query Account:

- Initial event → customer's request for account data
- Final event → System's delivery of account data to customer.

Process Transaction:

- Initial event → Customer's initiation of transaction.
- There are 2 final events → Committing or aborting the transaction.

Transmit data:

- Initial event → could be triggered by customer's request for account data.
- Another possible initial event is recovery from network, power or another kind of failure.
- Final event → is successful transmission of data.

Step-5. Preparing Normal Scenarios

- A scenario is a sequence of events among a set of interacting objects.

- Scenario illustrate
  - Major interactions
  - External display formats
  - Information exchanges

- Prepare Scenarios for "normal" cases – interactions without any unusual inputs or error conditions

Step-6. Adding Variation & Exception Scenarios

- After prepared typical scenarios, consider "special" cases, such as omitted input, max. & min. values and repeated values.

- Then consider error cases, including invalid values and failures to respond.

ATM example: some **variations** and **exceptions** are as follows:

We could prepare scenarios for each of these but **will not** go through the details here.

- The ATM cant read the card.
- The card has expired.
- The ATM Times out waiting for a response.
- The amount is invalid.
- The machine is out of cash or paper.
- The communication lines are down.
- The transaction is rejected because of suspicious patterns of card usage.

- There are additional scenarios or administrative parts of ATM system

such as

- Authorizing new cards
- Adding banks to the consortium
- Obtaining transaction logs.

Step-7. Finding External events

- Examine the scenario to find all external events - include all inputs, decisions, interrupts and interactions to and from users or external devices.
- An event can trigger effect for a target objects.
- Use scenarios to find normal events but **don't forget** unusual events and error conditions.

Cont…

- To find external event, transmission of information to an object is an event.
  - For example enter password is a message sent from external agent to application object ATM.

- Many **events has parameters**.
- Ex: Enter password should be an event whose parameter is **password value**.

  - The choice of password value does not affect the flow of control.

Prepare a Sequence diagram for each scenario.

- A sequence diagram shows the participants in an interaction and the sequence of messages among them.

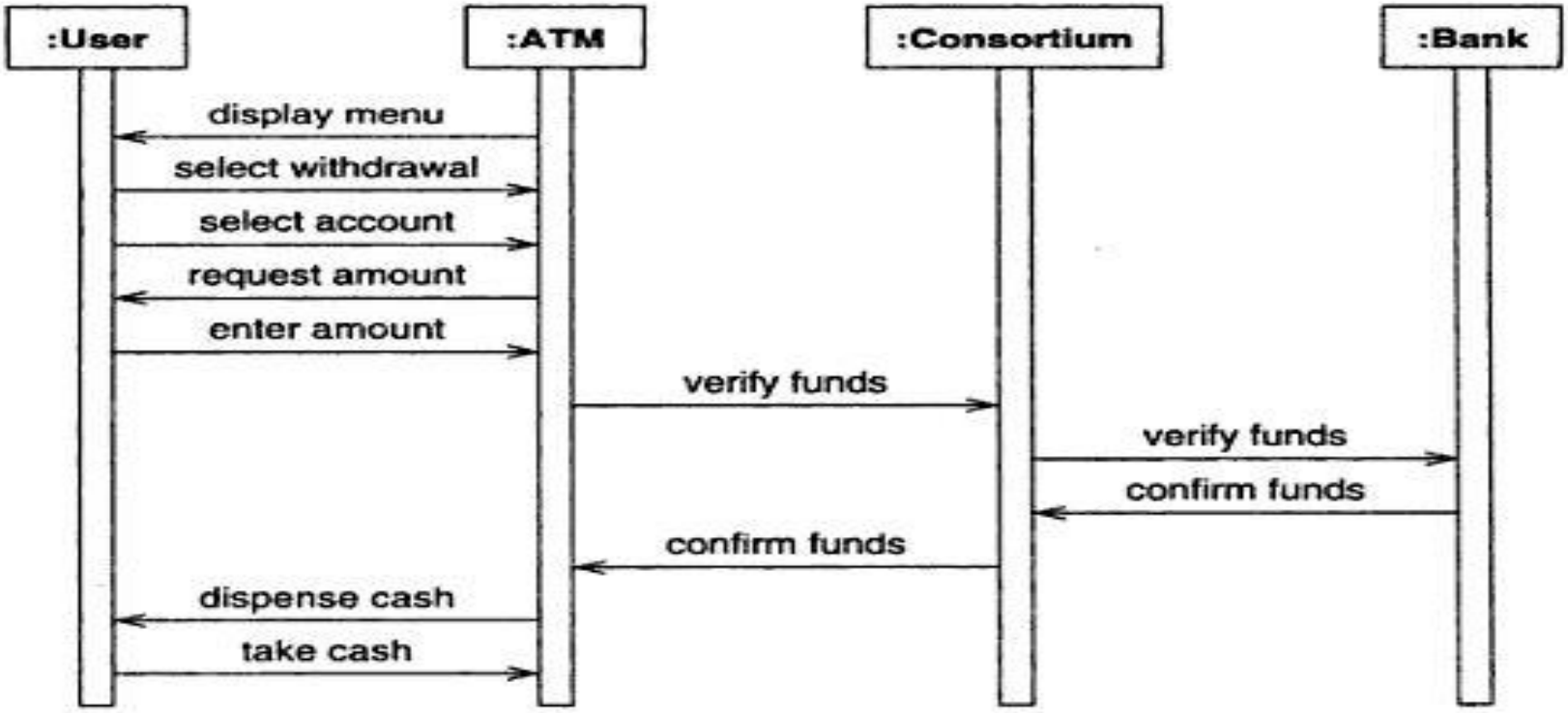


Figure 13.3 Sequence diagram for the *process transaction* scenario. A sequence diagram clearly shows the sender and receiver of each event.

Step-8. Preparing Activity diagram for complex Use cases

- Sequence diagram capture the dialog and interplay between actors, but they do not clearly show alternatives and decisions.

- For example:
- we need one sequence diagram for the main flow of interaction and additional sequence diagram for each error and decision point.

- Activity diagrams let you to consolidate all this behaviour by documenting forks and merges in the control flow.

ATM example for card verification:

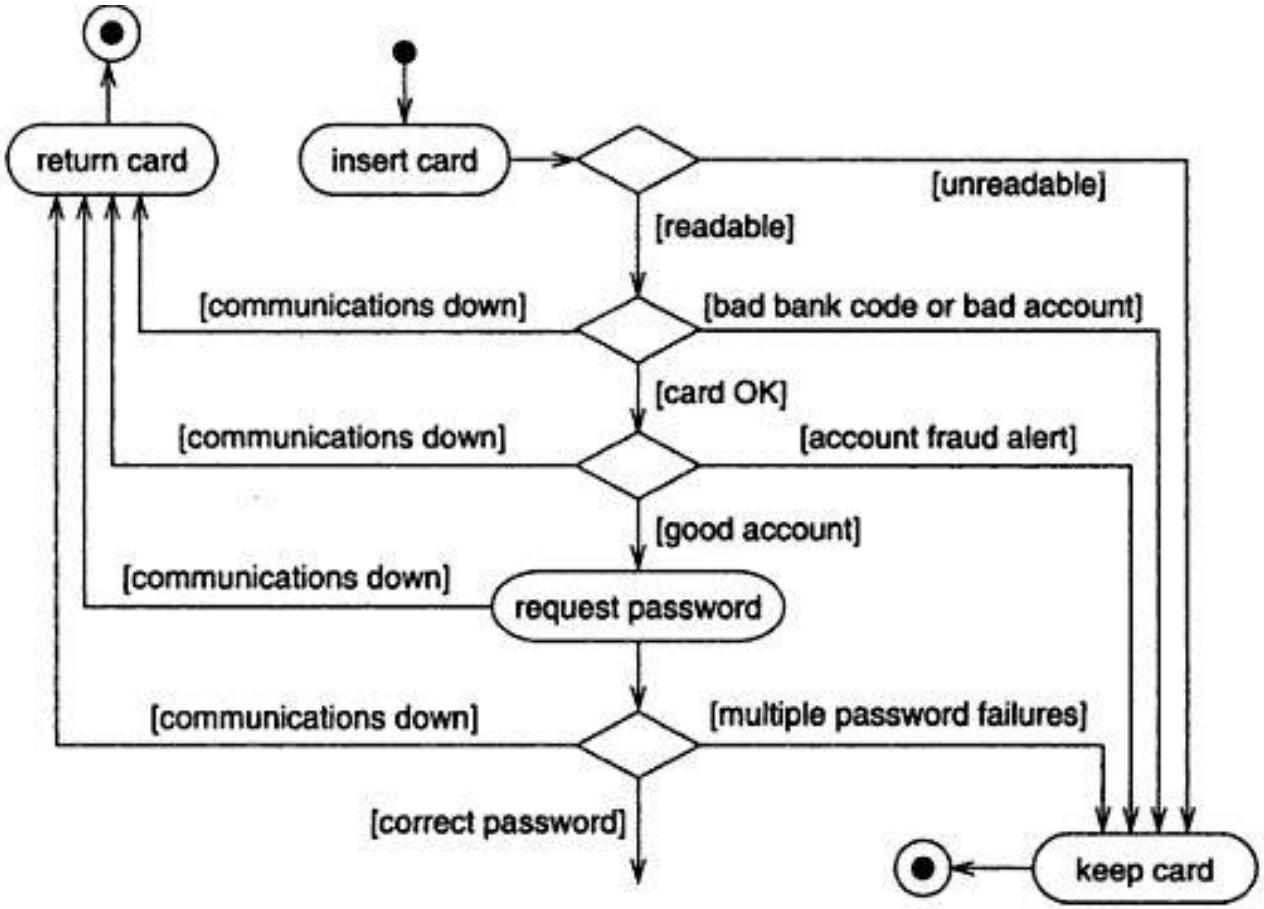


Figure 13.5 Activity diagram for card verification. You can use activity diagrams to document business logic, but do not use them as an excuse to begin premature implementation.

Step-9. Organizing actors and use cases

- Next step is to organize use cases with relationships (include, extend and generalization).
- This is especially helpful for large and complex systems.
- You can organize actors with generalization.

Example:

- Organizing the use cases once the basic use cases are identified, you can organize them with relationship.

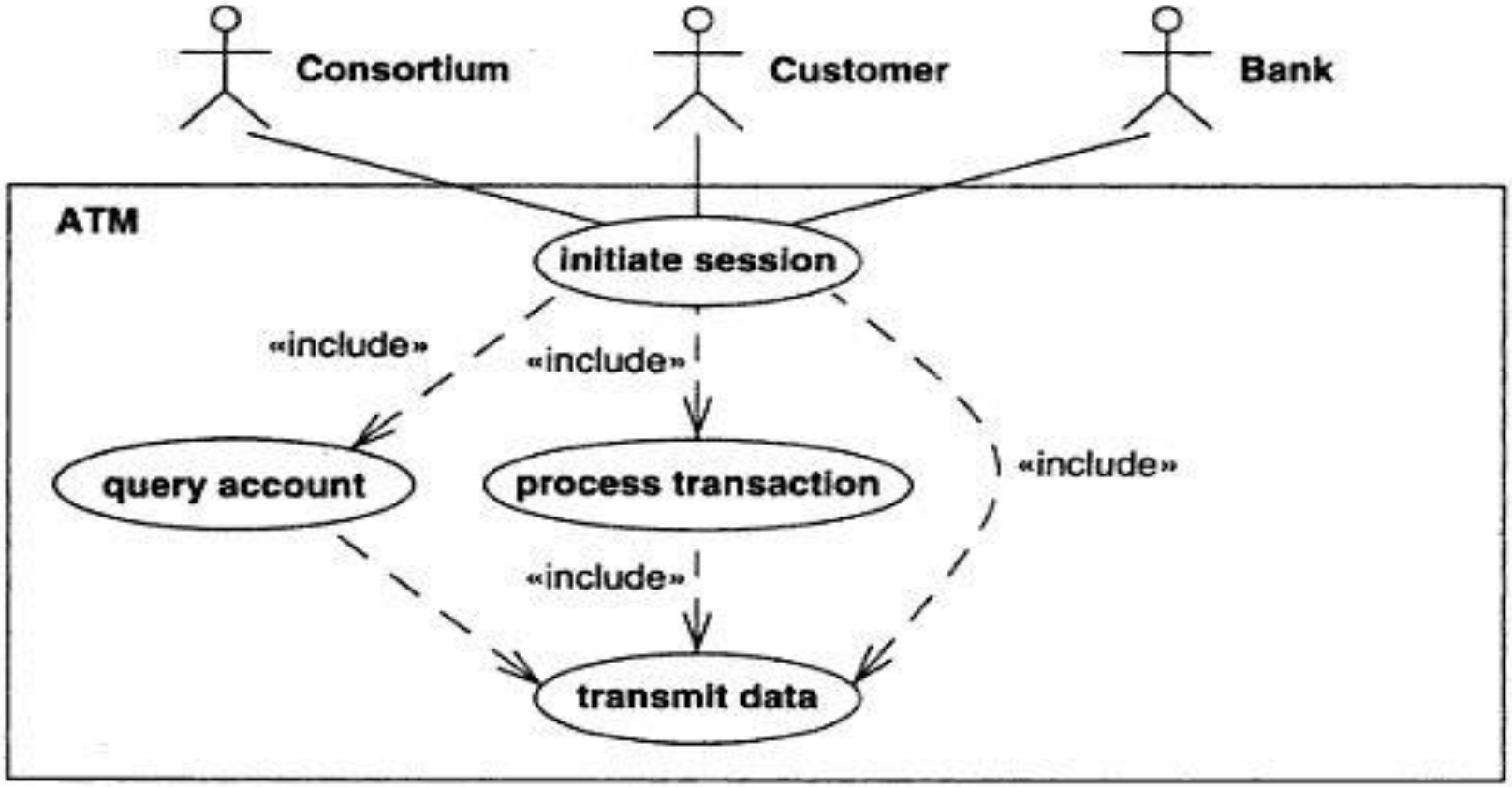


Figure 13.6 Organizing use cases. Once the basic use cases are identified, you can organize them with relationships.

Step-10. Checking against the domain class model

- At this point, the application and domain model should be mostly consistent.

- The actors, use cases and scenarios are all based on classes and concepts from domain model.

- Recall, one of the step in constructing the domain class model is to **test access path.**

- In reality such testing is a first attempt at use cases.

Cont…

- Cross check the application and domain model to ensure that there are no inconsistencies
- Examine the scenarios and make sure that the domain model has all the necessary data.
- Also make sure that domain model covers all event parameters.

Application Class Model

Application Class Model

- Application class defines application itself rather than real world objects that application acts on.

- Most application classes are computer oriented.

You can construct an application class model with following steps

1. Specify the user interfaces
2. Define boundary classes
3. Determine controllers
4. Check against the interaction model.

Step-1. Specify the user interfaces

- Interactions can be separated into 2 parts
  - Application logic
  - User interface
- A **user interface** is an object or group of objects that provides the **user** of a system with a **coherent way to access its domain objects, command and application options.**
- The **program logic** can accept input from **command line, files, mouse buttons, touch panels, push buttons or remote links**, if the surface details are carefully isolated.

- Sketch out a sample interface to help to visualize the operation of an application and see if anything important has been forgotten.

- You may also mock up interface so that users can try it

Example:

- Decoupling application logic from user interface let you evaluate the “look and feel” of the user interface while the application is under development.

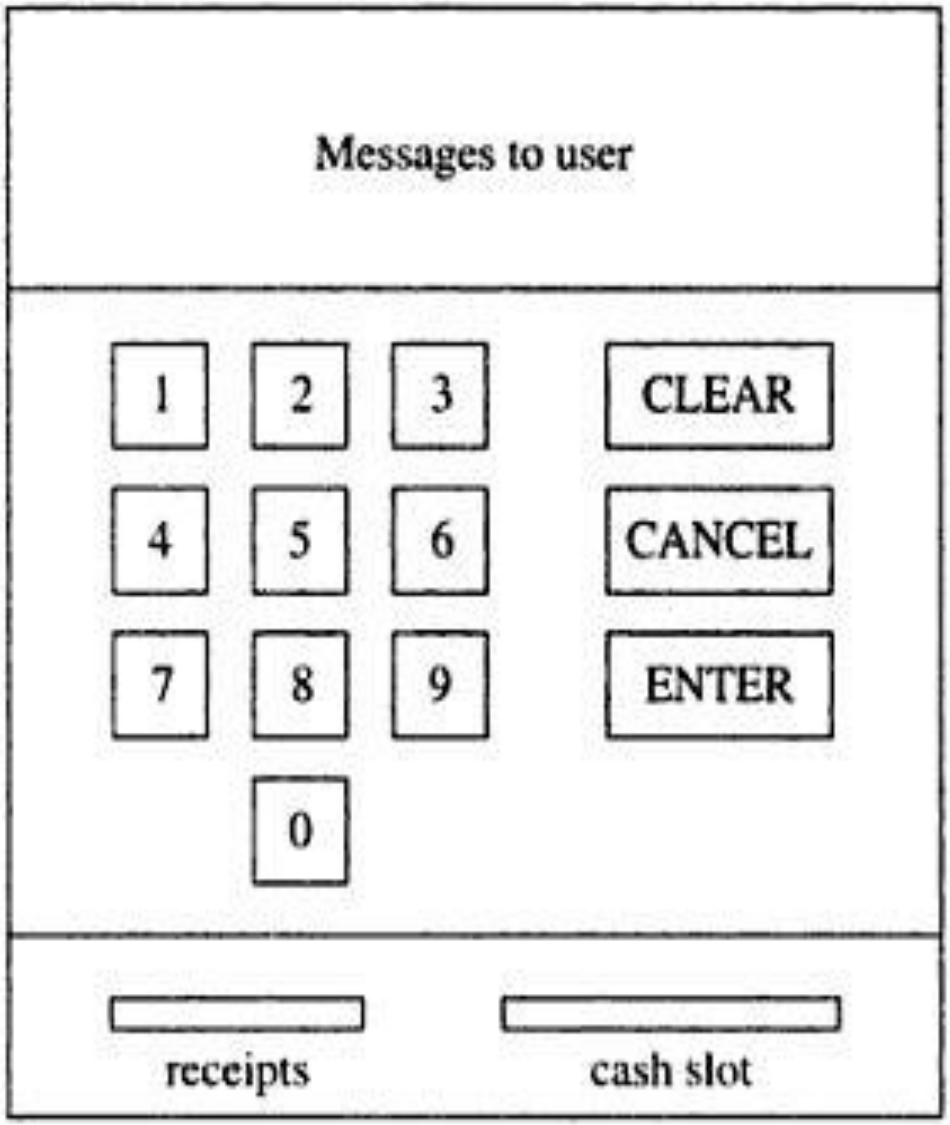


Figure 13.7 Format of ATM interface. Sometimes a sample interface can help you visualize the operation of an application.

Step-2. Defining boundary classes

- It is often helpful to define boundary classes to isolate the inside of a system from the external world.

- A boundary class is a class that provides a **staging area** for communication between a system and an external source.

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

for Example:

- ATM boundary classes,
- cash card boundary classes,
- account boundary
  - helps to encapsulate the communication between ATM and consortium.

Step-3. Determining controllers

- A controller is an active object that manages control within an application.

- It receives signals from **outside world** or **from objects within the system**, reacts to them, invokes operation and send signals to outside world.

- A controller is a **behaviour** captured in the form of an object behaviour that can be manipulated and transformed more easily than plain code.

Example: in ATM example, it has 2 major control loops.

- The outer verifies customer and accounts.
- The inner loop services transactions.

Step-4. Checking against the interaction model

- As we build application class model, go over **use cases** and think about how they would work?

Example:

- If a user send a command to a application, the parameters of the command must come from some user interface object.

- When domain and application model are in place, we should be able to **simulate a use case with the classes**.

Application State Model

Application State Model

- The application state model focuses on application classes and augments the domain state model.
- First identify application classes with multiple states and use interaction model to find events for these classes.
- Then organize permissible event sequences for each class with a state diagram.
- Next, check various state diagram to make sure that common events match.
- Finally check state diagram against the class model and interaction model to ensure consistency.

To construct Application state model use following steps:

1. Determine application classes with states.
2. Find events.
3. Build state diagram
4. Check against other state diagrams
5. Check against the class model
6. Check against the interaction model.

Adding Operations

Adding Operations

- The traditional programming based methodologies emphasis on defining operation.

- Here, OOA we de-emphasize operation because the list of potentially useful operations is open-ended and it is difficult to know when to stop adding them.

• Operations arises from following sources:

1. Operation from the class model:

Reading and writing of attribute values and association links are implied by class model.

2. Operations from use cases :

- Most of the complex functionality of a system comes from its use cases.
- During construction of interaction model, use cases lead to activities. Many of these corresponds to operation on class model.

3. Shopping-List operations:

- Some times real-world behavior of classes suggests operations.
- It is called "shopping-list" because the operations are not dependent on a particular application but are meaningful in their own right.
- Shopping-list operations provide an opportunity to broaden a class definition beyond the narrow needs of immediate problem.

- Ex; ATM eg for shopping list operation include:
- Account.close()
- Bank.createSavingsAccount(customer): account.
- Bank.createCashCardAuth(customer):cashCardAuthorization
- cashCardAuthorization.addAccount(account)
- cashCardAuthorization.removeAccount(account)
- cashCardAuthorization.close()

4. Simplifying operations:

- Examine class model for similar operations and variations in form on a single operation.
- Use inheritance where possible to reduce the number of distinct operations.
- Introduce new super classes as needed to simplify the operation.

END OF UNIT-4