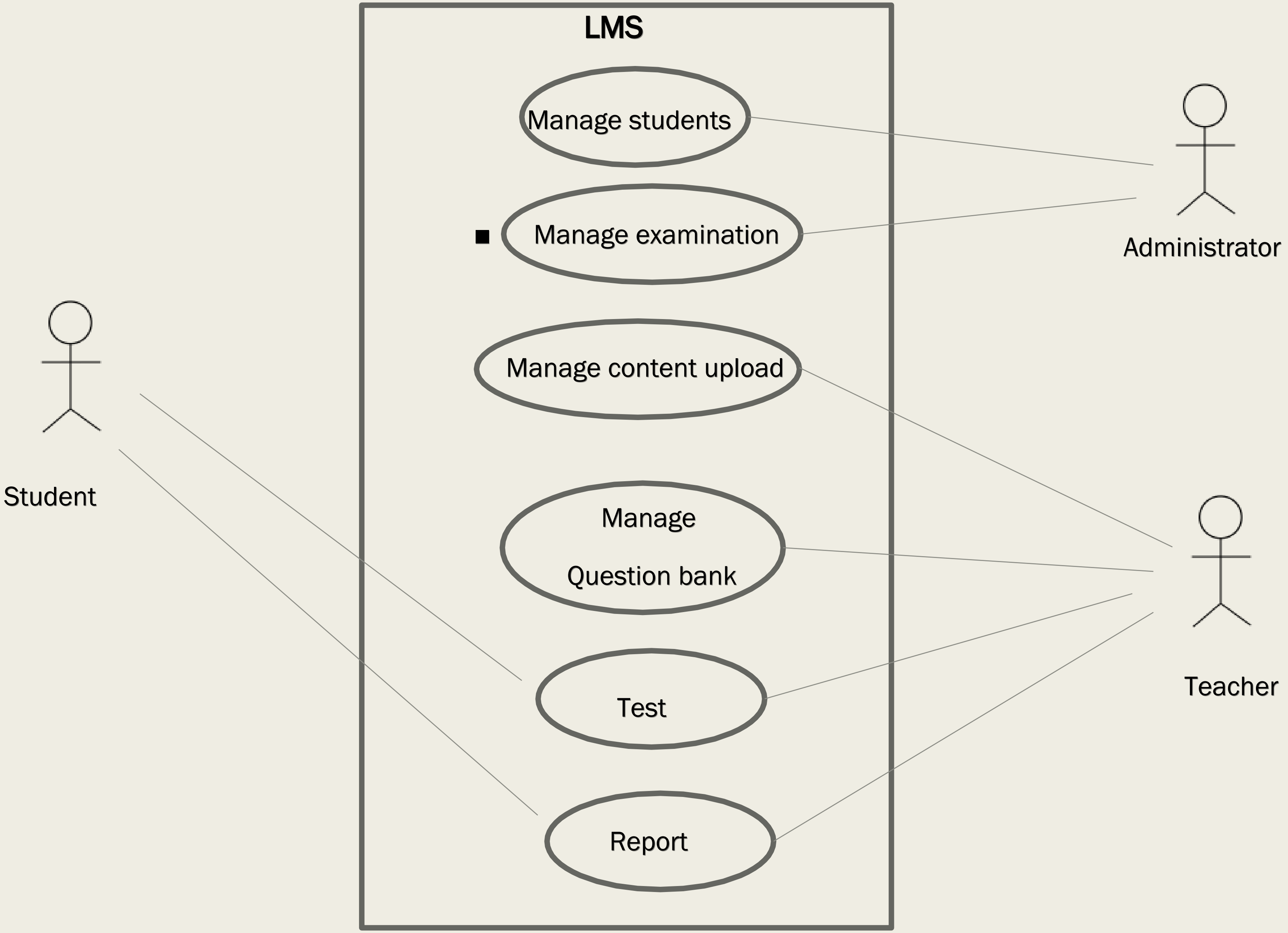
UNIT-2

Chapter -3

Advanced Interaction Model

Use case diagram for student LMS portal

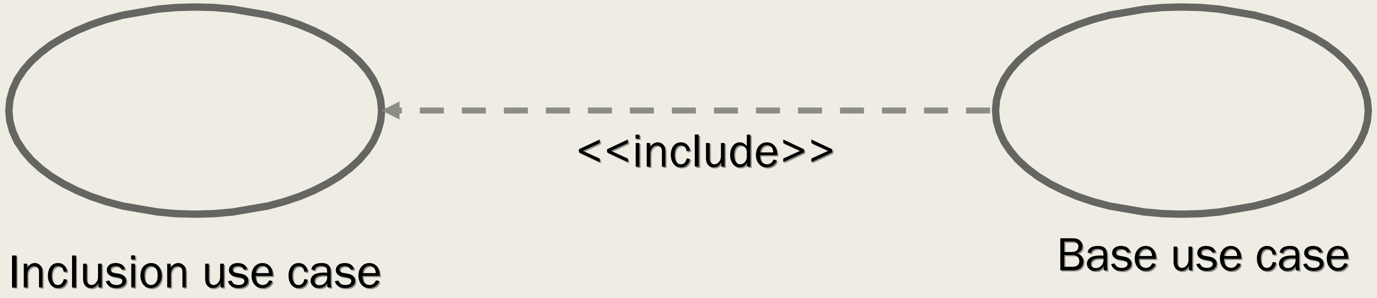


Use case Relationships

- Complex use cases can be built from smaller pieces with include, extend, and generalization relationship

Include Relationship

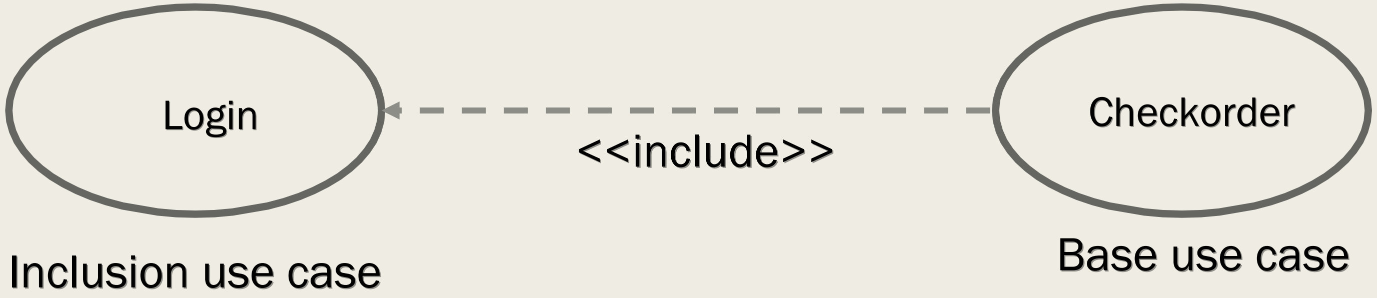
- ❖ The include relationship is a relationship in which one use case includes the functionality of another use case
- ❖ The include relationship supports the reuse of functionality in a use case model
- ❖ A base use case is dependent on the included use case(s); without it/them the base use case is incomplete as the included use case(s) represent sub-sequences of the interaction that may happen always OR sometimes
- ❖ The UML notation for an include relationship is a dashed arrow from the source use case to the target use case
- ❖ The keyword <<include>> annotates the arrow



<<include>>

Inclusion use case

Base use case



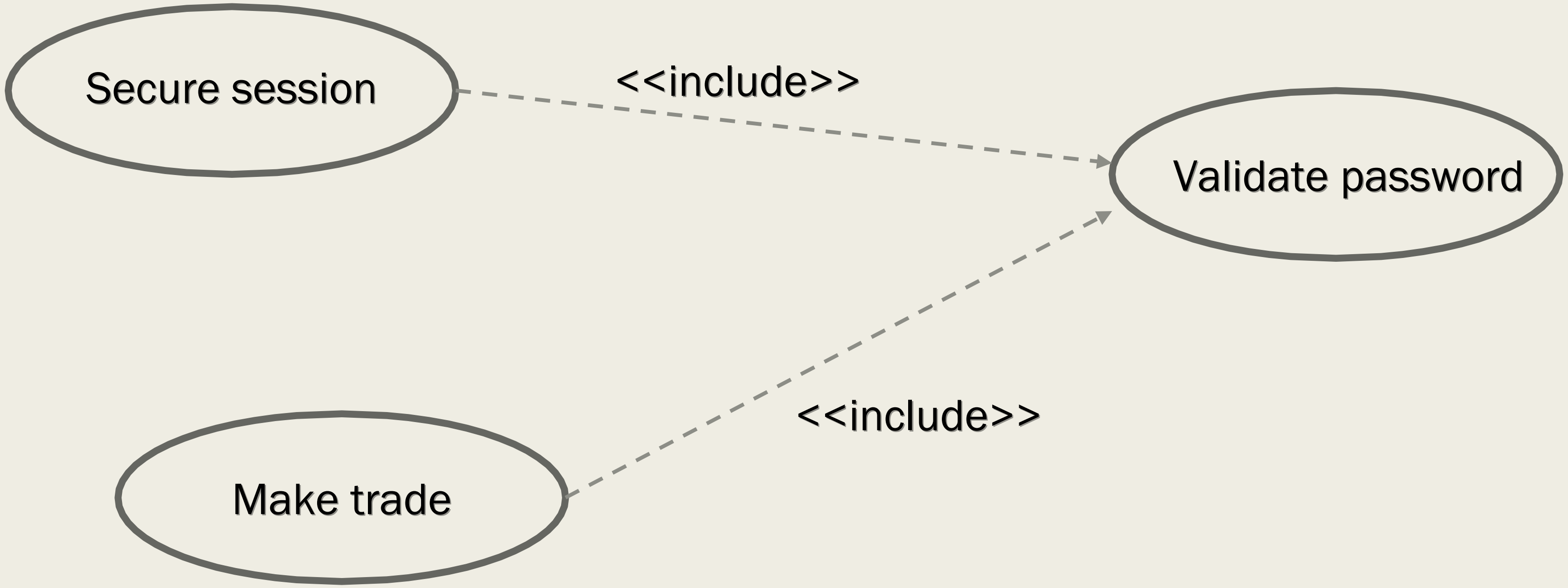
Login

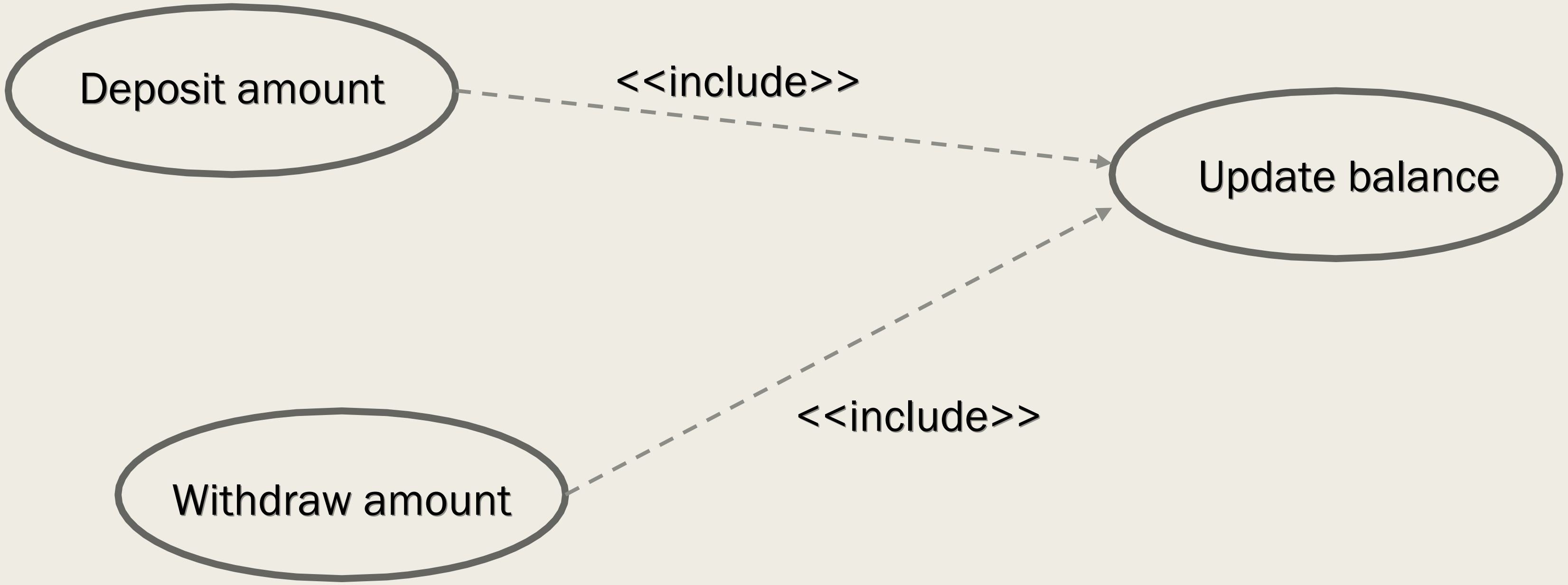
<<include>>

Checkorder

Inclusion use case

Base use case


Secure session
<<include>>
Validate password
<<include>>
Make trade

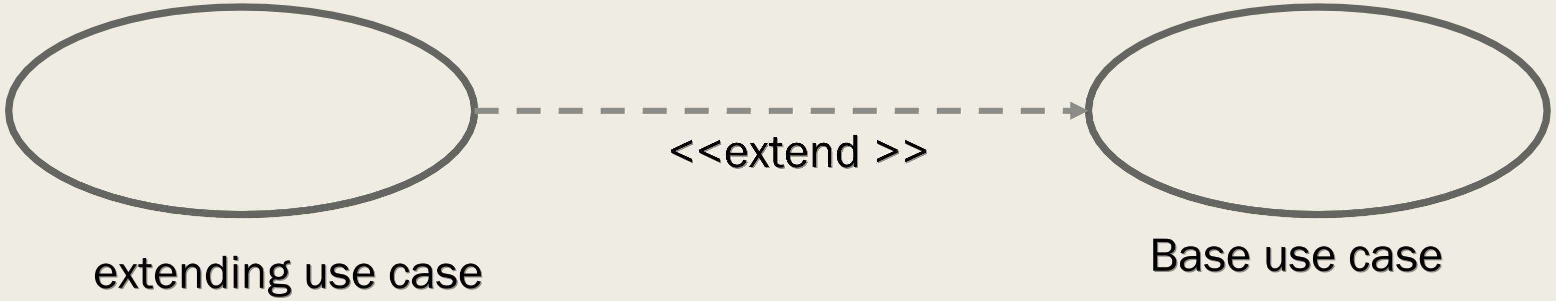

Deposit amount
<<include>>
Update balance
<<include>>
Withdraw amount

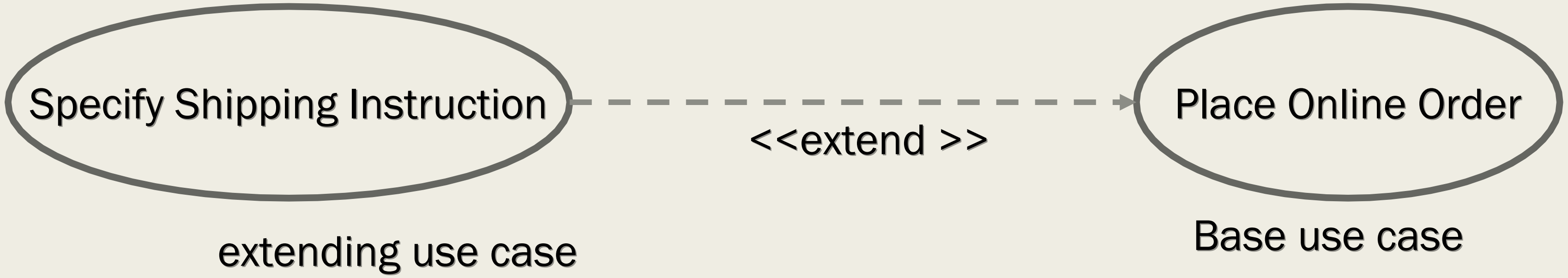
Few things to consider when using the <<include>> relationship.

- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.

Extend Relationship

- The extend relationship adds incremental behavior to a use case
- The extension adds itself to the base rather than the base explicitly incorporating the extension
- It represents the frequent situation in which some initial capability is defined and later features are added
- The extending use case is dependent on the base use case; it literally extends the behavior described by the base use case. The base use case should be a fully functional use case in its own right without the extending use case's additional functionality.
- The include and extend relationships both add behavior to a base use case
- The base use case represents the "must have" functionality of a project while the extending use case represents optional (should/could/want) behavior.
- The UML notation for extend relationship is a dashed arrow from the extension use case to the base use case
- The keyword <<extend>> annotates the arrow


extending use case
<<extend >>
Base use case



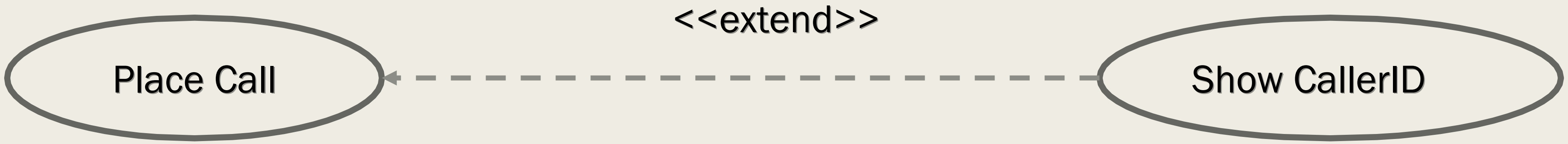
Specify Shipping Instruction

<<extend >>

Place Online Order

extending use case

Base use case



Place Call

<<extend>>

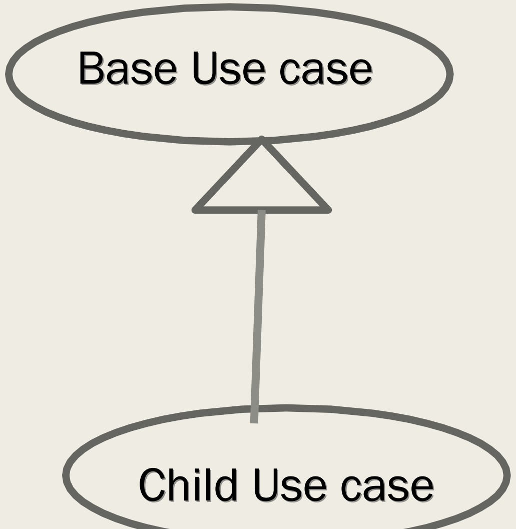
Show CallerID

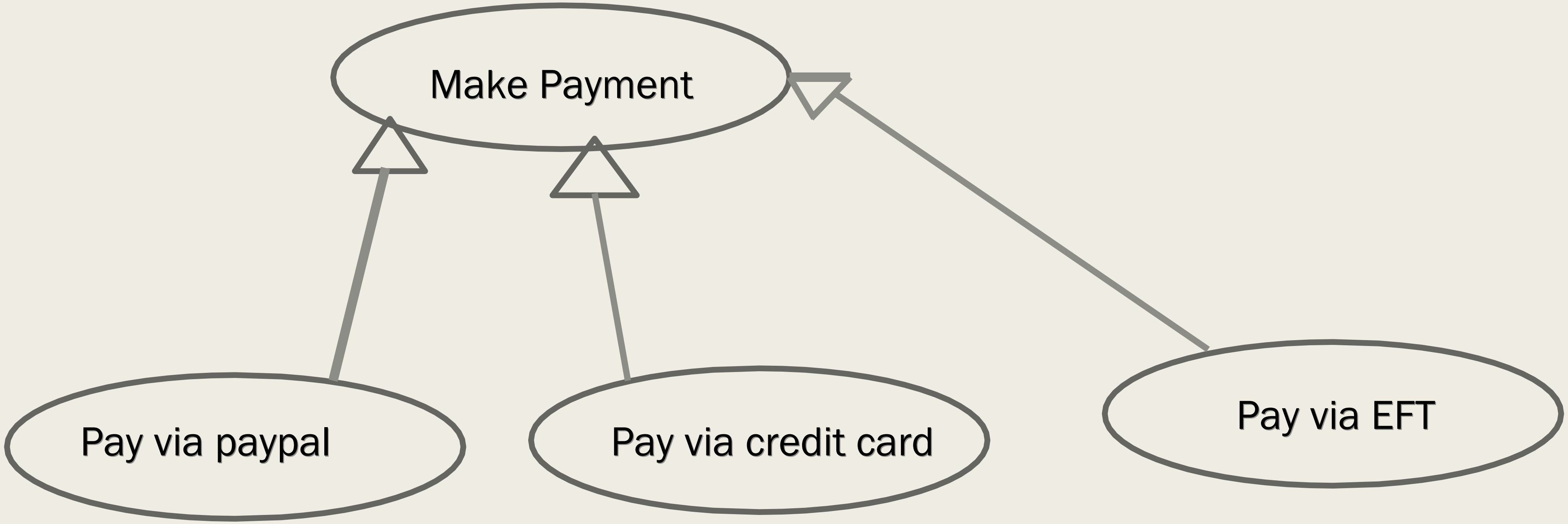
Few things to consider when using the <<extend>> relationship.

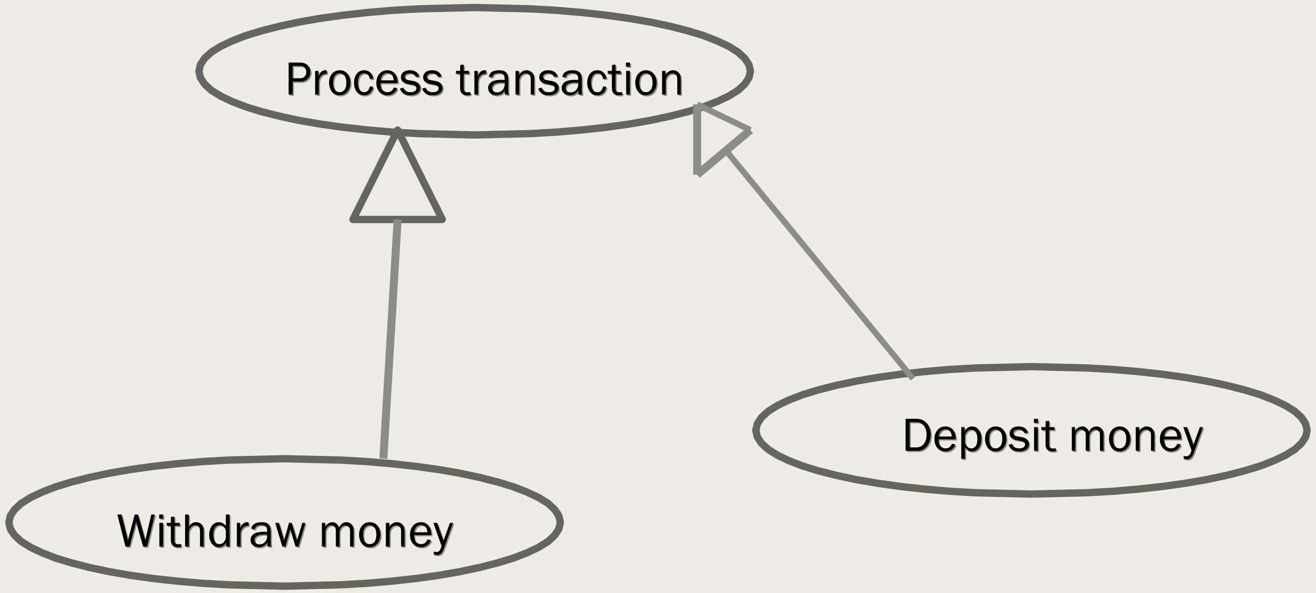
- Extend relationship is optional.(In this way, you separate optional behavior from mandatory behavior in your model.)
- subflow is executed only under certain (sometimes exceptional) conditions.

Generalization

- Generalization can show specific variations on a general use case
- A parent use case represents a general behavior sequence
- Child use cases specialize the parent by inserting additional steps
- UML notation for generalization is arrow with its tail on the child use case and a triangular arrowhead on the parent use case
- A parent use case may be abstract




Make Payment
Pay via paypal
Pay via credit card
Pay via EFT


Process transaction
Withdraw money
Deposit money

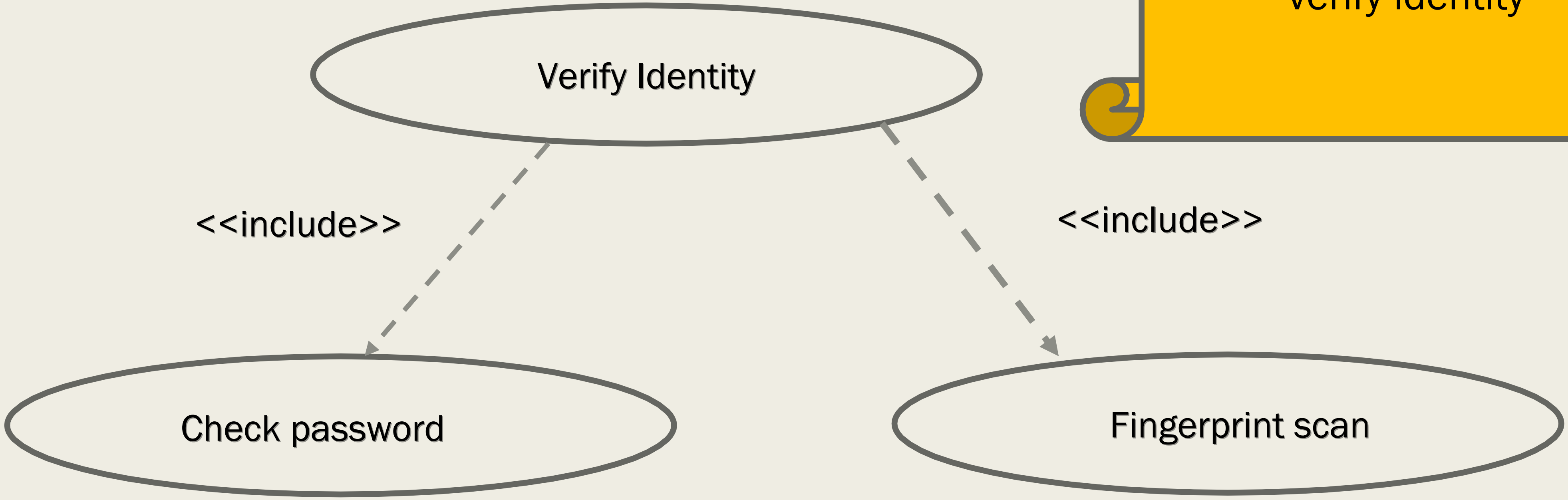
Guidelines for Use case Relationships

- Use case Generalization
  - *if use case comes in several variations, model the common behavior with an abstract use case and specialize each of the variations*
  - *Do not use generalization simply to share a behavior fragment*
- Use case Inclusion
  - *If a use case includes a well- defined behavior fragment that is likely to be useful in other situations, define a use case for the behavior fragment and include it in the original use case*

- Use case extension
  - *If it is possible to define a meaningful use case with optional features, then model the basic behavior as a use case and add features with extend relationship*
  - *This permits the system to be tested and debugged without the extensions, which can be added later*
- Include relationship vs. extend relationship
  - *The include and extend relationship can both factor behavior into smaller pieces*
  - *The include relationship implies that the included behavior is a necessary part of a configured system*
  - *The extend relationship implies that a system without the added behavior would be meaningful*

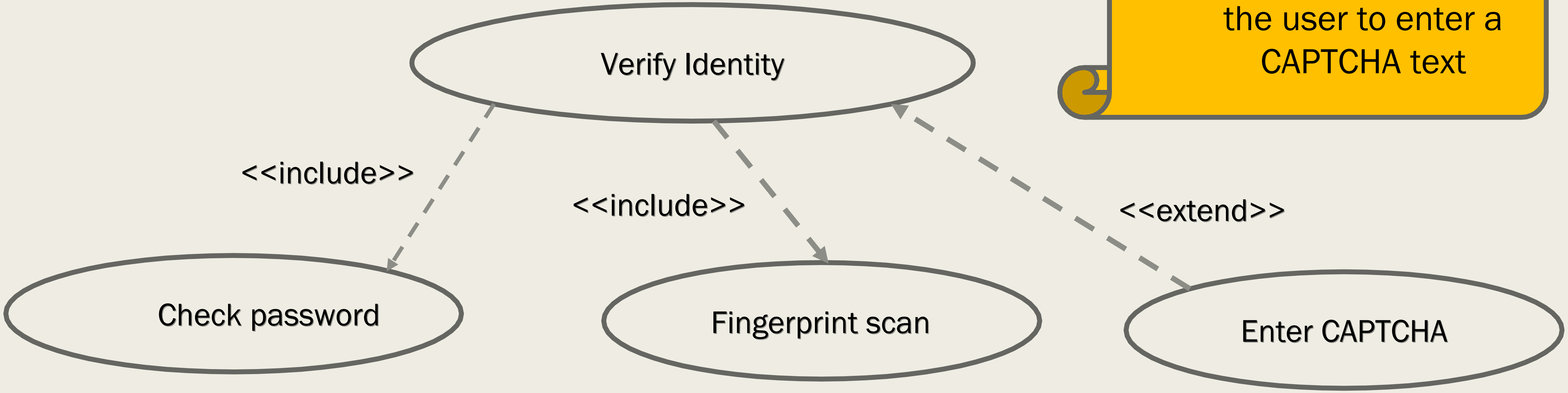
summary

- Include Relationship



BOTH a Password AND a Fingerprint Scan are required to verify identity

- Extend Relationship



Verify Identity

<<include>>

<<include>>

<<extend>>

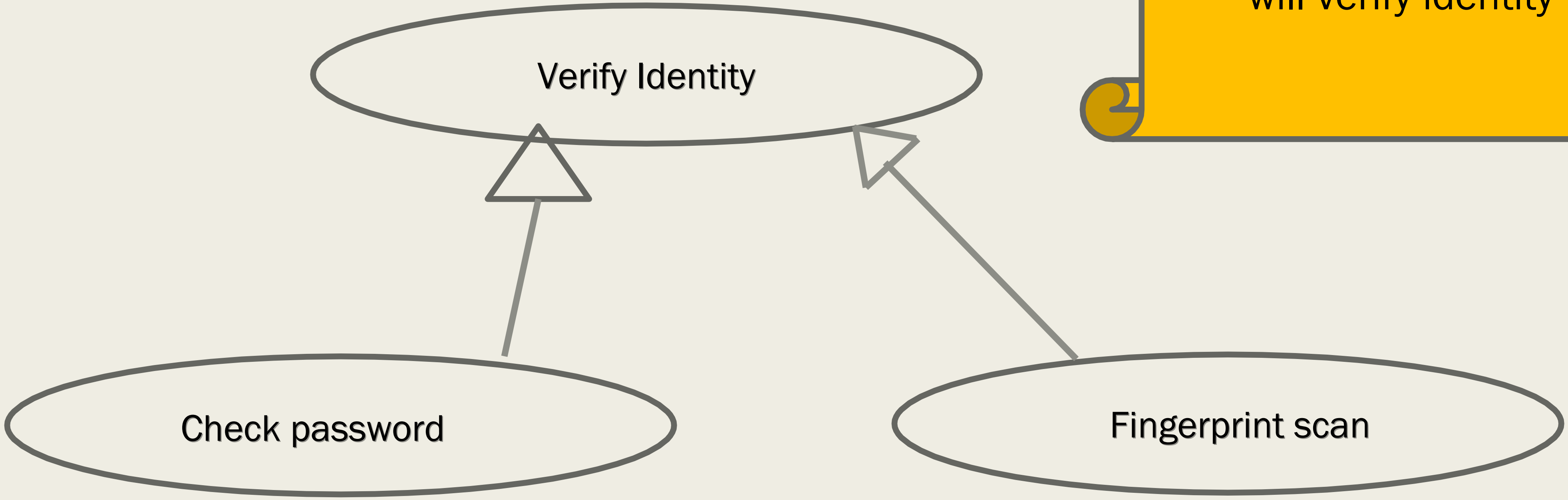
Check password

Fingerprint scan

Enter CAPTCHA

In additon to Password, and Fingerprint Scan, SOMETIMES the system will prompt the user to enter a CAPTCHA text

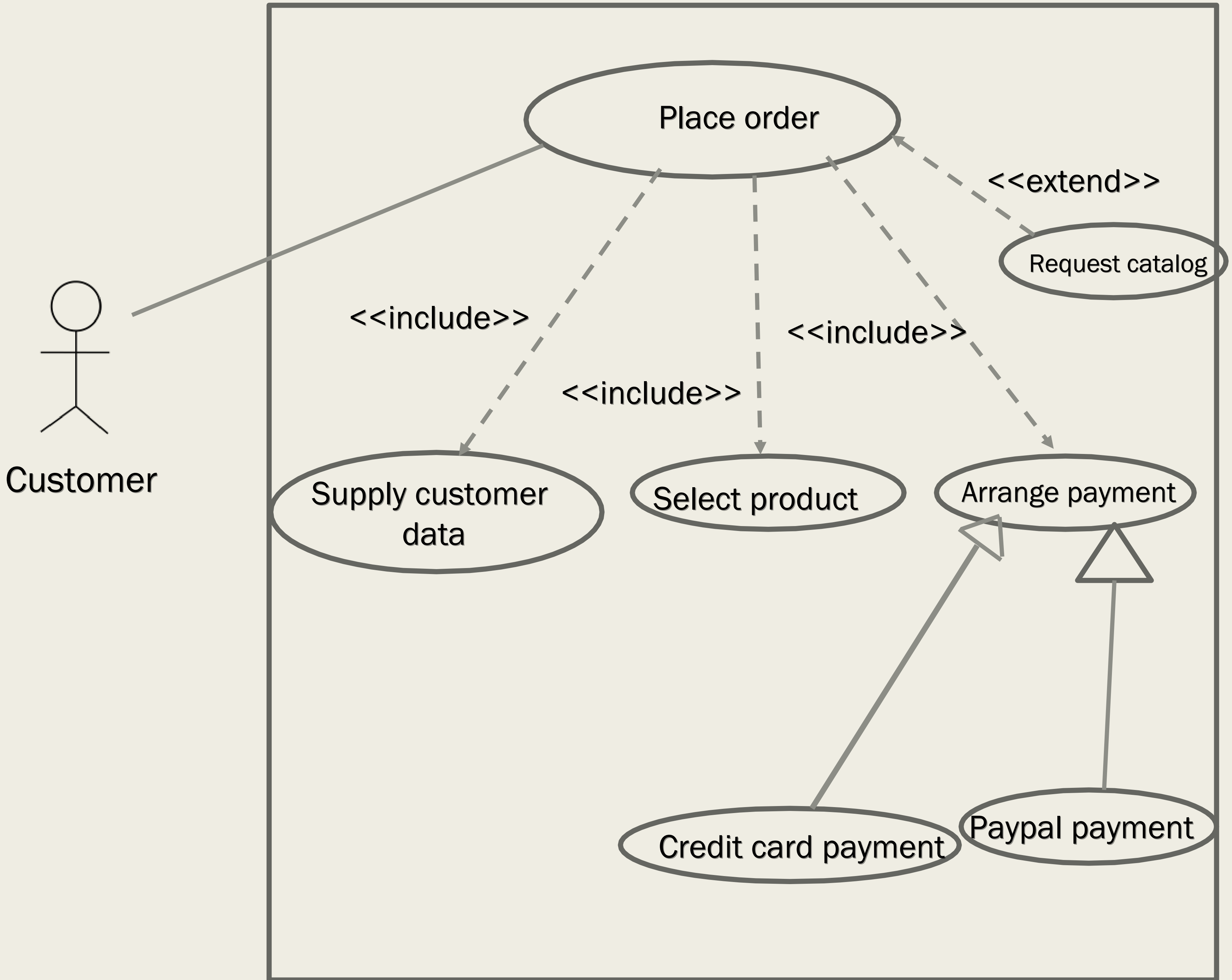
- Generalization Relationship



EITHER a Password
OR
a Fingerprint Scan
will verify identity

Combination of Use case Relationships

- A single diagram may combine several kinds of use case relationships


Place order
<<extend>>
Request catalog
<<include>>
<<include>>
<<include>>
Customer
Supply customer data
Select product
Arrange payment
Credit card payment
Paypal payment

Procedural Sequence Models

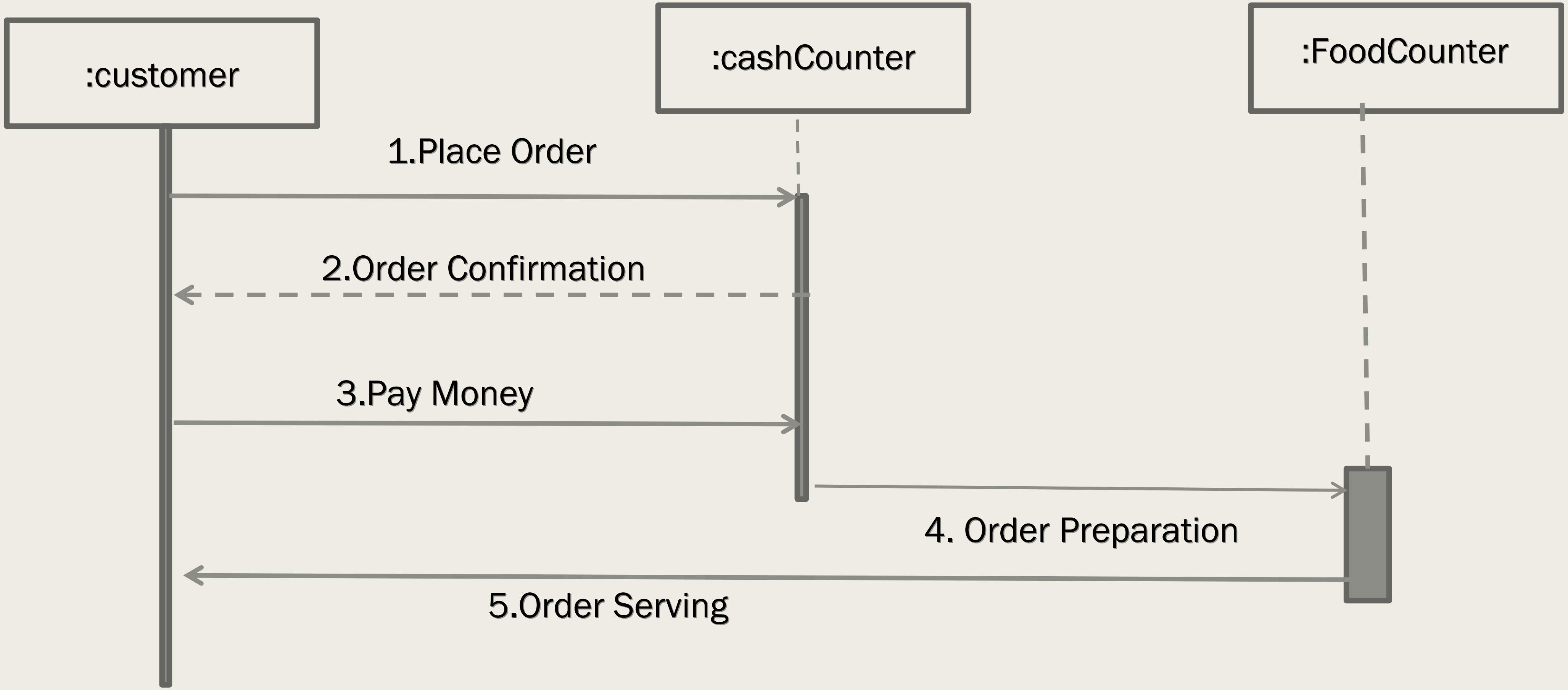
- The UML has elaborations for sequence diagrams to show procedure calls.

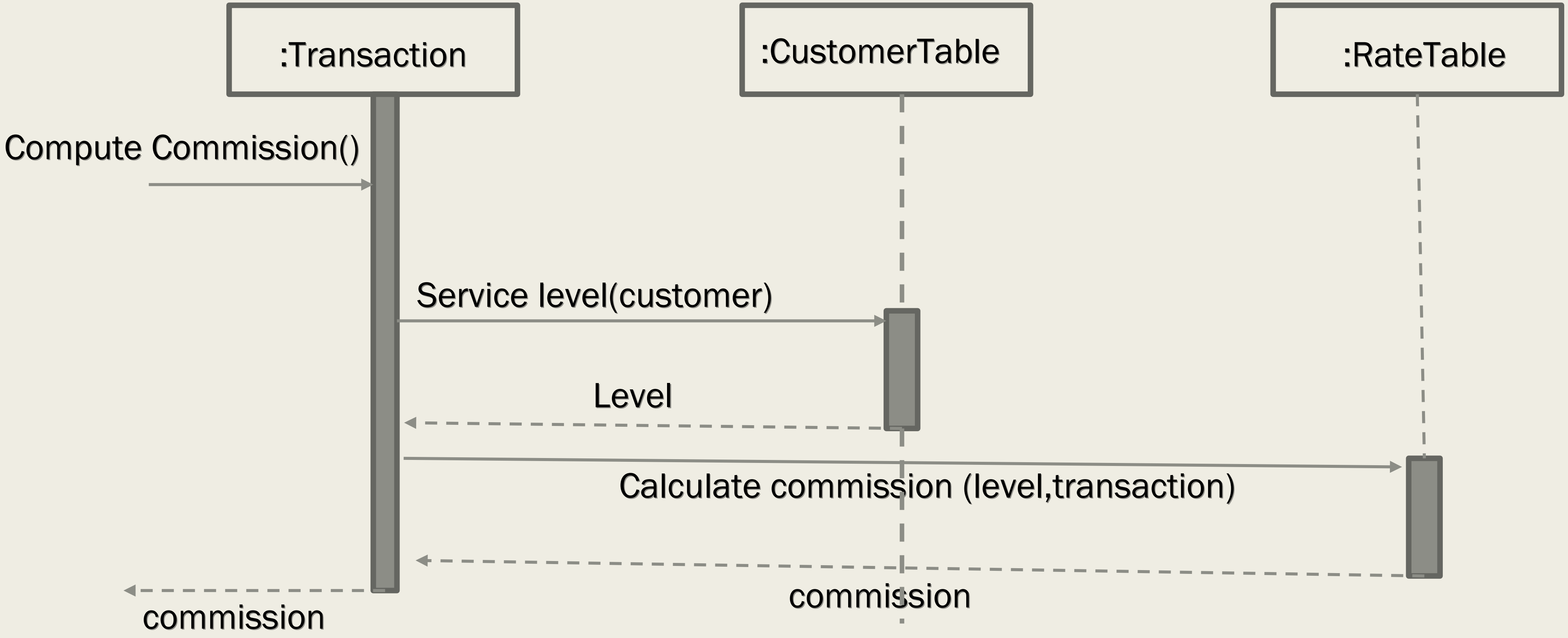
Sequence Diagrams with Passive Objects

- ***With procedural code all objects are not constantly active***
- Most objects are passive and do not have their own threads of control.
- A passive object is not activated until it has been called
- Once the execution of an operation completes and control returns to the caller the passive object becomes inactive

- The UML shows the period of time for an object’s execution as a thin rectangle this is called the activation or focus of control
- Activation shows the time period during which a call of a method is being processed, including the time when the called method has invoked another operation.
- The period of time when an object exists but is not active is shown as a dashed line
- The entire period during which the object exists is called the lifeline

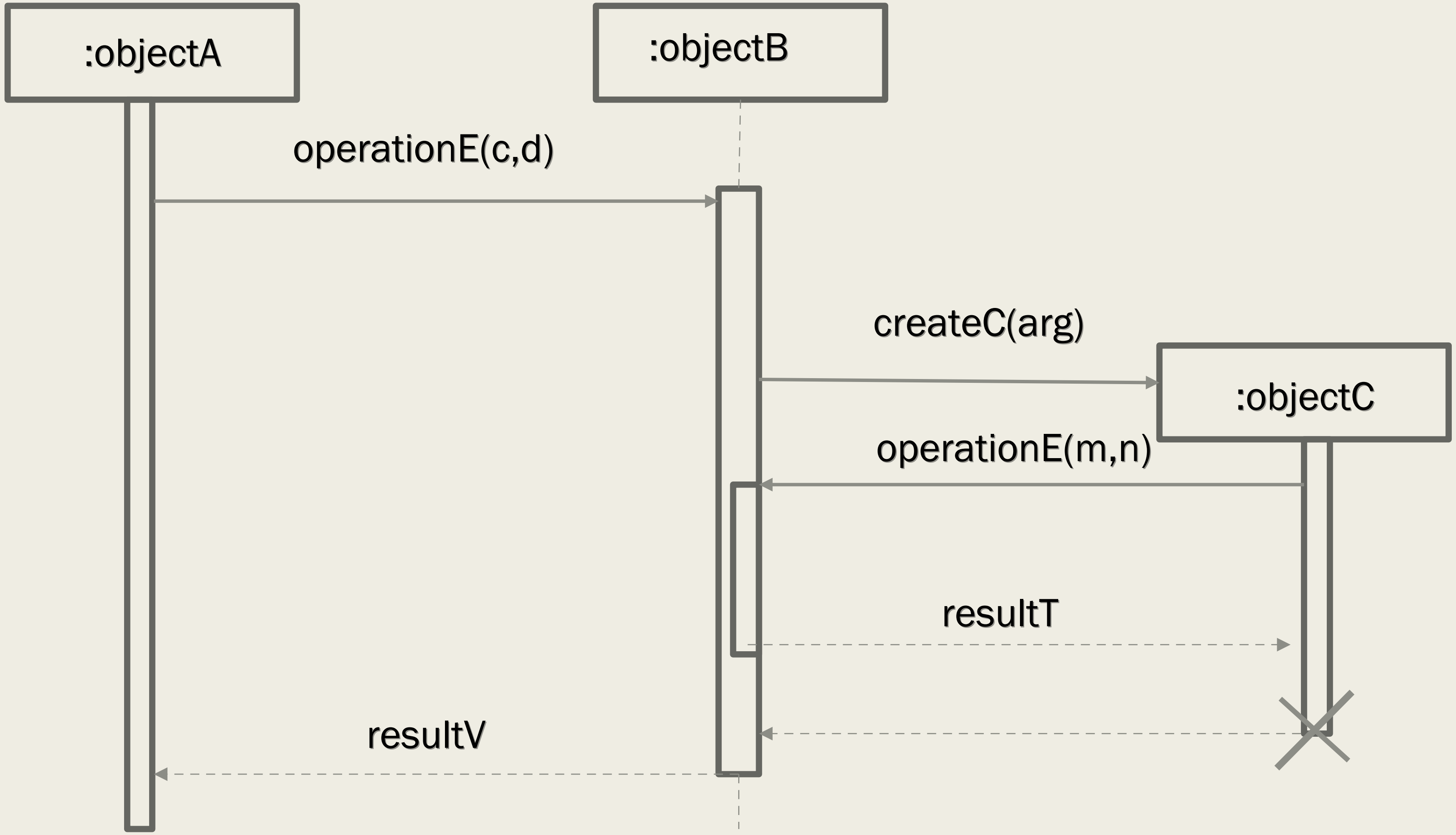
Example:


:customer
:cashCounter
:FoodCounter
1.Place Order
2.Order Confirmation
3.Pay Money
4. Order Preparation
5.Order Serving


:Transaction
:CustomerTable
:RateTable
Compute Commission()
Service level(customer)
Level
Calculate commission (level,transaction)
commission
commission

Sequence Diagrams with Transient Objects

- Object*A* is an active object that initiates an operation.
- ObjectB is passive object that exists during the entire time but not active for whole time
- objectC is created and destroyed during the time shown on the diagram
- The notation for a call is an arrow from the calling activation created by the call.
- Activation, therefore, has a call arrow coming into its top and a return arrow leaving its bottom.
- If an object does not exist at the beginning of a sequence diagram, then it must be created during the sequence diagram.

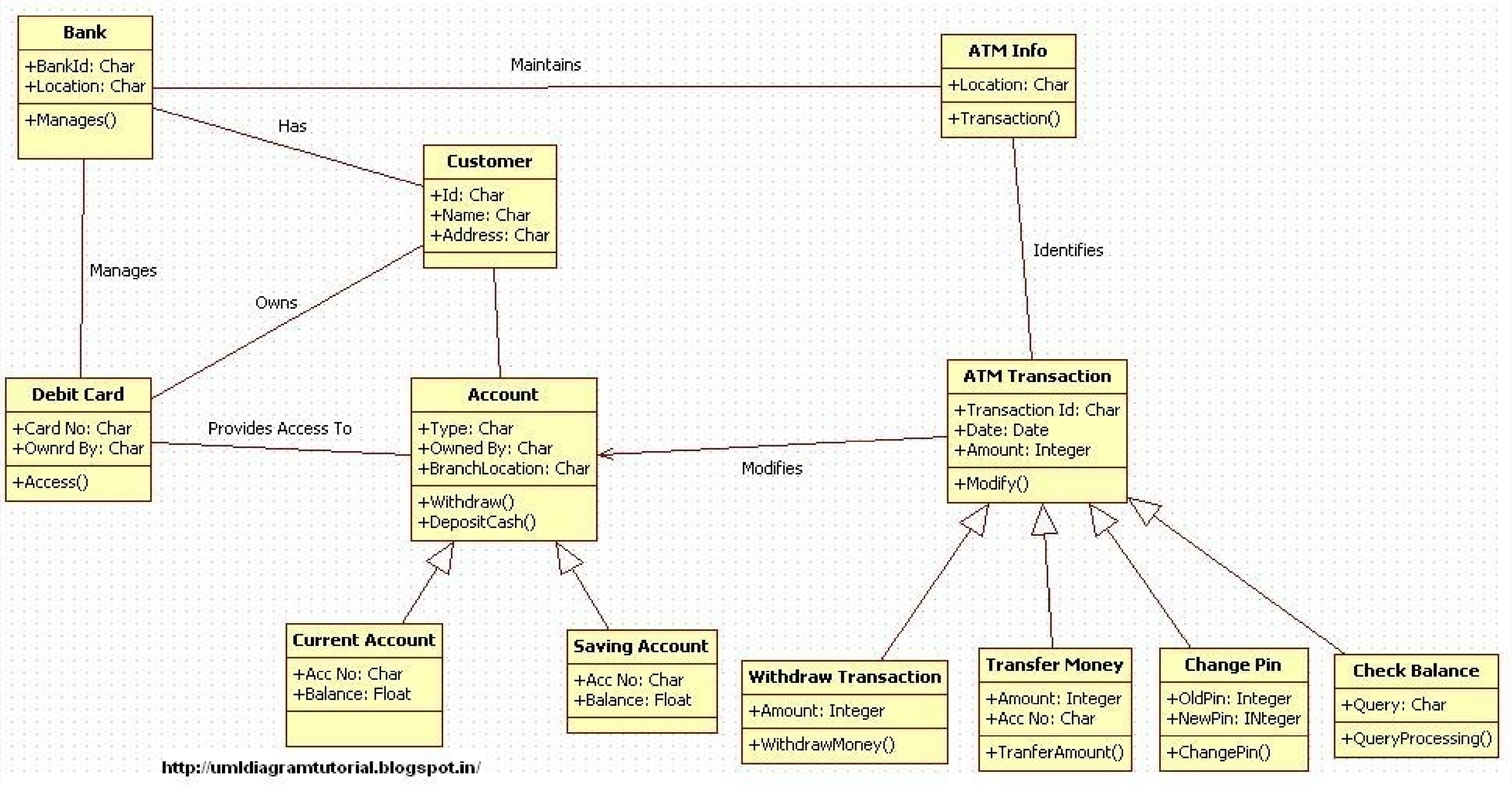

:objectA
:objectB
operationE(c,d)
createC(arg)
:objectC
operationE(m,n)
resultT
resultV

Guidelines for Procedural Sequence Models

- Active vs. passive objects
  - *Differentiate between active and passive objects*
  - *Most passive objects lack their own thread of control*
- Advanced features
  - *Advanced features can show the implementation of sequence diagrams*
  - *Be selective in using these advanced features*
  - *Implementation details can be shown only for difficult or important diagrams*

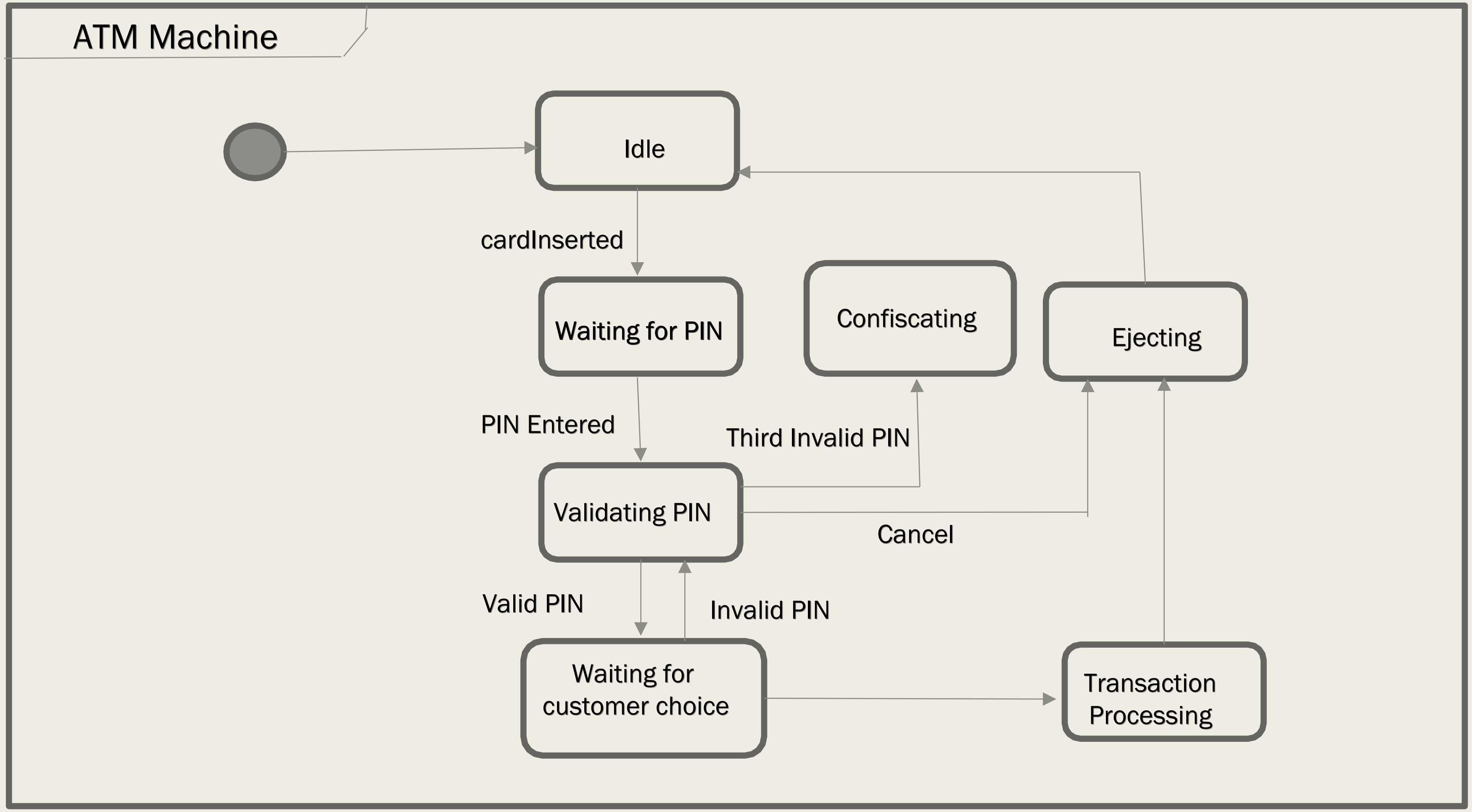
Example: ATM transaction

Class Model



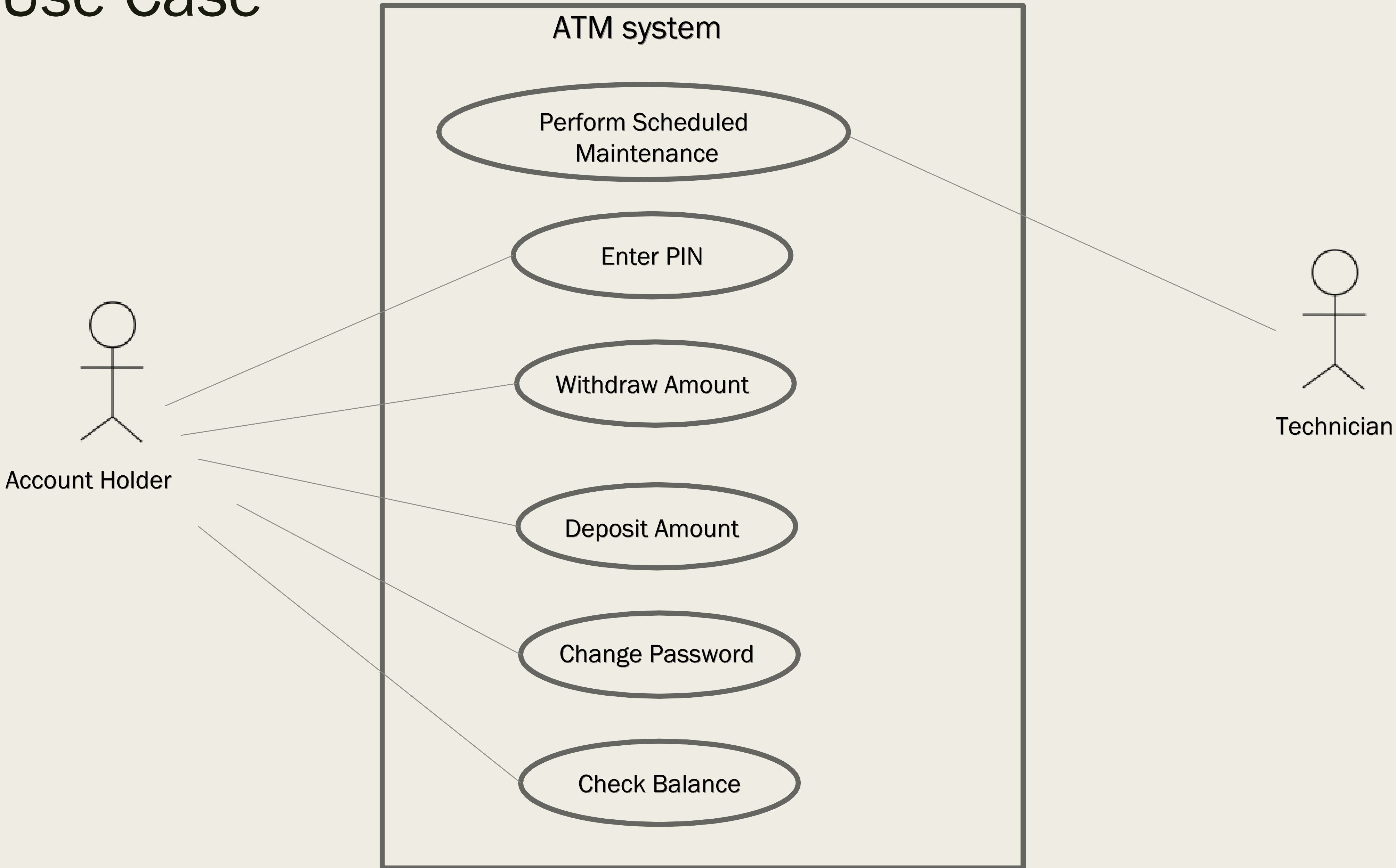
http://umldiagramtutorial.blogspot.in/

State Model

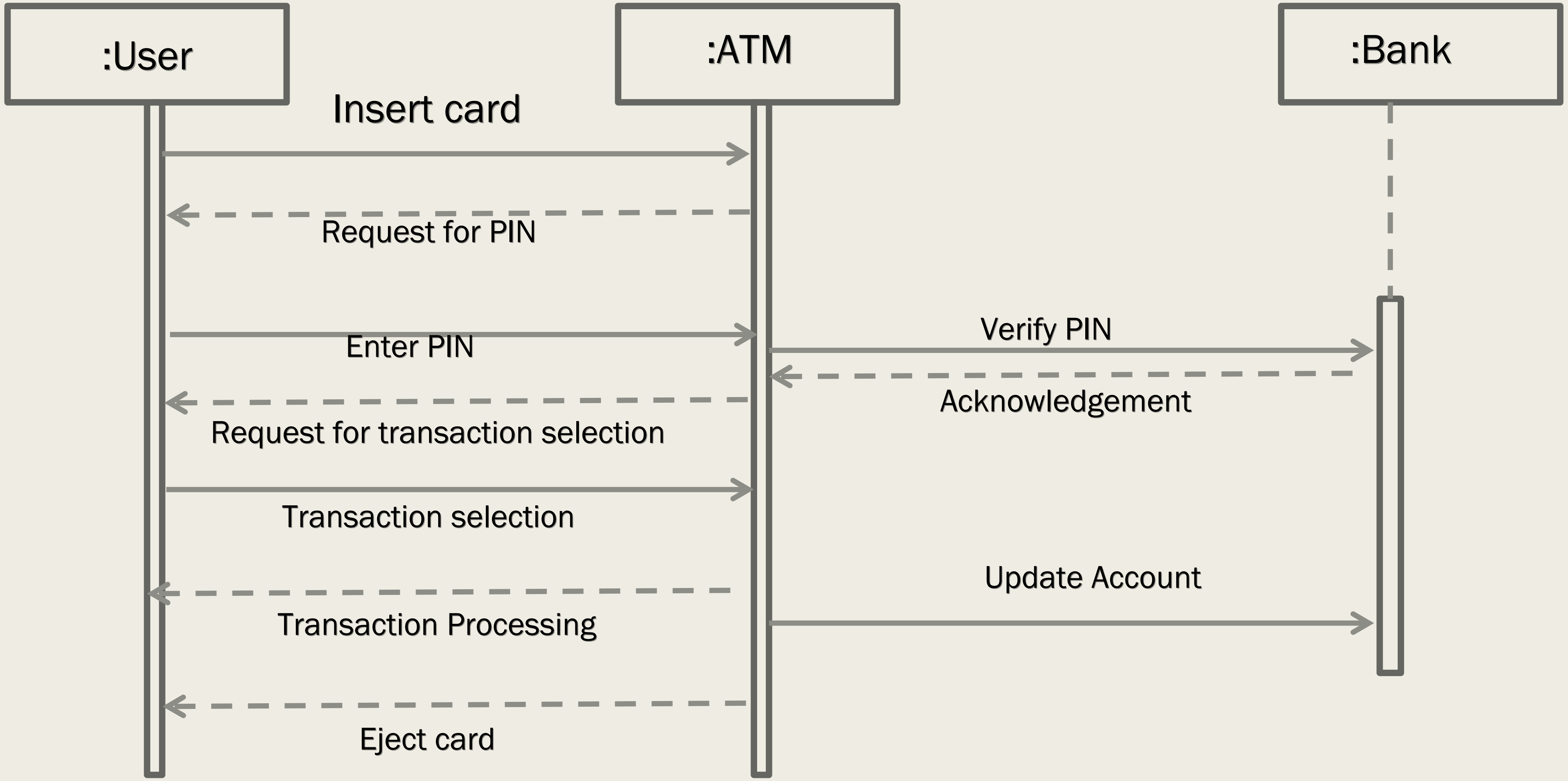

ATM Machine
Idle
cardInserted
Waiting for PIN
Confiscating
Ejecting
PIN Entered
Third Invalid PIN
Validating PIN
Cancel
Valid PIN
Invalid PIN
Waiting for customer choice
Transaction Processing

Interaction Model

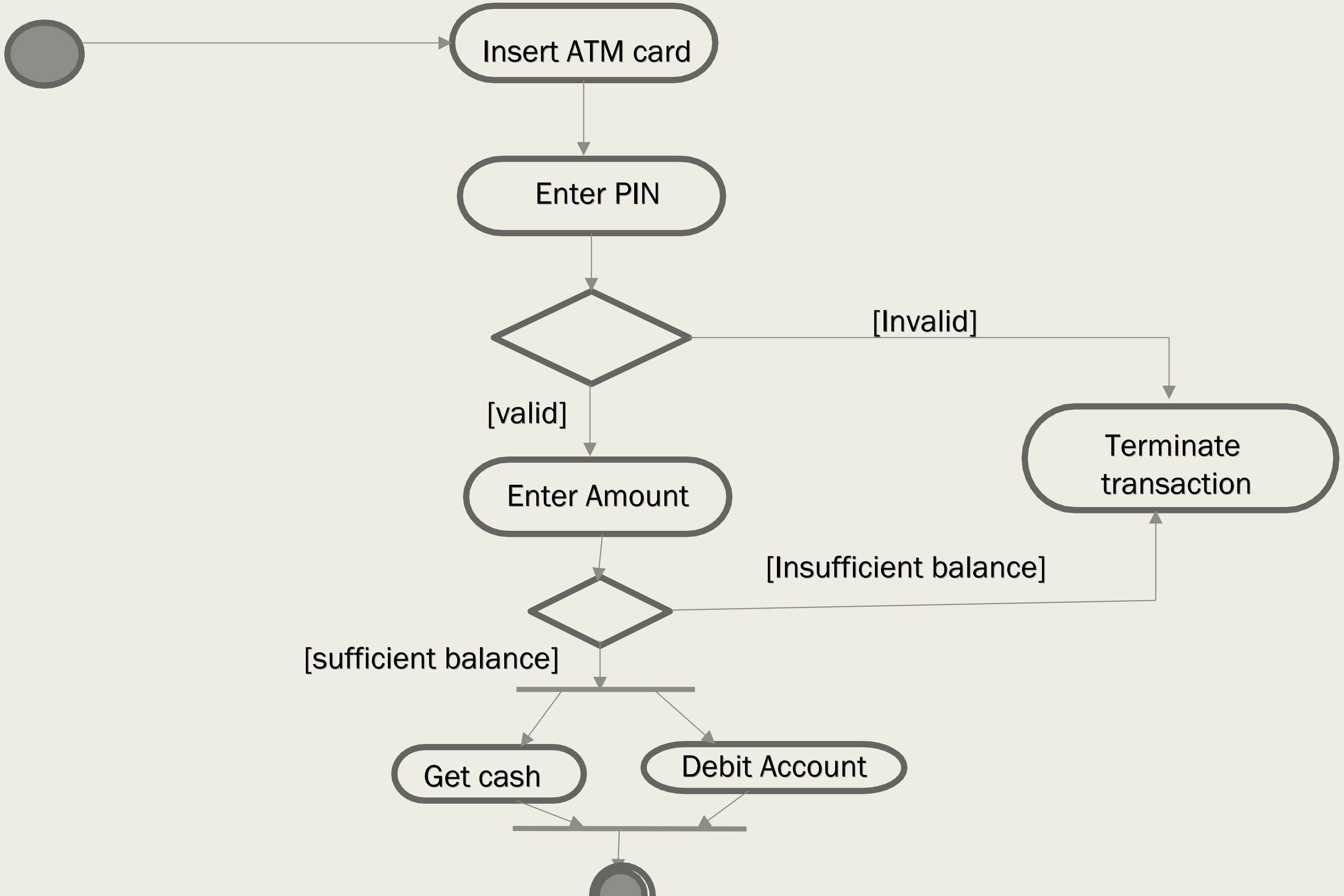
Use Case



Sequence Diagram



Activity Diagram



Thank you