AUTOMATION OF BANKING SYSTEM

CONTENTS

LIST OF DIAGRAMS

Page No.

1. PROBLEM STATEMENT

Here's the problem statement for the Automation of Banking System that is to be implemented: Design the software to support a computerized banking network including both human cashiers and mobile clients 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.

Mobile banking, Internet Banking and Automatic teller machines communicate with a central computer that clears transactions with the appropriate banks. The Account Holder Connects to the Bank server via his mobile by giving the username, password. The bank Machine accepts card information, interacts with the user, communicates with the central system to carry out the transaction, and dispenses cash and prints receipts. The system requires appropriate record keeping and security provisions. The system must handle allow to accesses the same account correctly with the same mobile identity.

ANALYSIS

USE CASE VIEW:

2. IDENTIFICATION OF ACTORS

Actors represent system users. They help to delimit the system and give a clear picture of what the system should do. It is important to note that an actor interacts with, but has no control over the use cases. An actor is someone or something that:

- interacts with or uses the system
- provides input to & receive information from the system
- is external to the system and has no control over the use cases

An actor can be represented as shown below:
(Actor diagrams)

Administrator

(from Actors)

The following questions should be answered to identify the actors:

1. Who is interested in a certain requirement?
2. Where in the organization the system is used?
3. Who will benefit from the use of the system?
4. Who will supply the system with the information, use this information and remove this information?

5. Who will support and maintain the system?
6. Does the system use the external resource?
7. Does one person play several different roles?
8. Do several people play same role?
9. Does the system interact with the legacy system?

Actors identified in the system are:

Service Provider

Service provider (Mobile network) is responsible in this banking system. And he is responsible for allowing the users to connect to bank systems. (if the system has any technical and network problems) if there is any temporary network problem he need to inform it to the mobile client immediately.

System Operator

Operator is responsible in this banking system. And he is responsible for the system startup, and the system shut down (if the system has any technical and network problems).

Customer

Customer is the person, who is interacting with the system whenever he wants to make a transaction. The interaction is in the sense for withdrawal, deposit, transfer, check balance, and ministatement (for the previous transactions).

BankComputer

Bank Computer is the system that is directly interacting with the Service Provider system (mobile service provider) to make the transactions successful and for the maintenance of the database. In the network and the BankComputer are connected. To check the user is authorized and username and password is valid, for those the database is responsible. Like wise the functionality can be taken place in main server system.

3. IDENTIFICATION OF USECASES AND SUB USE CASES

In its simplest form a use case can be described as a specific way of using the system from a user's perspective. A more detailed description might characterize a use case as:

- A pattern of behavior the system exhibits
- A sequence of related transactions performed by an actor and the system
- Delivering something of value to the actor.

Use case provides a means to:

1. Capture system requirements
2. Communicate with the end users and domain experts
3. Test the system

Use cases are the best discovered by examining the actors and defining what the actor will be able to do with the system. Since all needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies all ways of using the system.

UML notation for the use case:

1. An ellipse containing the name of the use case inside it.

Error: Reference source not found

2. An ellipse containing the name of the use case below it

The following questions are to be answered to identify the use cases:

1. What are the tasks of each actor?
2. Will any actor create, store, change, remove, or read information in the system?
3. What use cases will store, change, remove or read this information?
4. Will any actor need to inform the system about sudden external changes?
5. Does any actor need to be perform about certain occurrences in the system?
6. What use cases will support and maintain the system?
7. Can all functional requirements be performed by the use cases?

Use cases identified in the system are:

- Login
- ChangePassword
- Transaction
- Withdrawal
- Deposit
- Transfer
- CheckBalance
- MiniStatement
- TypeOfAccount
- SavingsType
- CurrentType
- ServiceProvider
- SystemStatus
- SystemStartUp
- SystenShutDown

4. FLOW OF EVENTS

A flow of events is a sequence of transactions (or events) performed by the system. They typically contain very detailed information, written in terms of what the system should do, not how the system accomplishes the task. Flow of events are created as separate files or documents in your favorite text editor and then attached or linked to a use case using the files tab of a model element.

Flow of events should include:

- When and how the use case starts and ends
- Use case / actor interactions
- Data needed by the use case
- Normal sequence of events for the use case
- Alternate or exceptional flows

Flow of events in our system is :(for one use case)

1. Post items:

1.1Brief Description:

Once customer has an account and he wants to make the transactions, First of all he has to check for network service. And then connect to the application type the username and password. Then the mobile in turn sends the information to the bank via service provider. If the customer is valid user then the system ask type of account, whether it is current or savings. Then ask the transaction type, it includes withdrawal, deposit, transfer, check Balance and mini statement. The system responds accordingly.

2. Flow of Events for Each Use case of Mobile Banking System Application:

1.0 Use Case Name: **System Status**

1.1 Brief Description: This use case started by operator and it includes installation of money and it shown the status of the system.

2.0 Flow of Events:

2.1 Basic Flow: the use case begins when the operator starts work, operator interacting with the bank computer.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **System Startup**

1.1 Brief Description: this is the use case it indicates that the system is in starting mode. It is the special type of generalized system status.

2.0 Flow of Events:

2.1 Basic Flow: it is started by operator.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: if system is in shutdown state then only system will be started.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **System Shutdown**

1.1 Brief Description: this is the use case it indicates that the system is in closed mode. It is the special type of generalized system status.

2.0 Flow of Events:

2.1 Basic Flow: it is started by operator.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: if system is in running state then only system will be shutdown.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **Login**

1.1 Brief Description: Customer makes use of his mobile to connect to the bank server so that he must login with appropriate UserName and Password.

2.0 Flow of Events:

2.1 Basic Flow: This use case depends on the system status use case and is associated with the bank computer to verify is the authorized user or not? And it has the specialized use case Change PIN.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: the status of system should be in startup state.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **Change Password**

1.1 Brief Description: Customer makes use of Mobile to connect the Bank server so that he has freedom to change Password whenever he wants to secure from others (great protection).

2.0 Flow of Events:

2.1 Basic Flow: This use case depends on the system status use case and is associated with the bank computer for the sake of customer.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: the status of system should be in startup state and login should be taken place.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **Transaction**

1.1 Brief Description: this is the use case it indicates the type of transaction. It is the general types of specialized use cases like withdrawal, deposit, transfer, check Balance and MiniStatement.

2.0 Flow of Events:

2.1 Basic Flow: it is started by the customer.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: system should be in login state.

5.0 Postconditions:

6.0 Extension Points: Extension is done with the specialized use cases like withdrawal, deposit, transfer, checkBalance and MiniStatement.

1.0 Use Case Name: **withdrawal**

1.1 Brief Description: this is use case to withdraw the money from the system. it includes the other use case type of account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the transaction as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: the amount withdrawal should be less than the Balance.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **Deposit**

1.1 Brief Description: this is use case to deposit the money to the system. It includes the other use case type of account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the transaction as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **Transfer**

1.1 Brief Description: this is use case to transfer the money to another account from the same account. It includes the other use case type of account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the transaction as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions: the amount to transfer should be less than the amount in the account.

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **CheckBalance**

1.1 Brief Description: this is use case to check the balance of the system. it includes the other use case type of account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the transaction as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **MiniStatement**

1.1 Brief Description: this is use case to show the last 15 transactions. it includes the other use case type of account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the transaction as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **AccountType**

1.1 Brief Description: This is a use case indicates that the type of account. It has two sub use cases savings and account.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to transactions as the dependency include.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **savings**

1.1 Brief Description: This is a use case indicates that the type of account (savings).

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the type of account as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

1.0 Use Case Name: **current**

1.1 Brief Description: This is a use case indicates that the type of account (current).It indicates that for the regular users.

2.0 Flow of Events:

2.1 Basic Flow: this use case started by the customer. It is having the relation to the type of account as generalized.

2.2 Alternative Flow:

3.0 Special Requirements:

4.0 Preconditions:

5.0 Postconditions:

6.0 Extension Points:

5. USECASE DIAGRAMS

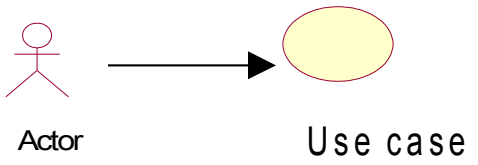
Use case diagrams depict the system behavior (use cases).These diagrams present a high level view of how the system is used as viewed from an outsider's (actor's) perspective. A use case diagram may depict all or some of the use cases of a system.

A use-case diagram can contain:

- actors ("things" outside the system)
- use case (system boundaries identifying what the system should do)
- Interactions or relationships between actors and use case in the system including the associations, dependencies and generalizations.

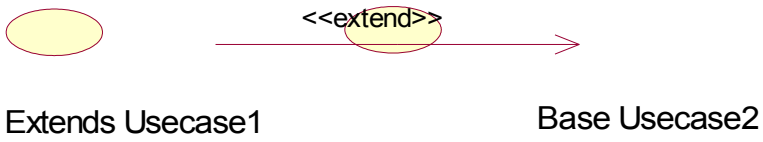
Uses association:

The uses association occurs when we are describing our use cases and notice that some of them have sub flows in common to avoid describing a sub flow more than once in several use cases, you can extract the common sub flow and make it a use case of its own. This new use case then can be used by other use cases. The relationships among the other use cases and this new extracted use case are called uses association.



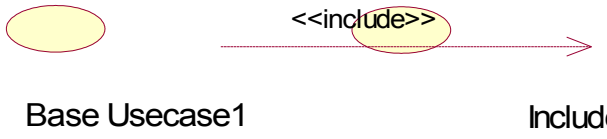
Extends association:

An extends association is a stereo typed association that specifies how the functionality of one use case can be inserted in to the functionality of another use case. Extend relationships between use cases are modeled as dependency by using the extend stereotype.

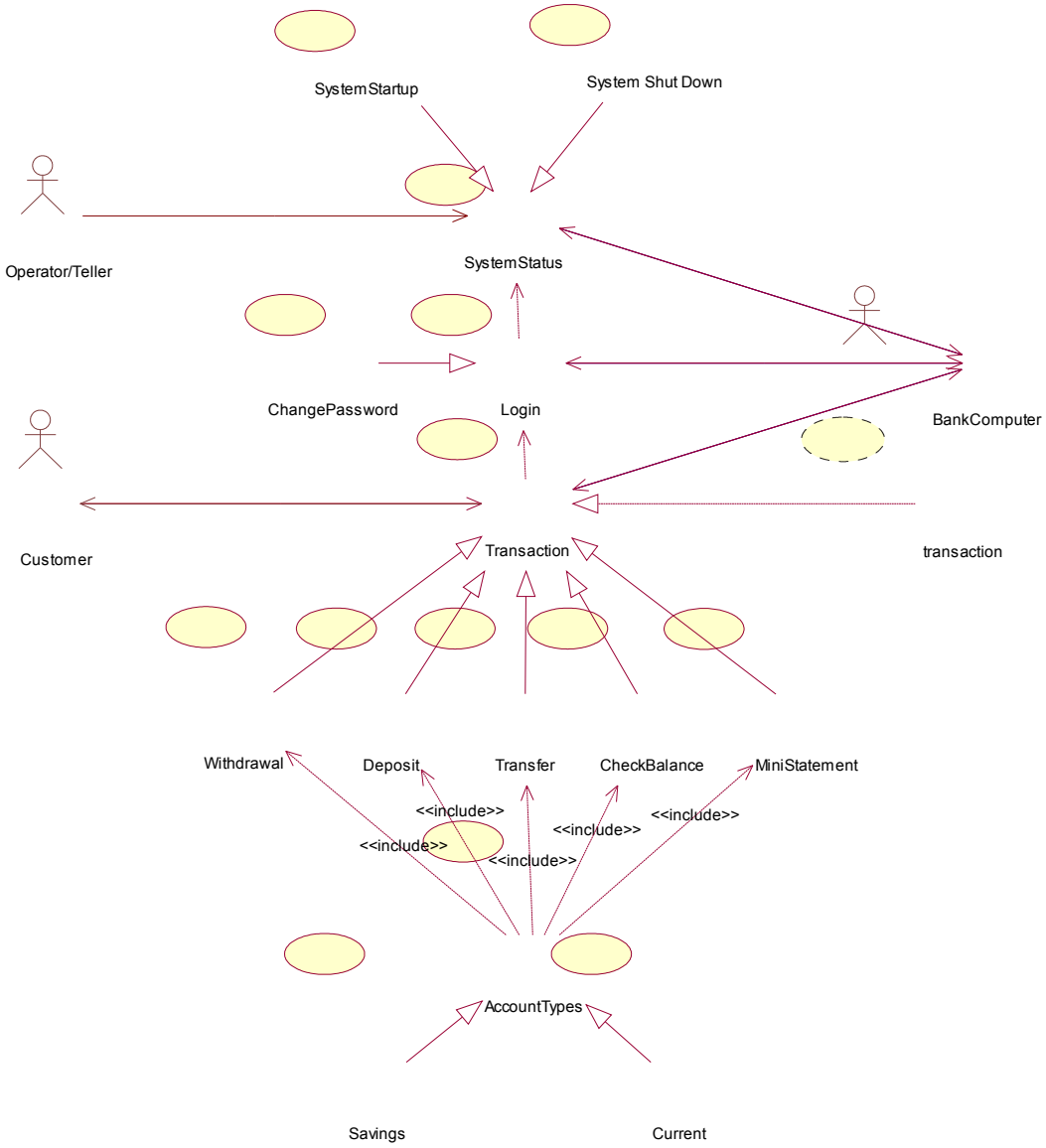


Includes association:

An include association is a stereo typed association that connects a base use case to an inclusion use case.



USE CASE Diagram



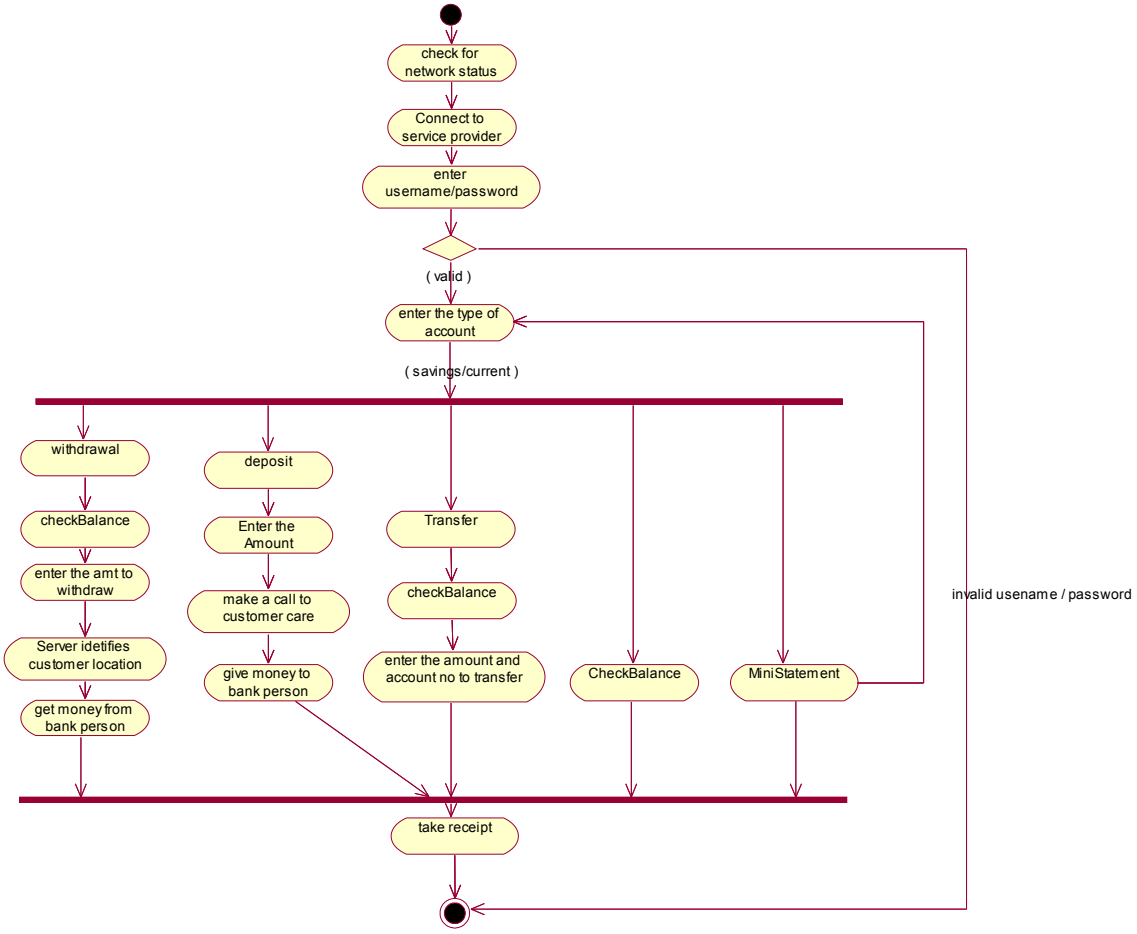
6. BUILD A BUSINESS PROCESS MODEL USING UML ACTIVITY DIAGRAM

Activity diagrams provide a way to model the workflow of a business process. An activity diagram is typically used for modeling the sequence of Activities in a process. Activity diagrams can model many different types of workflows. A software company could use activity diagrams to model a software development process.

The following tools are used on the activity diagram toolbox to model activity diagrams.

- Decisions: A decision represents a specific location on activity diagram when the workflow may branch based upon guard conditions.
- Synchronization: Synchronizations visually define forks and joins representing parallel workflow.
- Forks and Joins: A fork construct is used to model single flows. A join consists of two or more flows of control that unite into a single flow of control.
- States: "A state represents a condition or situation during the life of an object during which it satisfies some condition or waits for some event.
- Transitions: A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied.
- Start States: A start state (also called an "initial state") explicitly shows the beginning of a workflow.
- End States: An end state represents a final or terminal state.
- Swim Lane: A unique diagram feature that defines who or what is responsible for carrying out activity or state.
- Work Flow: Each activity represents the performance of a group of actions in a workflow.

Figure: Activity Diagram of Mobile Client Application



7. IDENTIFICATION OF ANALYSIS CLASSES

Identification of classes can be done by the noun phrase approach, the common class patterns approach, the use-case driven sequence/collaboration modeling approach and the classes' responsibilities and collaborators (CRC) approach.

(i) NOUN PHRASE APPROACH:

This approach was proposed by Rebecca Wirfs-Brock, Brain Wilker-son, and Luaren Wiener. In this method, we read through the requirements or use-cases looking for noun phrases. Nouns in the textual description are considered to be classes and verbs to be methods of the classes then, the nouns are listed, and divided in to three categories: relevant classes, fuzzy classes and the irrelevant classes.

(ii) COMMON CLASS PATTERN APPROACH:

It is based on the knowledge base of common existing classes. The candidate classes can be formulated using the following:

Concept Class: Concept is an understanding of our world.

Event Class: These are points in time that must be recorded.

Organization Class: It is a collection of people, resources, facilities or groups to which the users belong.

People class: The people class represents the different roles users play in interacting with the application.

Places Class: These are the physical locations.

Tangible things and Devices Class: It includes physical objects or groups of objects that are tangible**.**

(iii) USE-CASE DRIVEN APPROACH:

Here the scenarios are described in text or through a sequence of steps. It is a problem driven approach to object oriented analysis in which the designer first considers the problem at hand and not the relationship between objects.

(iv) CLASSES RESPONSIBILITIES AND COLLABORATORS:

This technique is used to identify the classes, responsibilities and therefore their attributes and methods. It is based on the idea that an object can either accomplish a certain responsibility by itself or it may require the assistance of other objects in which case it collaborates.

Out of these, the noun phrase is used to identify the classes to increase our understanding of the subject.

1. The initial list of noun phrases :(Candidate classes-may be)

BankComputer	Office Computer	Service Provider	Client
Bank Client	Account	Joint Account	Current Account
Savings Account	Transactions	Withdrawal	Deposit
Transfer	Check Balance	Mini Statement	PIN
Identification NO	Account No	Card No	Login
Password	Cash	Demand Draft	Cheque

2. Reviewing the irrelevant classes

It is safe to eliminate the irrelevant classes. The candidate classes must be selected from relevant classes and fuzzy classes. The following irrelevant classes can be eliminated:

Joint Account	Identification No	Account No

3. Reviewing the redundant classes and building a common vocabulary

Here the several classes appear more than once. Hence we eliminate the redundant classes

BankComputer = Office Computer	Client=Bank Client
Cash, DD, Cheque = Cash	Account, Joint Account = Account
Password, PIN = PIN	

4. Reviewing the classes containing adjectives

In this system we have no classes containing adjectives that we can eliminate.

5. Reviewing the possible attributes

The noun phrases used only as values should be restated as attribute classes.

In this system there are no Attribute classes.

6. The Final List of Classes is:

BankComputer	GPS System	Bank Client	Account
Current Account	Savings Account	Transaction	

8. IDENTIFICATION OF RESPOSIBILITIES OF EACH CLASS

CRC developed by Cunningham, Wilkerson, and Beck. Classes Responsibilities and Collaborators is a technique used for identifying classes' responsibilities and their attributes and methods. They also help in identifying the classes. Classes Responsibilities and Collaborators is based on the idea that an object either can accomplish a certain responsibility itself or it may require the assistance of other objects .If it requires the assistance of other objects, it must collaborate with those objects to fulfill its responsibility. By identifying an object's responsibilities and collaborators attributes and methods can be identified.

The Classes, Responsibilities, and Collaborators process consists of three steps:

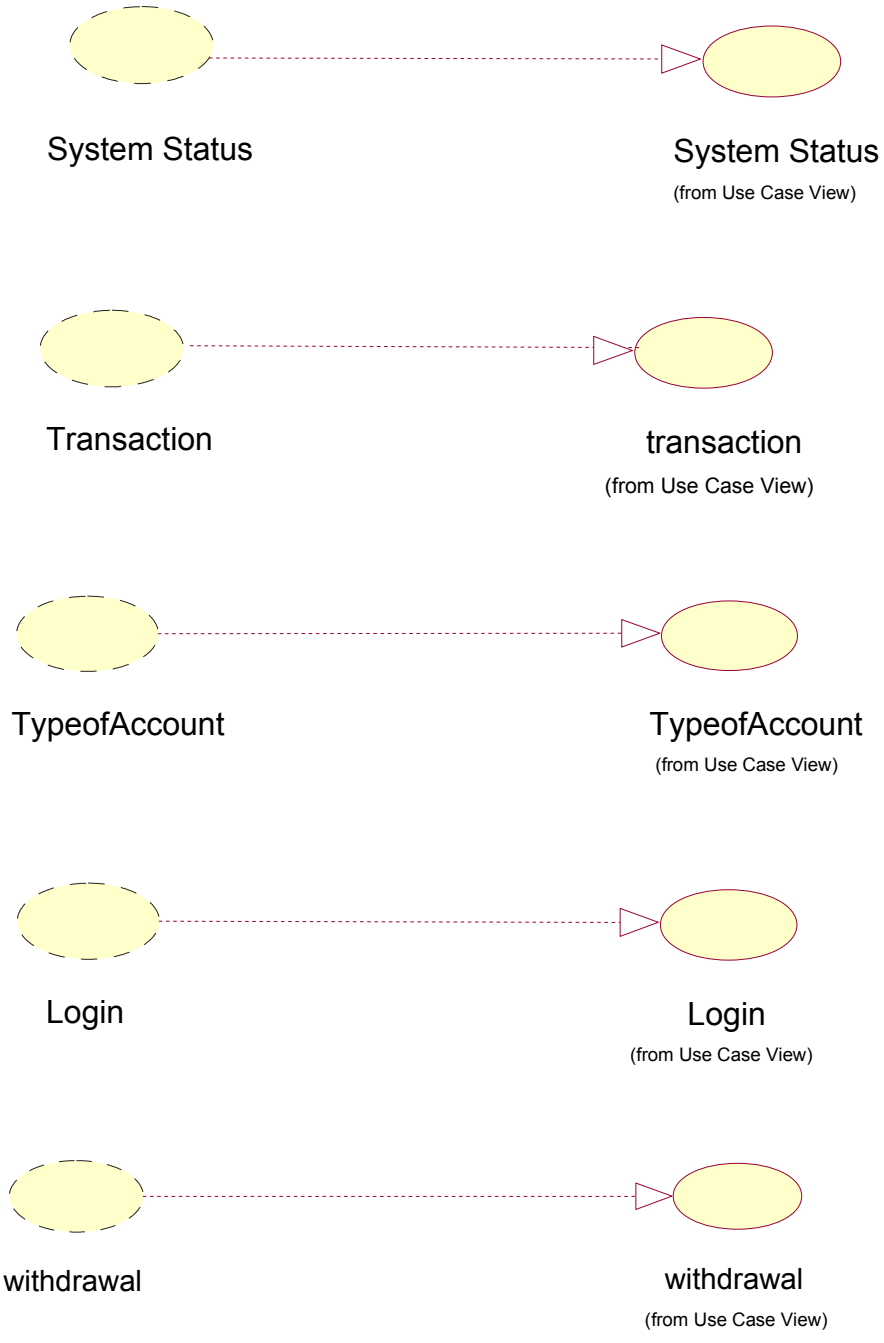
1. Identify classes' responsibilities(and identify classes)
2. Assign responsibilities
3. Identify collaborators

Class name	**Responsibilities**
BankComputer	BankSystem is a class. where aggrigation of two classes (GP system, Account) is done. and maintenance of the data can be from here only.
GPSystem	GPS-System is a class having the aggrigation relation to BankSystem.
BankClient	BankClient is a class is associated with Account class and GPS system class interface.
Account	Account class is associated to the classe BankClient and having aggregation relation to BankSystem. And it has transactional functions.
Current Account	CheckingAccount is a class and it is a sub class of Account class. It has the relation to the SavingsAccount.
Savings Account	SavingsAccount is a class and it is a sub class of Account class.
Transaction	Transaction is a class which is having the details of the transactions like transaction Id, transaction date and so on.

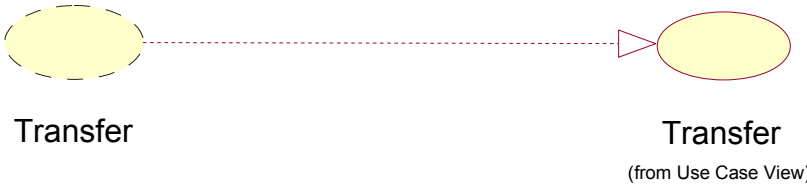
9. CONSTRUCTION OF USE CASE REALIZATIONS

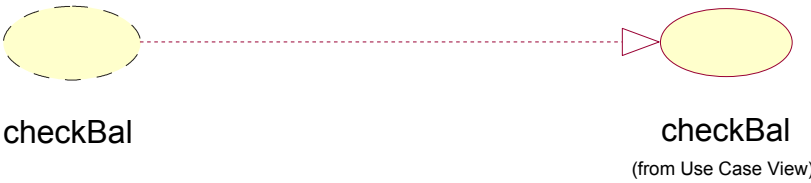
A use case realization is a graphic sequence of events, also referred as a scenario or an instance of a use case. These realizations or scenarios are depicted in either a sequence or collaboration diagrams.

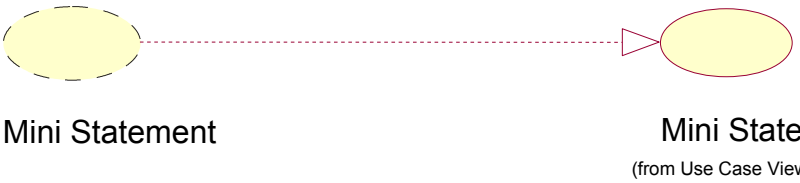
(Figures)

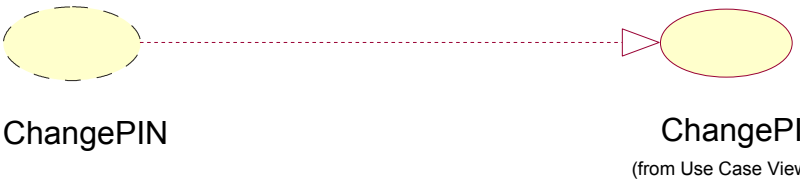


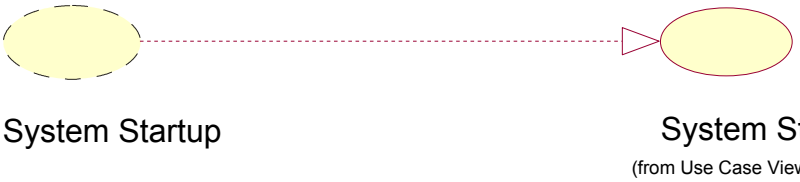

Deposit
Deposit
(from Use Case View)

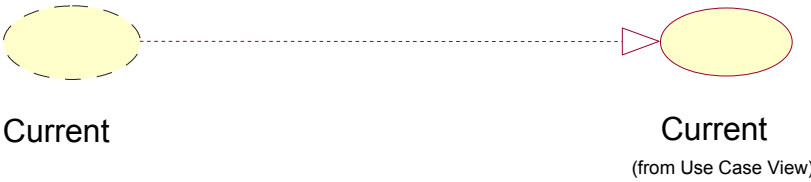

Transfer
Transfer
(from Use Case View)


checkBal
checkBal
(from Use Case View)


Mini Statement
Mini Statement
(from Use Case View)


ChangePIN
ChangePIN
(from Use Case View)


System Startup
System Startup
(from Use Case View)


Current
Current
(from Use Case View)

10. CONSTRUCTION OF SEQUENCE DIAGRAM

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence-what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object's lifeline. The **lifeline** represents the object's existence during the interaction. This form was first popularized by Jacobson. An object is shown as a box at the top of a dashed vertical line. A role is a slot for an object within a collaboration that describes the type of object that may play the role and its relationships to other roles. However, a sequence diagram does not show the relationships among the roles ort eh association among the objects. An object role is shown as a vertical dashed line, the lifeline.

Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name. The label also can include the argument and some control information and show self-delegation, a message the argument and some control information and show self-delegation, a message that an object sends to itself, by sending the message arrow back to the same lifeline. The horizontal ordering of the life lines is arbitrary. Often, call arrows are arranged to proceed in one direction across the page, but this is not always possible and the order conveys no information.

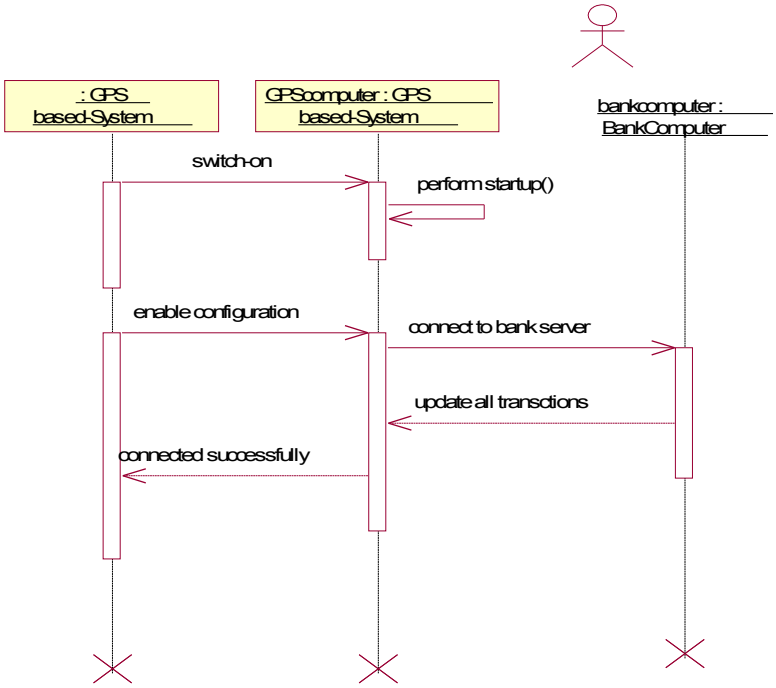
The sequence diagram is very simple and has immediate visual appeal. This is its great strength. A sequence diagram is an alternative way to understand the overall flow of the control of a program. Instead of looking at the code and trying to find out the overall sequence of behavior.

The following tools located on the sequence diagram toolbox which enable to model sequence diagrams:

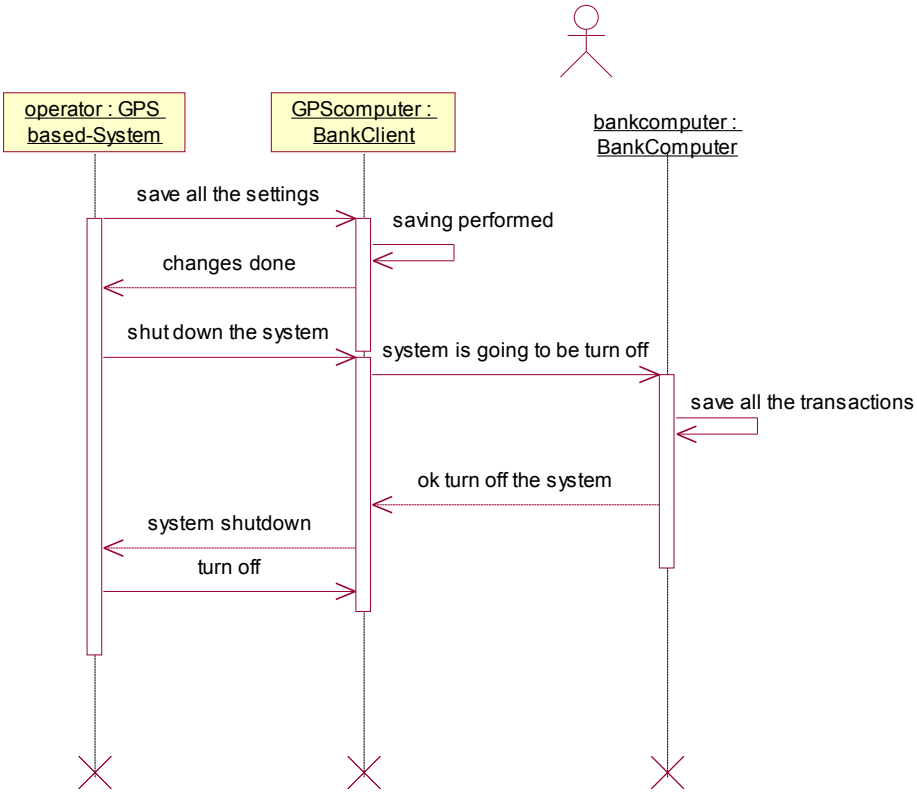
- **Object:** An object has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. Each object in a diagram indicates some instance of a class. An object that is not named is referred to as a class instance.
- **Message Icons**: A message icon represents the communication between objects indicating that an action will follow. The message icon is a horizontal, solid arrow connecting two lifelines together.
- **Focus of Control**: Focus of Control (FOC) is an advanced notational technique that enhances sequence diagrams. It shows the period of time during which an object is performing an action, either directly or through an underlying procedure.
- **Message to Self**: A Message to Self is a tool that sends a message from one object back to the same object. It does not involve other objects because the message returns to the same object. The sender of a message is the same as the receiver.
- **Note**: A note captures the assumptions and decisions applied during analysis and design. Notes may contain any information, including plain text, fragments of code, or references to other documents.
- **Note Anchor**: A note anchor connects a note to the element that it affects.

Sequence diagrams for Banking Applcation are as follows

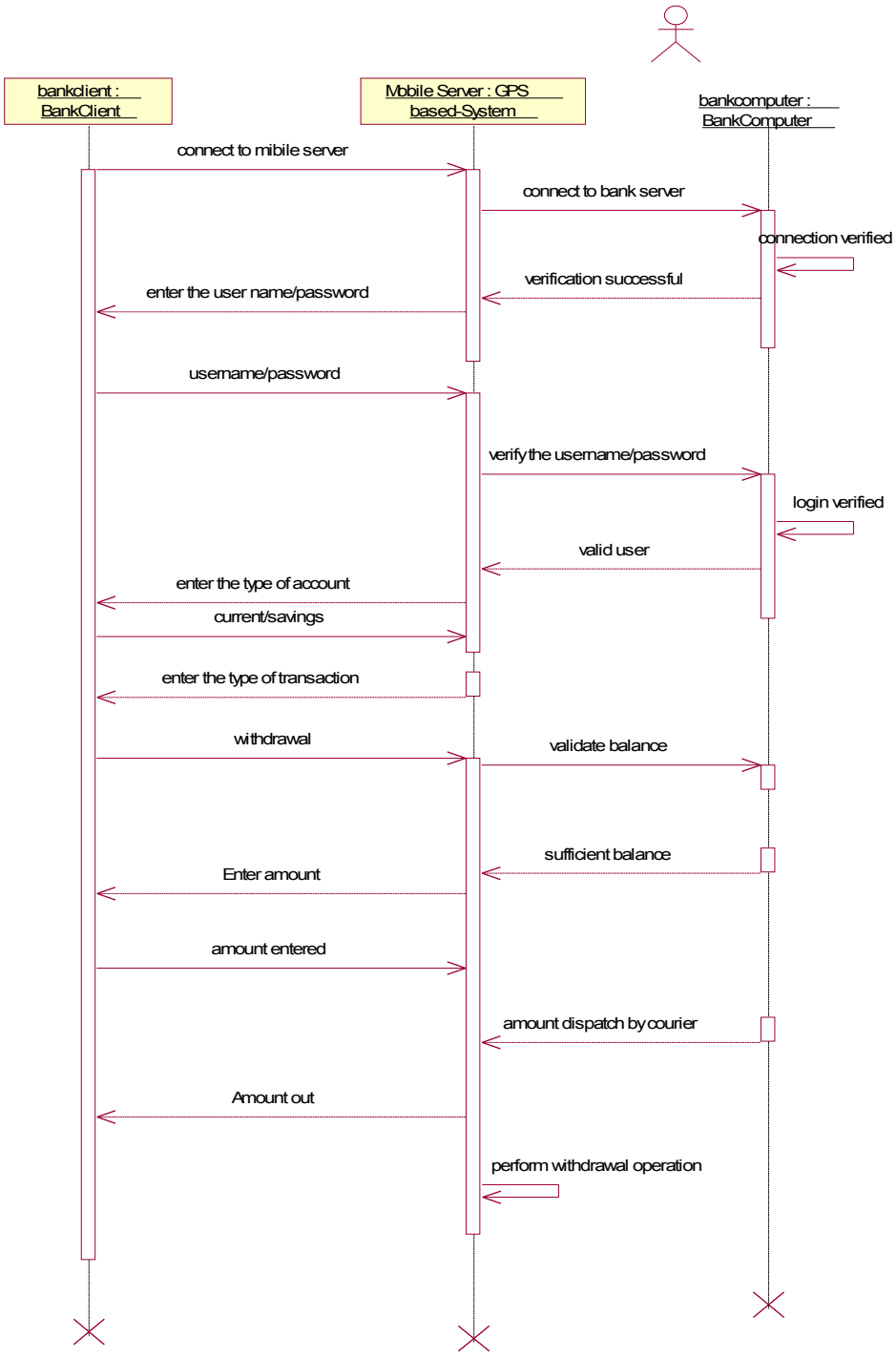
System Start Up:



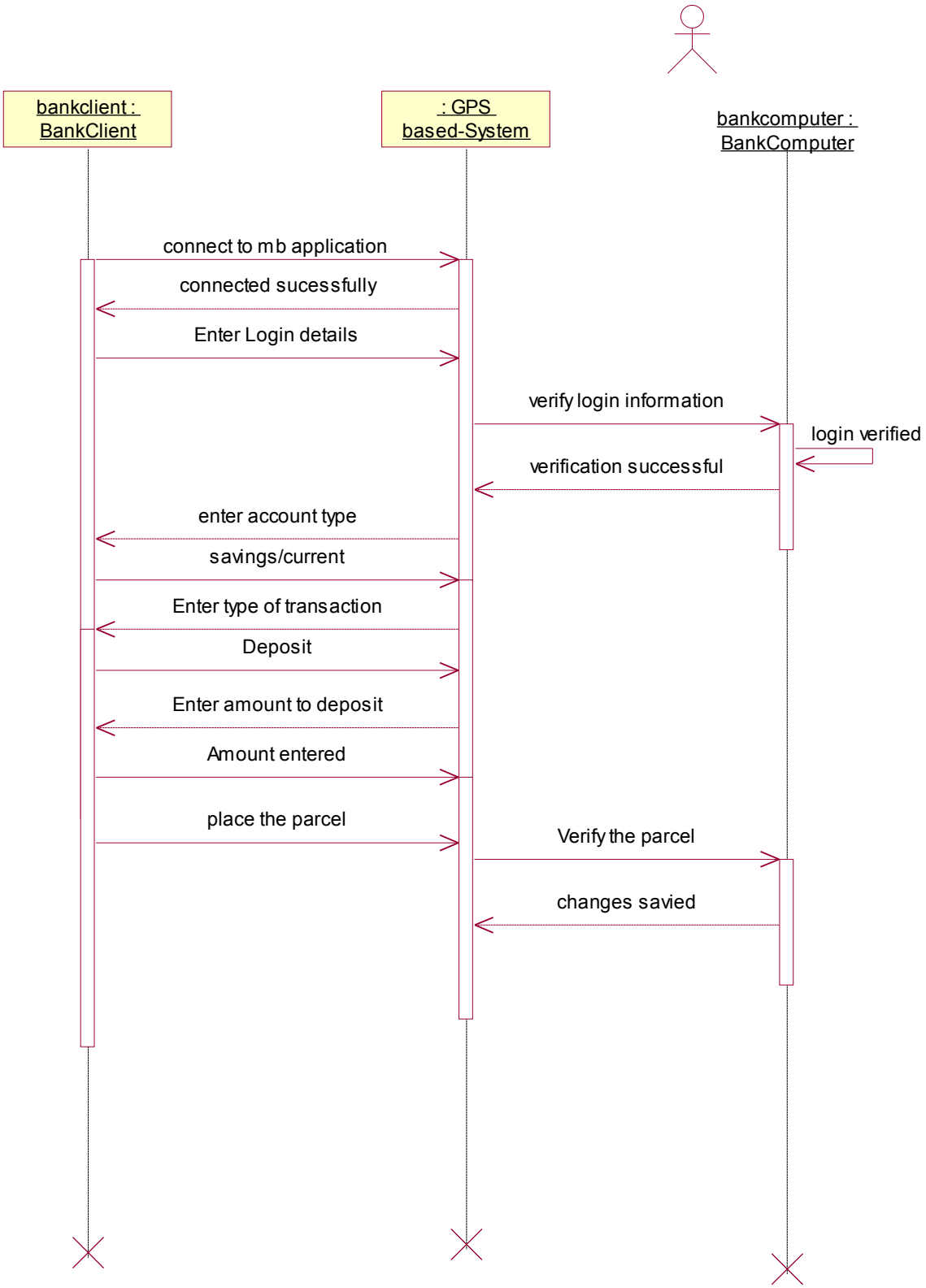
System Shut Down:


operator : GPS based-System
GPScomputer : BankClient
bankcomputer : BankComputer
save all the settings
saving performed
changes done
shut down the system
system is going to be turn off
save all the transactions
ok turn off the system
system shutdown
turn off

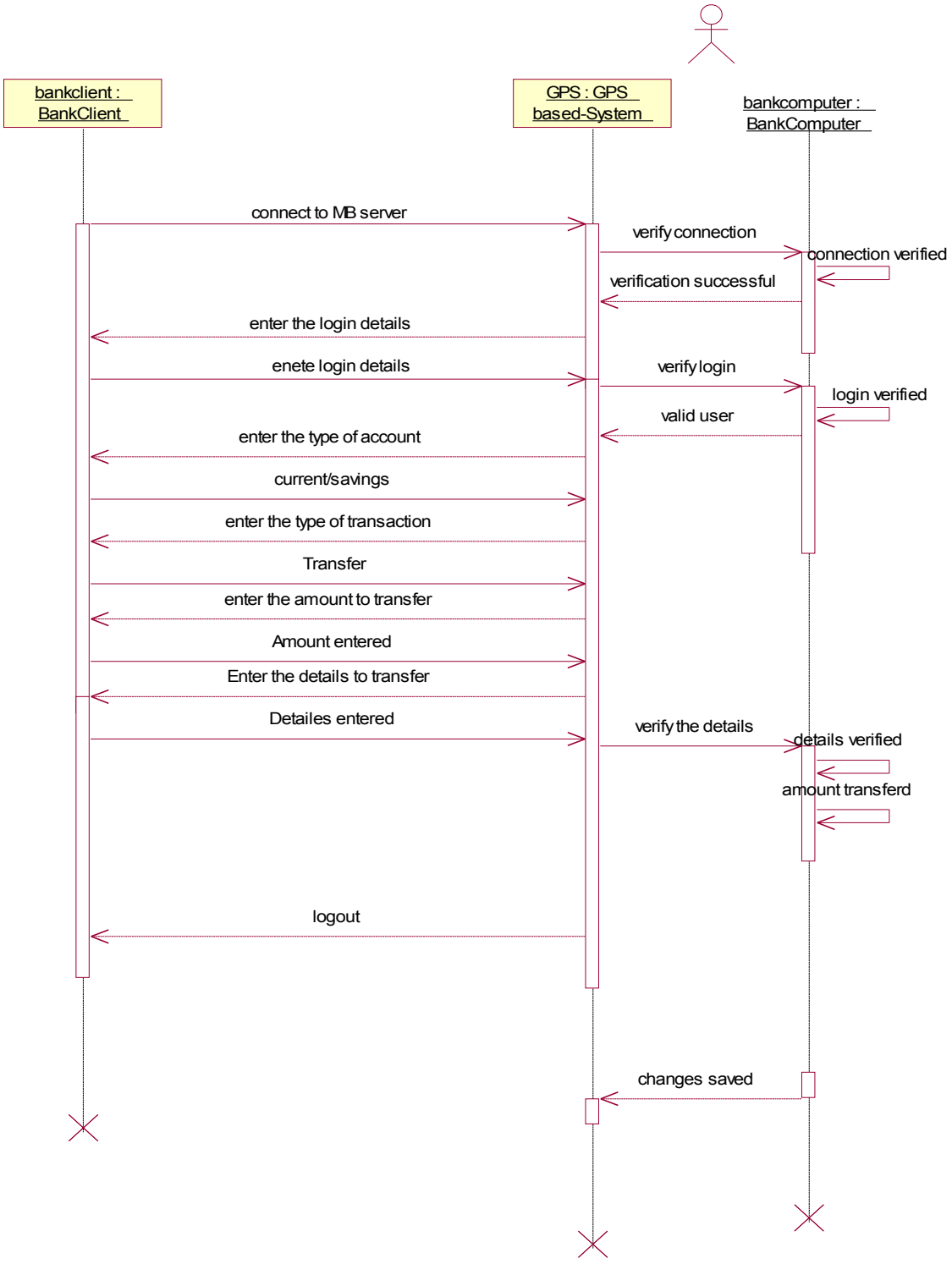
Withdrawal:


bankclient : BankClient
Mobile Server : GPS based-System
bankcomputer : BankComputer
connect to mibile server
connect to bank server
connection verified
verification successful
enter the user name/password
username/password
verify the username/password
login verified
valid user
enter the type of account
current/savings
enter the type of transaction
withdrawal
validate balance
sufficient balance
Enter amount
amount entered
amount dispatch by courier
Amount out
perform withdrawal operation

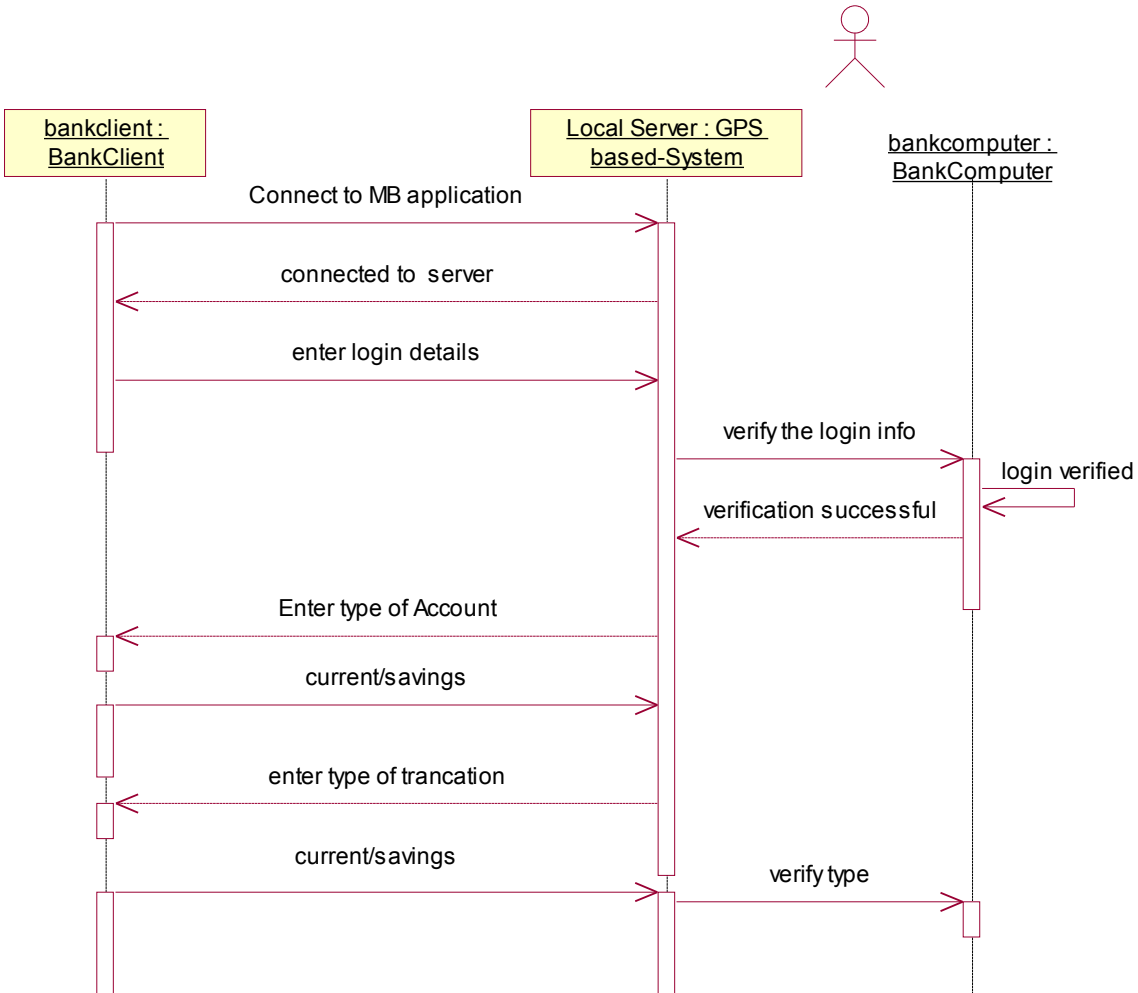
Deposit:

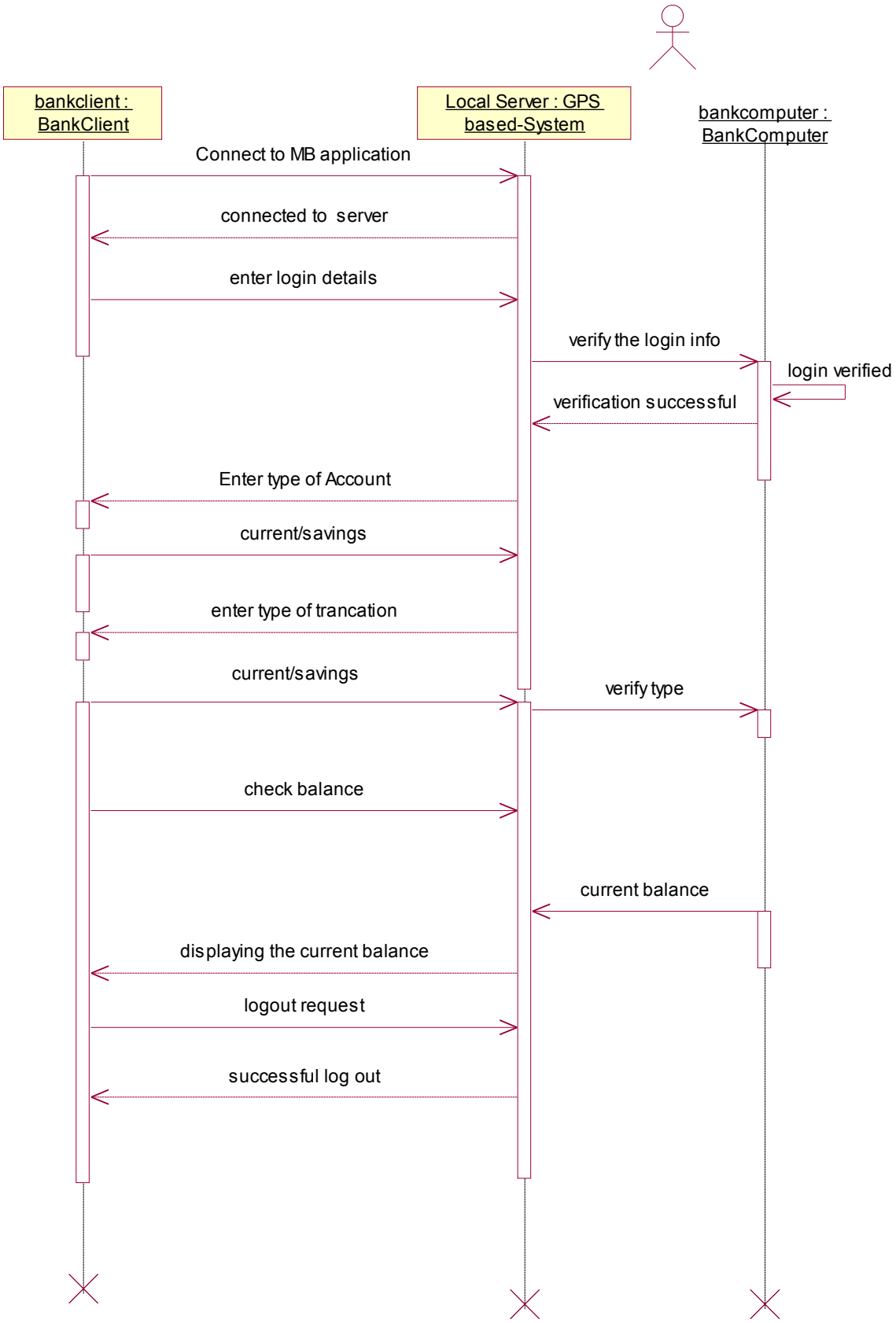

bankclient : BankClient
: GPS based-System
bankcomputer : BankComputer
connect to mb application
connected sucessfully
Enter Login details
verify login information
login verified
verification successful
enter account type
savings/current
Enter type of transaction
Deposit
Enter amount to deposit
Amount entered
place the parcel
Verify the parcel
changes savied

Transfer:

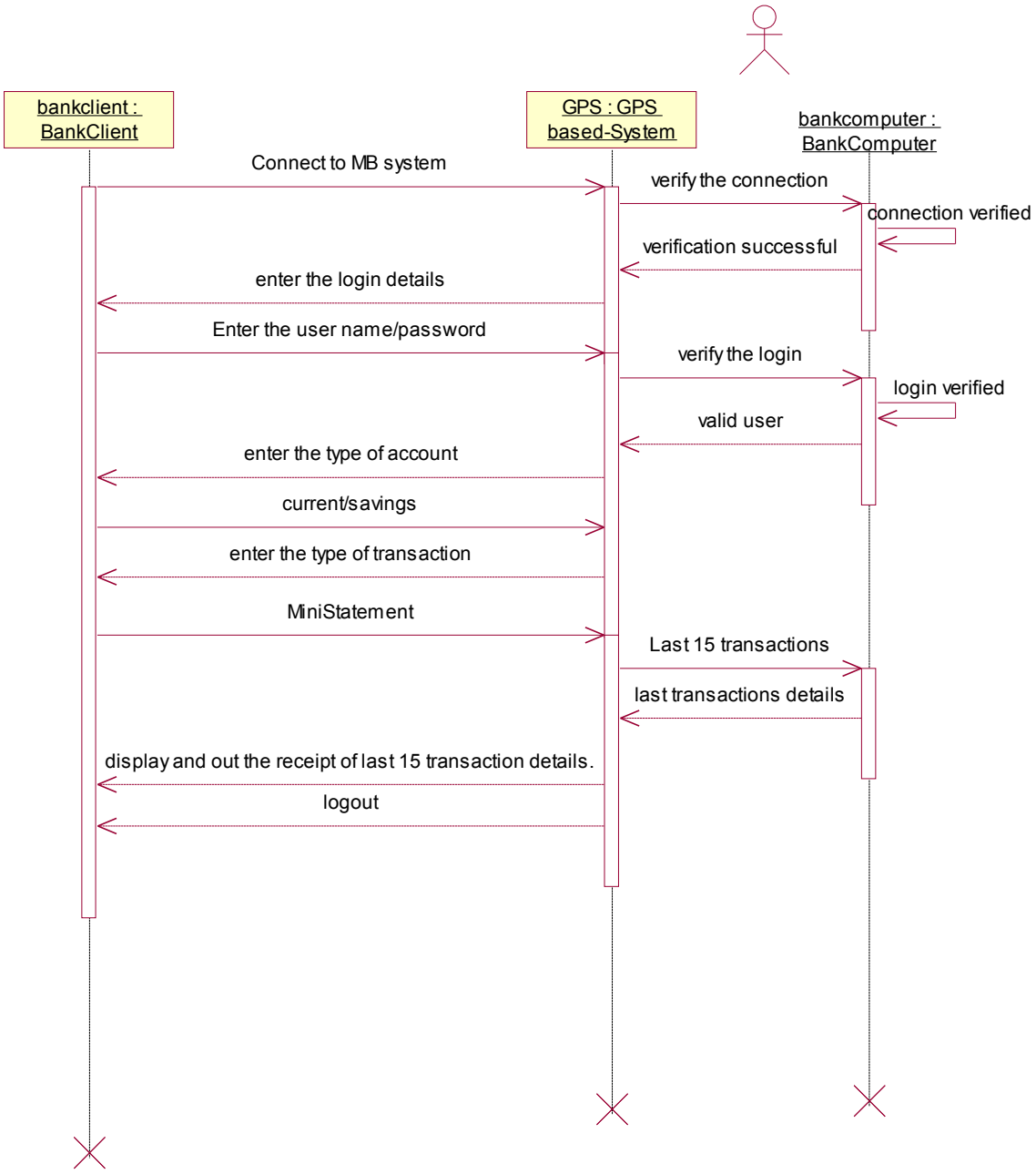

bankclient : BankClient
GPS : GPS based-System
bankcomputer : BankComputer
connect to MB server
verify connection
connection verified
verification successful
enter the login details
enete login details
verify login
login verified
valid user
enter the type of account
current/savings
enter the type of transaction
Transfer
enter the amount to transfer
Amount entered
Enter the details to transfer
Detailes entered
verify the details
details verified
amount transferd
logout
changes saved

CheckBalance:




bankclient : BankClient
Local Server : GPS based-System
bankcomputer : BankComputer
Connect to MB application
connected to server
enter login details
verify the login info
login verified
verification successful
Enter type of Account
current/savings
enter type of trancation
current/savings
verify type
check balance
current balance
displaying the current balance
logout request
successful log out

MiniStatement:



11. CONSTRUCTION OF COLLABORATION DIAGRAM

Collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction. Another type of interaction diagram is the collaboration diagram. A **collaboration diagram** is a set of objects related in a particular context, and interaction, which is a set of messages exchanged among the objects within the collaboration to achieve a desired outcome. In a collaboration diagram, objects are shown in figures. As in a sequence diagram, arrows indicate the message sent within the given use case. In a collaboration diagram, the sequence is indicated by numbering the messages. Some people argue that numbering the messages makes it more difficult to see the sequence than drawing the lines on the page. However, since the collaboration diagram is more compressed, other things can be shown more easily. A collaboration diagram provides several numbering schemes.

Two types of Numbering Sequences are:

1. **Flat Sequence.**

2. Decimal Sequence

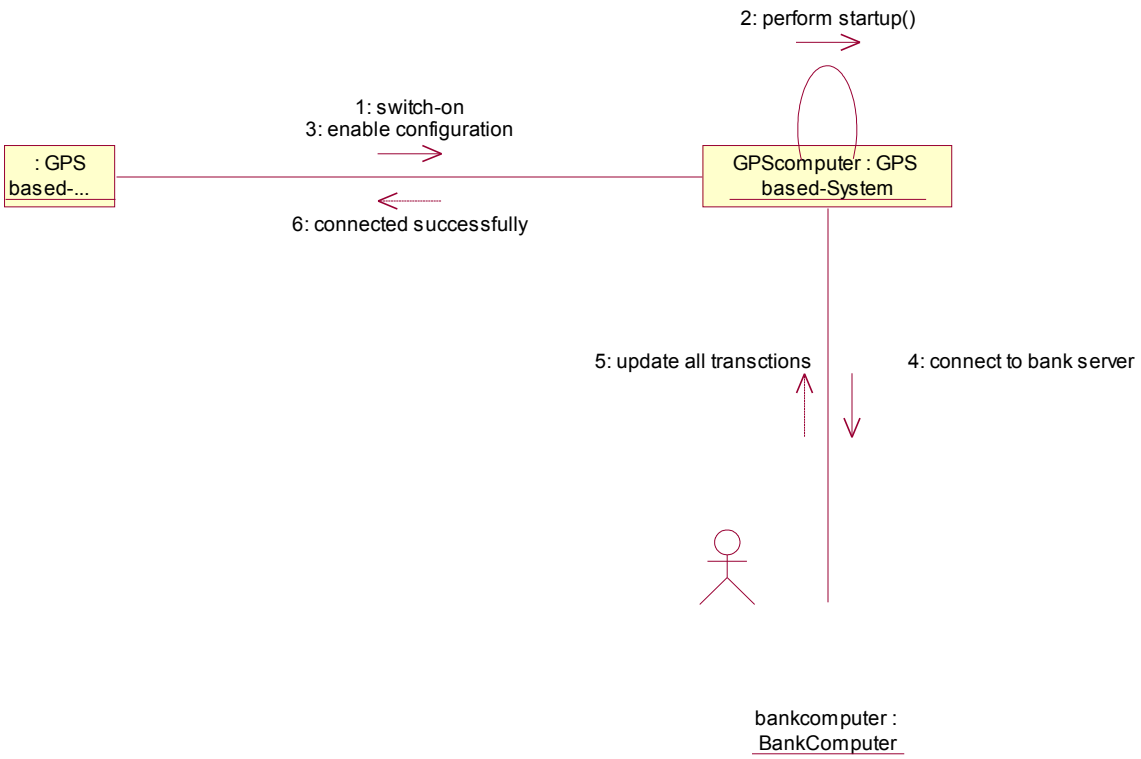
The disadvantage of interaction diagrams is that they are great only for representing a single sequential process; they begin to break down when you want to represent conditional looping behavior. However, conditional behavior can be represented in sequence or collaboration diagrams for each scenario.

Differences between sequence and collaboration diagrams:

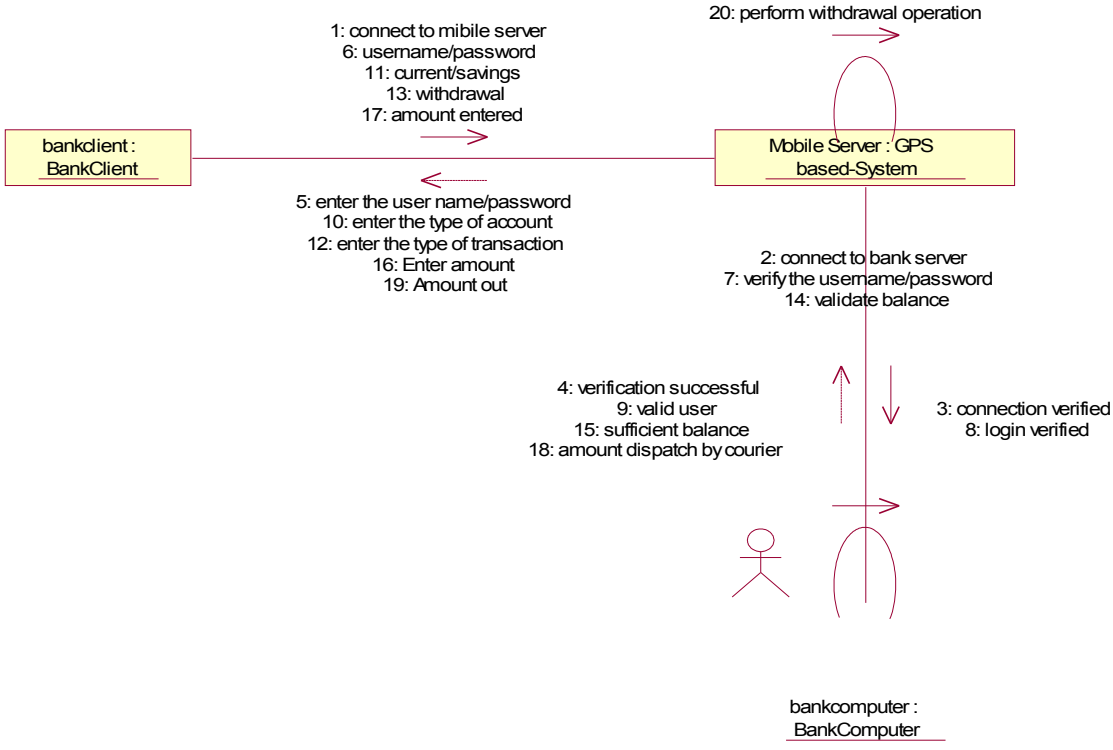
- Sequence diagrams show time-based object interaction while Collaboration diagrams show how objects associate with each other.
- The Create Collaboration Diagram Command creates a collaboration diagram from information contained in the sequence diagram. The Create Sequence Diagram Command creates a sequence diagram from information contained in the interaction's collaboration diagram.
- Sequence diagrams are closely related to collaboration diagrams and both are alternate representations of an interaction. Sequence diagram is easier to read where as collaboration diagram shows how objects are statically connected

Collaboration diagrams for ATM system are as follows

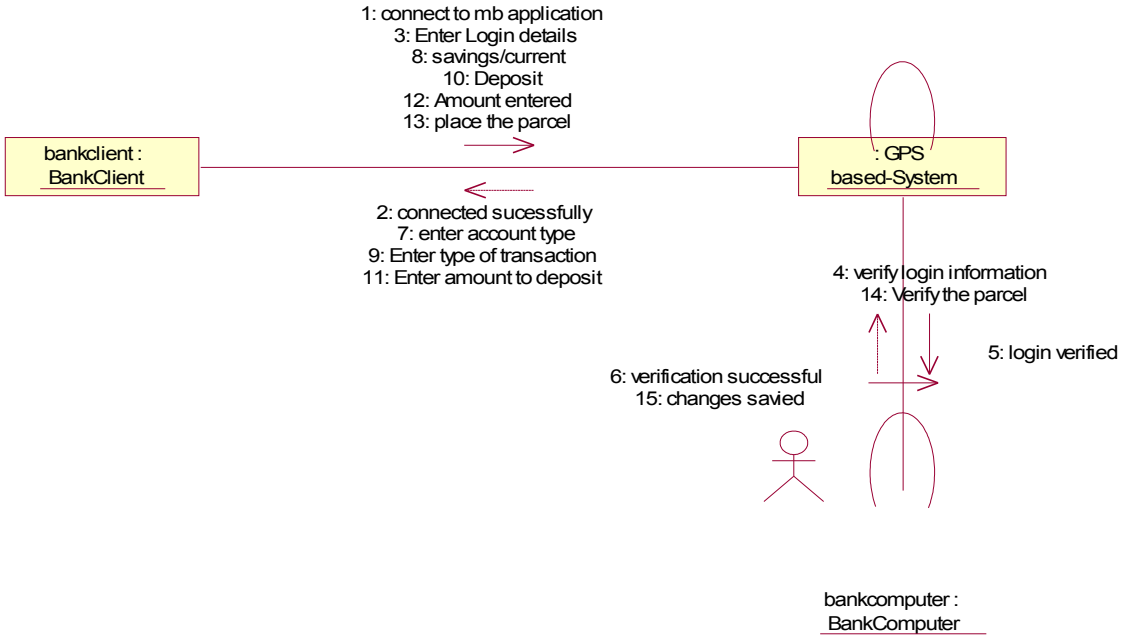
SystemStartup:



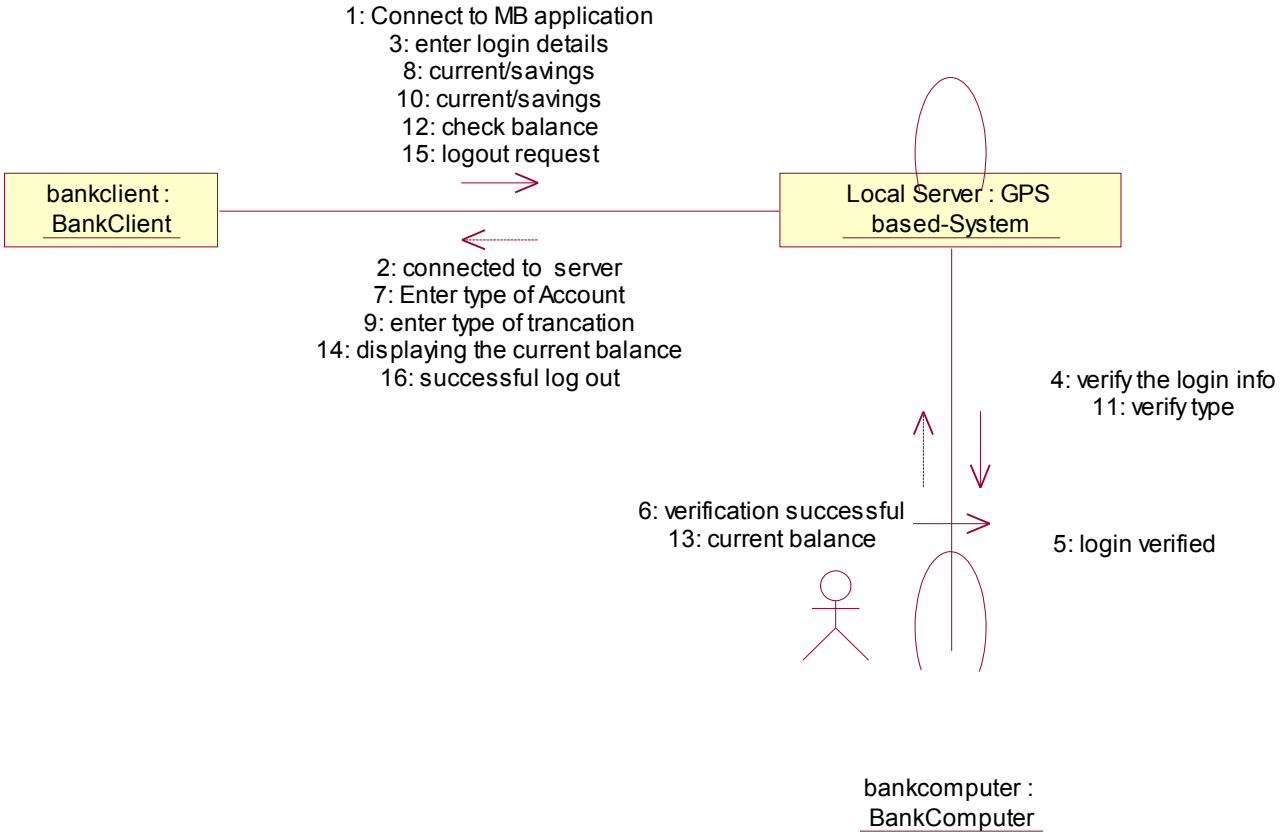
Withdrawal:



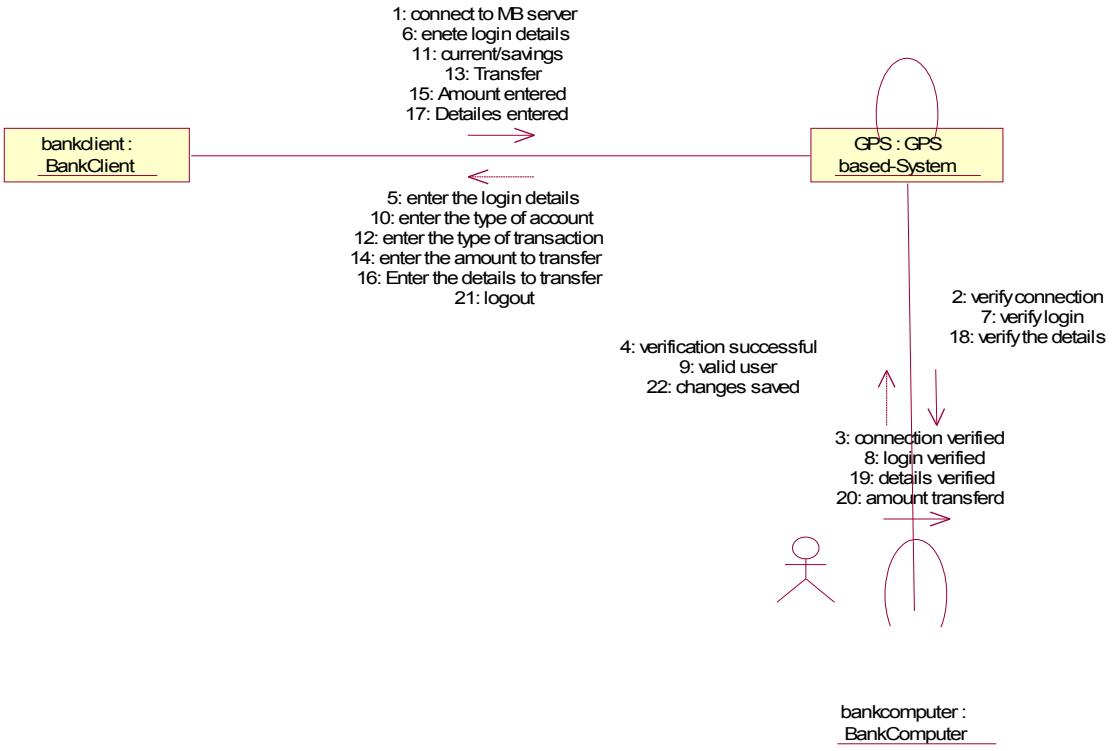
Deposit:


1: connect to mb application
3: Enter Login details
8: savings/current
10: Deposit
12: Amount entered
13: place the parcel
bankclient :
BankClient
: GPS
based-System
2: connected sucessfully
7: enter account type
9: Enter type of transaction
11: Enter amount to deposit
4: verify login information
14: Verify the parcel
5: login verified
6: verification successful
15: changes savied
bankcomputer :
BankComputer

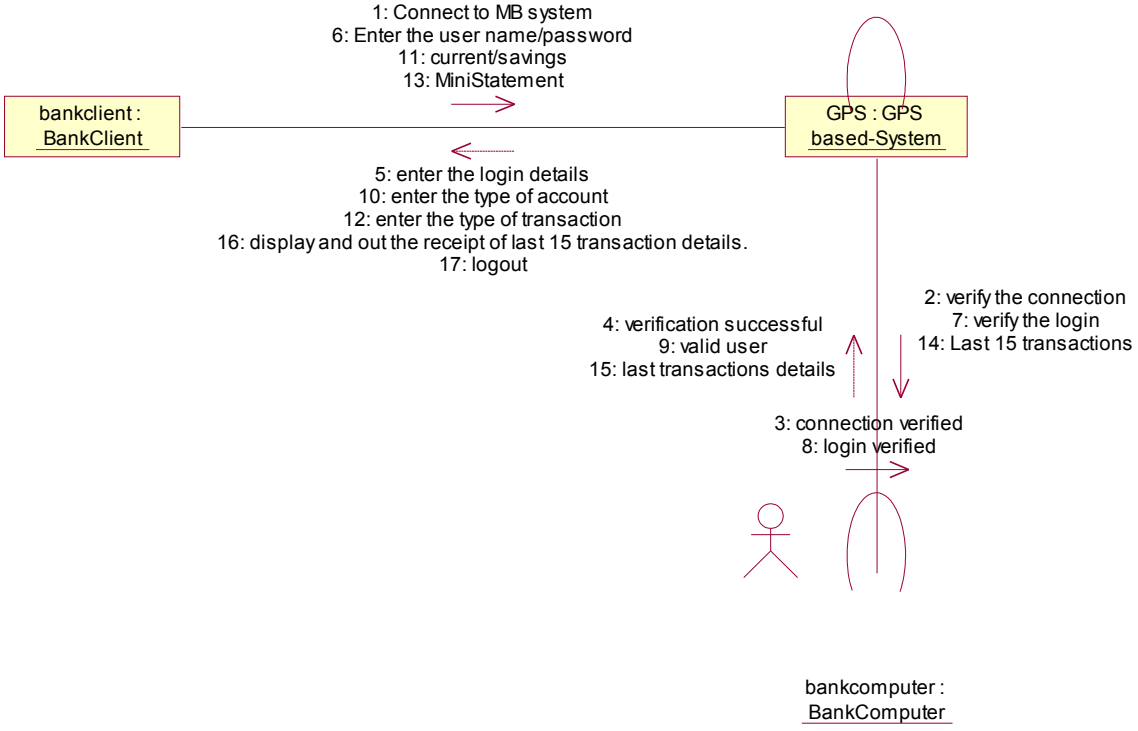
CheckBal:



Transfer:



MiniStstement:



12. IDENTIFICATION OF ATTRIBUTES AND METHODS OF CLASS

Guidelines for identifying attributes of classes are as follows:

- Attributes usually correspond to nouns followed by prepositional phrases. Attributes also may correspond to adjectives or adverbs.
- Keep the class simple; State only enough attributes to define the object state.
- Attributes are less likely to be fully described in the problem statement.
- Omit derived attributes.
- Do not carry discovery attributes to excess.

The following questions help in identifying the responsibilities of classes and deciding what data elements to keep track

- What information about an object should we keep track of?
- What services must a class provide?

Answering the first question help we to identify the attributes of a class .Answering the second question help us to identify class methods.

13. IDENTIFICATION OF RELATIONSHIPS AMONG CLASSES

There are three types of relationships between classes. They are:

- **Association**: This relationship represents a physical or conceptual connection between two or more objects.

- **Super-sub structure** (Generalization hierarchy): These allow objects to be build from other objects. The super-sub class hierarchy is a relationship between classes , where one class is the parent class of another class

- **A-part-of relationship** (Aggregation): This represents the situation where a class consists of several component classes.

14. CONSTRUCTION OF UML STATE CHART DIAGRAM

State chart diagrams model the dynamic behavior of individual classes or any other kind of object. They show the sequences of **states** that an object goes through, the events that cause **a transition** from one state to another and the actions that result from a state change.

State chart diagrams are closely related to **activity diagrams**. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

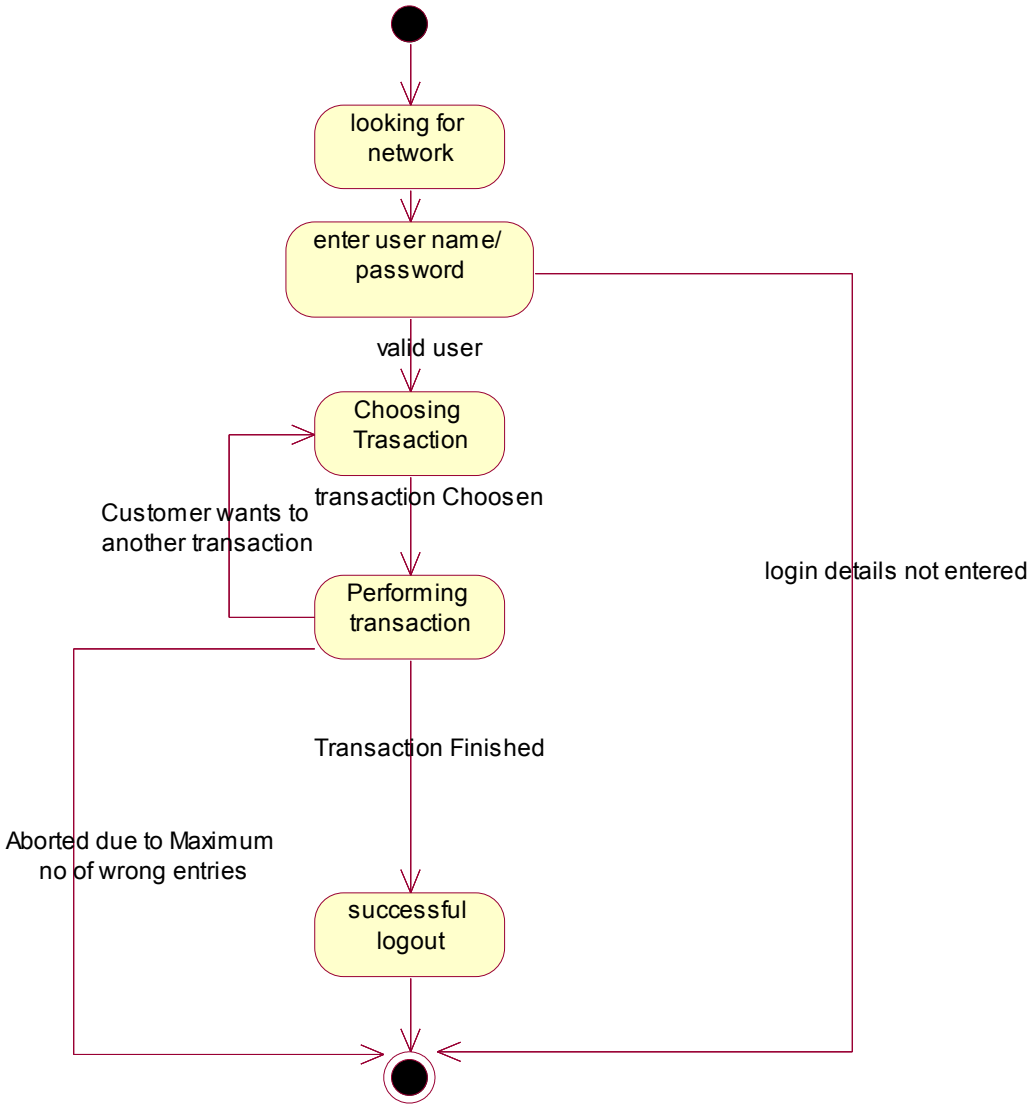
Each state represents a named condition during the life of an object during which it satisfies some condition or waits for some event. A state chart diagram typically contains one **start state** and multiple **end states**. Transitions connect the various states on the diagram.
The following tools are used on the state chart diagram toolbox to model state chart diagrams:

- **Decisions:** A decision represents a specific location on state chart diagram where the workflow may branch based upon guard conditions.
- **Synchronizations**: Synchronizations visually define forks and joins representing parallel workflow.
  - **Forks and Joins**: A fork construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. A join consists of two of more flows of control that unite into a single flow of control

.

- **States*:*** A state represents a condition or situation during the life of an object during which it satisfies some condition or waits for some event.
- **Transitions*:*** A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied.
- **Start states*:*** A start state (also called an "initial state") explicitly shows the beginning of a workflow.
- **End States:** An end state represents a final or terminal state.

State Chart Diagram:


looking for network
enter user name/ password
valid user
Choosing Trasaction
transaction Choosen
Customer wants to another transaction
Performing transaction
login details not entered
Transaction Finished
Aborted due to Maximum no of wrong entries
successful logout

15. CONSTRUCTION OF UML STATIC CLASS DIAGRAM

A class diagram is a picture for describing generic descriptions of possible systems. Class diagrams and collaboration diagrams are alternate representations of object models. Class diagrams contain classes and object diagrams contain objects, but it is possible to mix classes and objects when dealing with various kinds of metadata, so the separation is not rigid.

Class diagrams are more prevalent than object diagrams. Normally you will build class diagrams plus occasional object diagrams illustrating complicated data structures or message-passing structures.

Class diagrams contain icons representing classes, interfaces, and their relationships. We can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. We can also create one or more class diagrams to depict classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they depict; the icons representing logical packages and classes in class diagrams.

We can change properties or relationships by editing the specification or modifying the icon on the diagram. The associated diagrams or specifications are automatically updated.
Classes may be of 3 types. They are:

1. Entity class
2. Boundary class
3. Control class

- **Entity class**: An entity class models information and associated behavior that is generally long live.
- **Boundary Class**: They handle the communication between the systems. They can provide the interface to the user or another system. Ex: Registration form.
- **Control Class**: Control class model sequencing behavior specific to one or more use-cases. You can think of control class as running or executing the use-case i.e., they represent the dynamics of the use-case. Ex: Registration Manager.

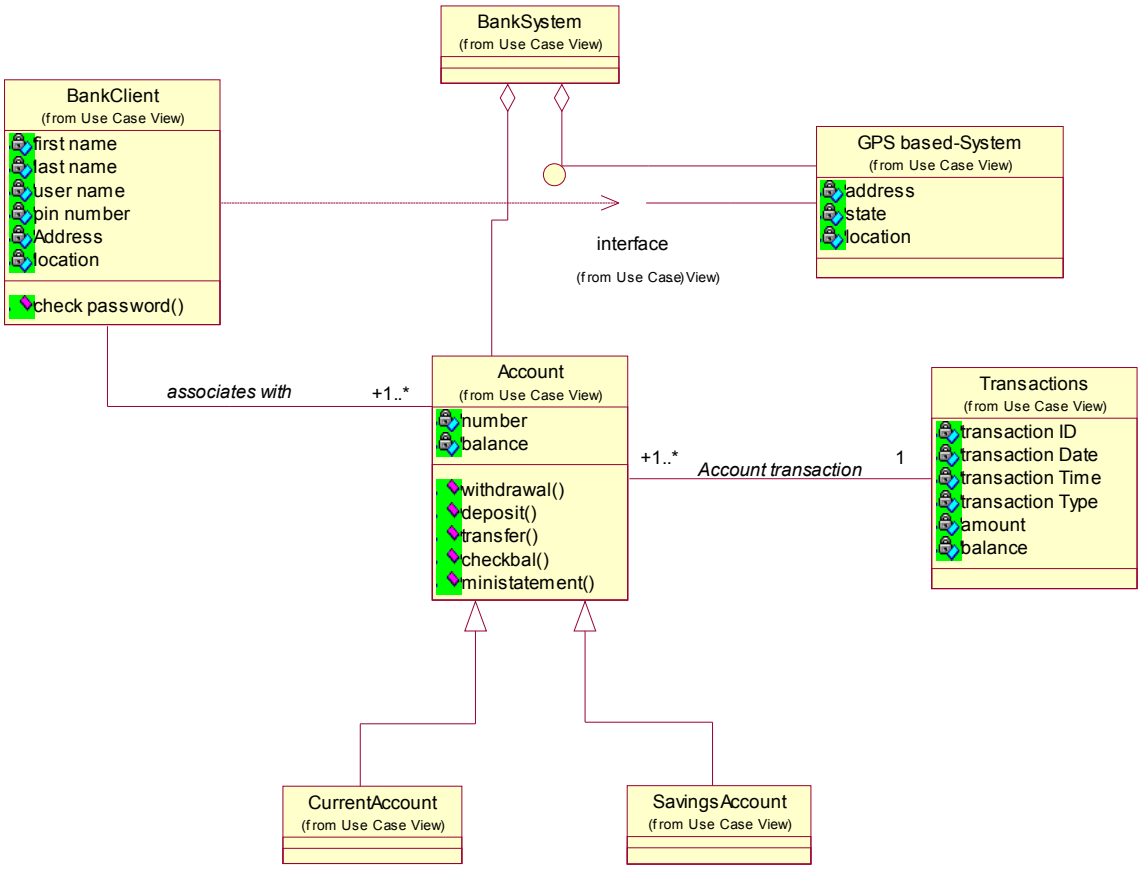


Fig: A UML class diagram of the Banking System Application.

DESIGN

16. DESIGN CLASSES BY APPLYING DESIGN AXIOMS

An **axiom** is a fundamental truth that always is observed to be valid and for which there is no counterexample or example. Suh explains that axioms may be hypothesized from a large number of observations by noting the common phenomena shared by all cases; they cannot be proven or derived, but they can be invalidated by counterexamples or exceptions. A **theorem** is a proposition that may not be self-evident but can be proven from accepted axioms.

Here the design axioms have been applied to object-oriented design. Axiom 1 deals with relationships between system components such as classes, requirements, and software components and Axiom2 deals with the complexity of design.

- **Axiom 1**: The independence axiom: Maintain the independence of components.
- **Axiom 2**: The information axiom: Minimize the information content of the design.

Axiom 1 states that, during the design process, as we go from requirement and use case to a system component, each component satisfy that requirement without affecting other requirements.

Axiom 2 is concerned with simplicity. Scientific theoreticians often rely on a general rule known as Occam's razor, after William of Occam, a 14th century scholastic philosopher. Occam's razor says that, "The best theory explains the known facts with a minimum amount of complexity and maximum simplicity and straightforwardness."

A **corollary** is a proposition that follows from an axiom or another proposition that has been proven.

Corollary 1: Uncoupled design with less information content: Highly cohesive objects can improve coupling because only a minimal amount of essential information need be passed between objects.

Corollary 2: Single purpose: Each class must have a single, clearly defined purpose. When we document, we should be able to easily describe the purpose of a class in a few sentences.

Corollary 3: Large number of simple classes: Keeping the classes simple allows reusability.

Corollary 4: Strong mapping*:* There must be a strong association between the physical system (analysis's object) and logical design (design's object).

Corollary5: Standardization*:* Promote standardization by designing interchangeable components and reusing existing classes or components.

Corollary 6: Design with inheritance*:* Common behavior must be moved to superclasses. The superclass-subclass structure must make logical sense.

17. REFINING ATTRIBUTES, METHODS & RELATIONSHIPS

DESIGNING CLASSES: THE PROCESS

Apply design axioms to design classes, their attributes, methods associations, structures, and protocols.

1. Refine and complete the static UML class diagram by adding details to that diagram.
   1.1. Refine attributes.
   1.2. Design methods and the protocols by utilizing a UML activity diagram to represent the method's algorithm.
   1.3. Refine the associations between classes (if required).
   1.4. Refine the class hierarchy and design with inheritance (if required).
2. Iterate and refine.

REFINING ATTRIBUTES:

In the analysis phase, the name of the attribute was sufficient. However, in the design phase, detailed information must be added to the model. There are three basic types of attributes. They are:

1) Single-value attributes.
2) Multiplicity or mulivalue attributes.
3) Reference to another object, or instance connection.

UML ATTRIBUTE PRESENTATION:

The following is the attribute presentation suggested by UML:

Visibility name: type-expression=initial-value

Visibility is one of the following:

+ public visibility (accessibility to all classes).
# protected visibility (accessibility to subclasses and operations of the class).
- private visibility (accessibility only to operations of the class).

Type-expression is a language-dependent specification of the implementation type of an attribute. Initial-value is a language-dependent expression for the initial value of a newly created object. The initial value is optimal. The UML style guidelines recommend beginning attribute names with a lowercase letter. In the absence of a multiplicity indicator (array), an attribute holds exactly one value. Multiplicity may be indicated by placing a multiplicity indicator in brackets after attribute name. The multiplicity of 0..1 provides the possibility of null values: the absence of a value, as opposed to a particular value from the range.

DESIGNING METHODS AND PROTOCOLS:

The main goal of this activity is to specify the algorithm for methods identified so far. Once you have designed your methods in some formal structure such as UML Activity diagrams with an OCL description, they can be converted to programming language manually or in automated fashion i.e. using CASE tools. A class can provide several types of methods:

- **Constructor:** Method that creates instances of the class.
- **Destructor:** The method that destroys instances.
- **Conversion method:** The method that converts a value from one unit of measure to another.
- **Copy method:** The method that copies the contents of one instance to another instance.
- **Attribute set:** The method that sets the values of one or more attributes.
- **Attribute get:** The method that returns the values of one or more attributes.
- **I/O methods:** The methods that provide or from a device.
- **Domain specific:** The method specific to the application.

Here are five rules:

1. If it looks messy, then it's probably a bad design.
2. If it is too complex, then it's probably a bad design.

3. If it is too big, then it's probably a bad design.

4. If people don't like it, then it's probably a bad design.

5. If it doesn't work, then it's probably a bad design.

UML OPERATION PRESENTATION:

The following operation presentation has suggested by the UML. The operation syntax is this:

Visibility name (parameter-list) return-type-expression

Where visibility is one of:

+ public visibility (accessibility to all classes).

protected visibility (accessibility to subclasses and operations of the class).

- private visibility (accessibility only to operations of the class).

Here, name is the name of the operation. Parameter-list is a list of parameters, separated by commas, each specified by:

name: type-expression=default value

Return-type-expression is a language-dependent specification of the implementation of the value returned by the method. If return-type is omitted, the operation does not return a value.

REFINING METHODS FOR THE STUDENT INFORMATION SYSTEM OBJECTS:

Refined methods for Account class:

The refined classes are:

+Withdrawal()

+Deposit()

+Transfer()

+CheckBal()

+MiniStatement()

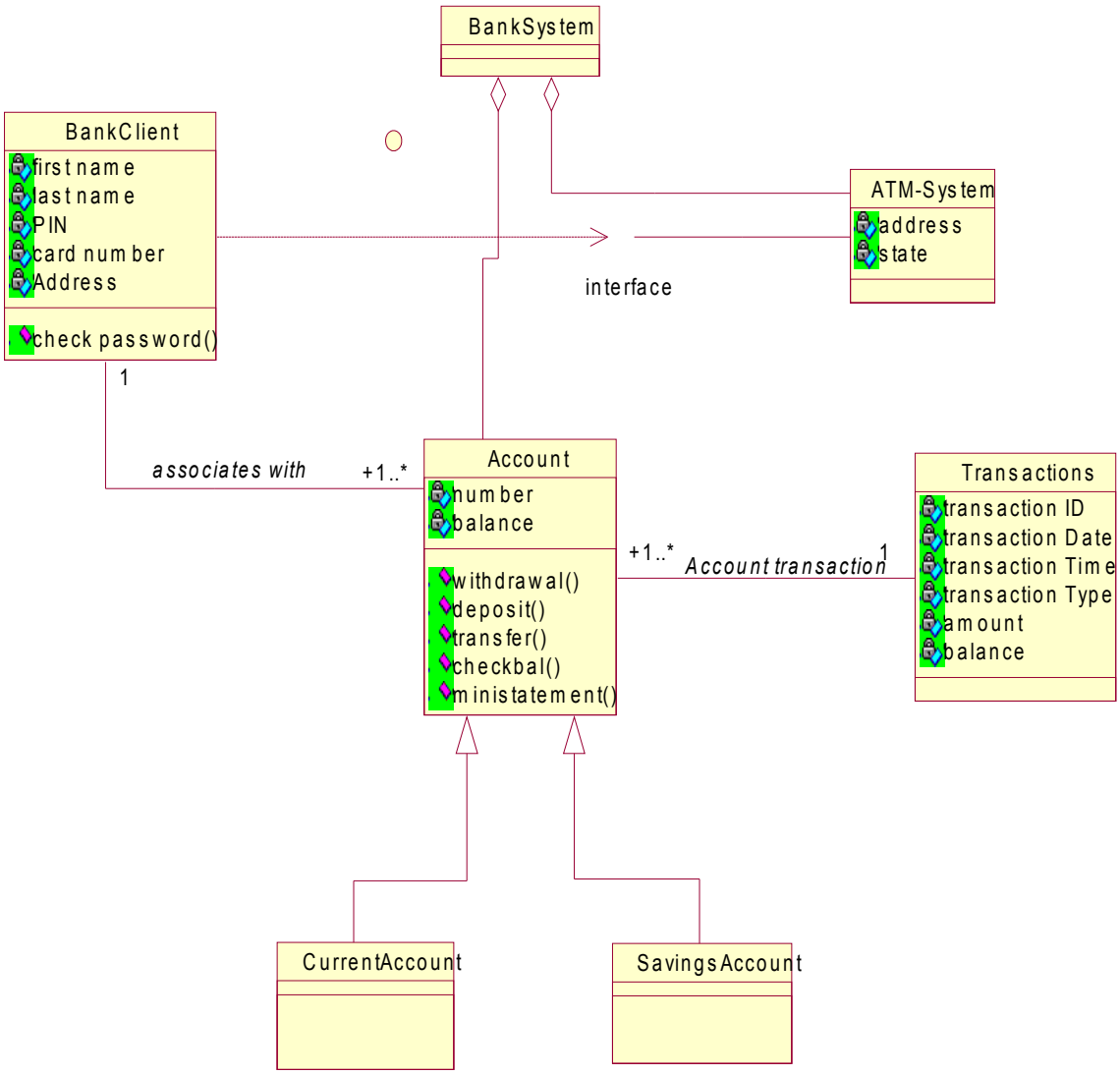
Refined methods for Client class

The refined classes are:

+ChangePassword()

18. AN OVERALL REFINED CLASS DIAGRAM

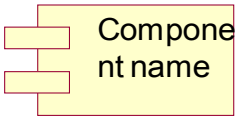
The refined class diagram for the system is:



19.THE COMPONENT VIEW

It contains the implementation view which concerns itself with the actual software module organization within the development environment. The implementation view of architecture takes into account derived requirements related to ease of development, software management, reuse, and constraints imposed by programming languages and development tools. The modeling elements in the component view of architecture are the packages and components along with their connections.

In the component view of the model, a source code component represents a software file that is contained by a package. The type of file is language independent. Each component is assigned a language that is discussed in the language-dependent appendices. Classes in logical view are mapped to components in the component view. The UML notation for a component is as follows:

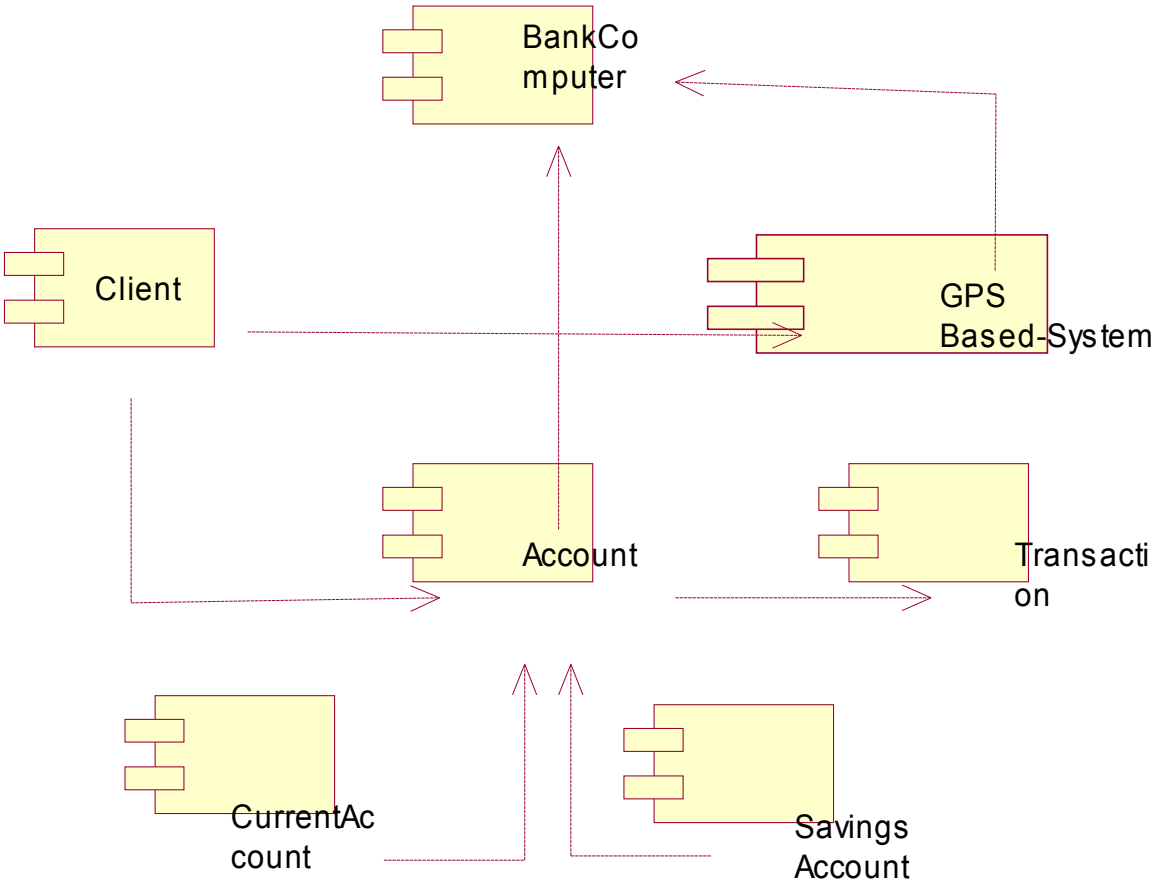


Classes in logical view are mapped or assigned in realizes tab of component.

THE PROCESS VIEW

This view of architecture focuses on the run-time implementation of the system. The process view of architecture takes into account requirements such as performance, reliability, scalability, integrity, system management, and synchronization. Component diagrams are created to view the run-time and executable components created for the system. Components are related via dependency relationships. Executable components show the interfaces and calling dependencies among executables.

For the ATM System UML requires the component diagram as follows



20. THE DEPLOYMENT VIEW

The deployment view of architecture involves mapping software to processing nodes – it shows the configuration of run-time processing elements and the software processes living on them. The deployment view takes into account requirements such as system availability, reliability, performance, and scalability. Deployment diagrams are created to chow the different nodes along with their connections in the system. The deployment diagram visualizes the distribution of components across the enterprise. Run-time processing elements are represented as nodes, which are connected by associations indicating communications path between them. Software processes are illustrated as text attached to a node or group of nodes.

This diagram allows the architecture team to understand the system topology and aids in mapping components to executable processes. Issues such as processor architecture, speed, and capacity, along with inter process communication bandwidth/capacity, physical location of the hardware, and the distributed processing techniques, all come into play.

Here in the discussion forum, the deployment diagram includes a processor which initiates forum's webpage (XML document) and forum controller component in xml or html will be present in each of replicated web pages in any browser depending on xml DTD, as such the main forum controller component is present in device-server, in common.

This Diagram defines the typical Banking system Application network configurations, including those typically used by customers (users), as well as special configurations used for development and test.

· Allocate processes to the various nodes. Allocation takes into account the capacity of the nodes (in terms of both memory and processing), bandwidth of the communication medium, and the availability of the hardware and communication links, rerouting, ...