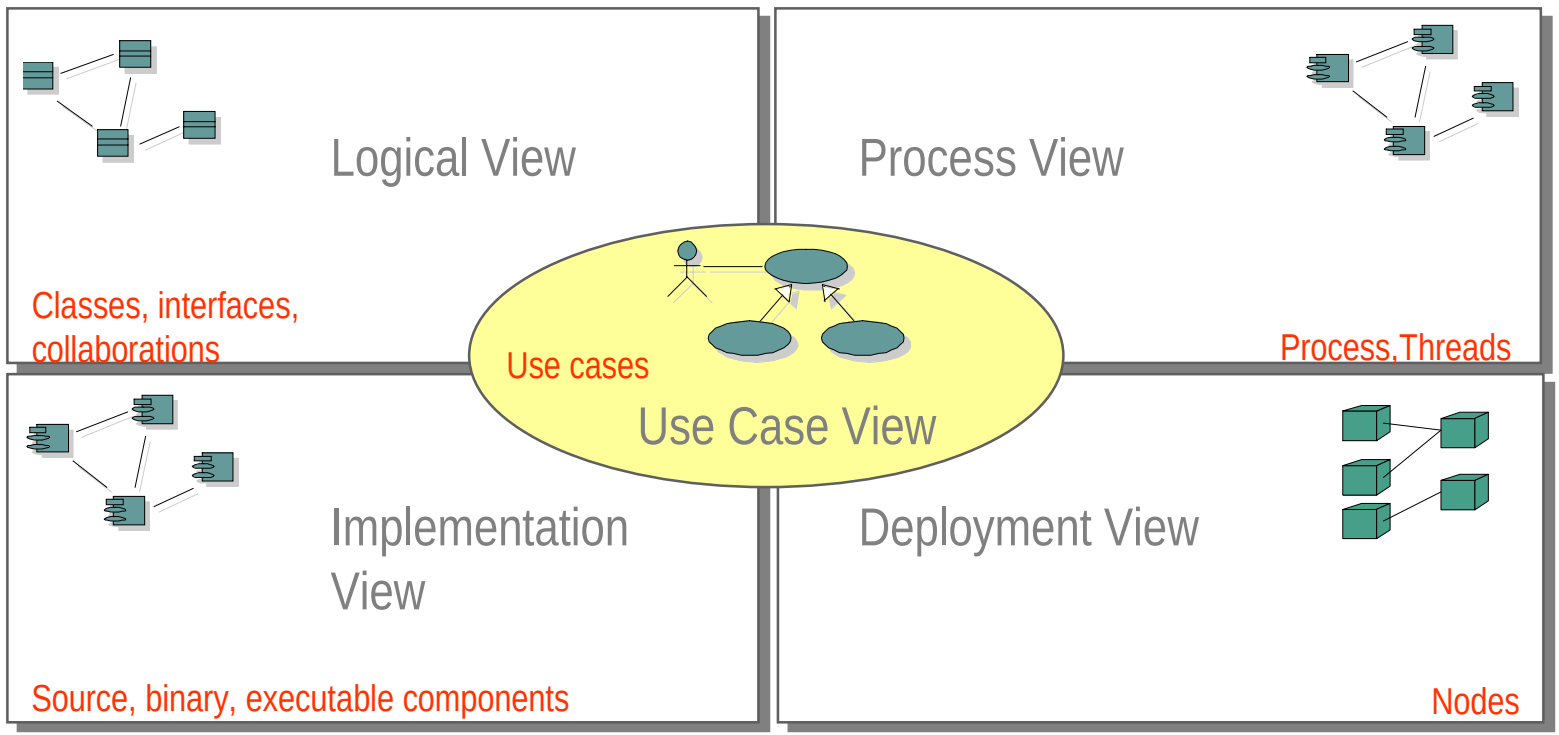
Use Case View

1. Overview
2. Graphical Constructs
3. Textual Description
4. The Architectural View of the Use Case Model



1. Overview

Use Case View

- ☞Captures system functionality as seen by users
- ☞Built in early stages of development
- ☞Developed by ***analysts and domain experts***
- ☞**System behavior**, that is what functionality it must provide, is ***documented*** in a use case model.

Use Case Model

☞illustrates the system's ***intended functions*** (use cases), its ***surroundings*** (actors), and ***relationships*** between the use cases and actors (use case diagrams).

- provides a vehicle used by the customers or end users and the developers to ***discuss the system's functionality and behavior***.

- starts in the Inception phase with the ***identification of actors*** and ***principal use cases*** for the system, and is then matured in the elaboration phase, by adding more details and additional use cases.

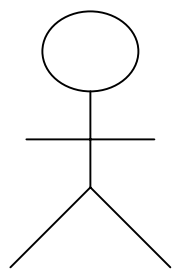
2. Graphical Constructs

Actors

- represent anyone or anything that must interact with the system: they are not part of the system.

- may:
  - only input information to the system
  - only receive information from the system
  - input and receive information to and from the system

- In the UML, an actor is represented as a stickman with a name:



Administrator

Use Cases

- represent the functionality provided by the system; that is:
  - what capabilities will be provided to an actor by the system or
  - what tasks are performed by each actor?

- sequence of transactions performed by a system that yields a measurable result of values for a particular actor.

- In the UML, a use case is represented by an oval with a name inside:

Use Case Relationships

☞*Association*:

- a relationship that represents communication between an actor and a use case;
- can be navigable in both ways or in only one way.

☞Inheritance:

- Generalization or specialization relationships that may exist between actors.

☞Two types of relationships that may exist between use cases: *uses* and *extends*:

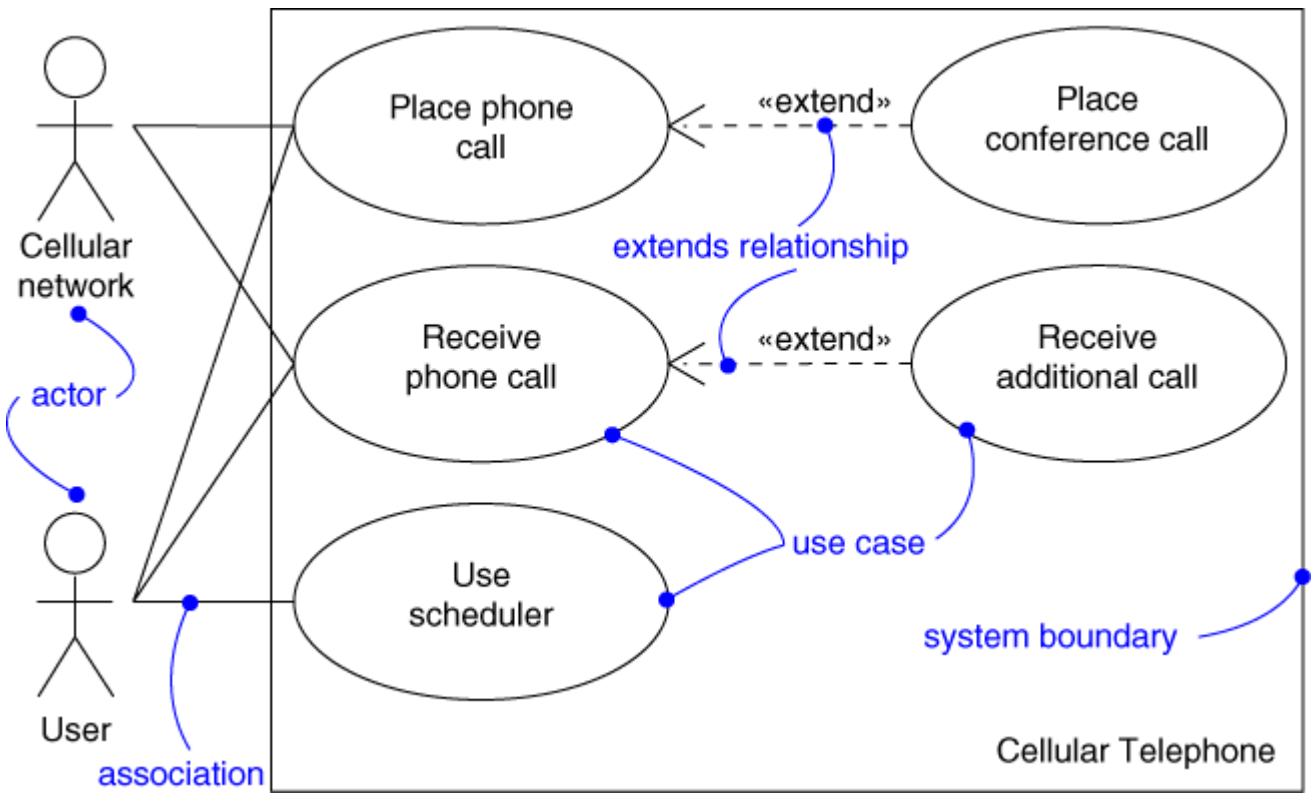
- A functionality shared by multiple use cases can be placed in a separate use case which is related to these uses cases by a *uses* relationship.

- An *extends* relationship is used to show:
  - -Optional behavior
  - -Behavior that is only run under certain conditions, such as triggering an alarm
  - -Different flows which may be run based on actor selection

Use Case Diagrams

- A graphical view of the actors, use cases, and their interactions identified for a system.
  - Consists of the system boundary, and the graphical description of the actors, use cases, and their relationships.

- Example: (Embedded) Cellular Telephone System



3. Textual Description of a Use Case

- Each use case is documented with a flow of events, which is a description of the events needed to accomplish the required behavior.
- The flow of events is written in the language of the domain and describe what the system should do and not how the system does it.
- The flow of events should include:
  - -When and how the use case starts and ends
  - -What interaction the use case has with the actors
  - -What data is needed by the use case
  - -The normal sequence of events for the use case
  - -The description of any alternate or exceptional flows
- **Template:**

*X Flow of Events for the <**name**> Use Case*

X.1 Preconditions
X.2 Main Flow
X.3 Subflows (if applicable)
X.4 Alternative Flows

Where X is a number from 1 to the number of use cases

4. The Architectural View of the Use Case Model

☞Contains only architecturally significant use cases (whereas the final use case model contains all the use cases).

- Is defined during inception and elaboration phases and allows the establishment of a resilient architecture.
- The *logical view* is derived using the use cases identified in the architectural view of the use case model.

☞**Architecturally significant use cases**:

- are the ones that cover the main tasks or functions the system is to accomplish.
- could possibly impact the architecture
- include:
  - -critical use cases, those that are most important to the users of the system
  - -use cases that have the most important nonfunctional requirements, such as performance, responses times etc.

☞Secondary and optional use cases are not key to the architecture.

Example

-Provide the use case view for the architecture of an ATM System.

Overview of the Requirements

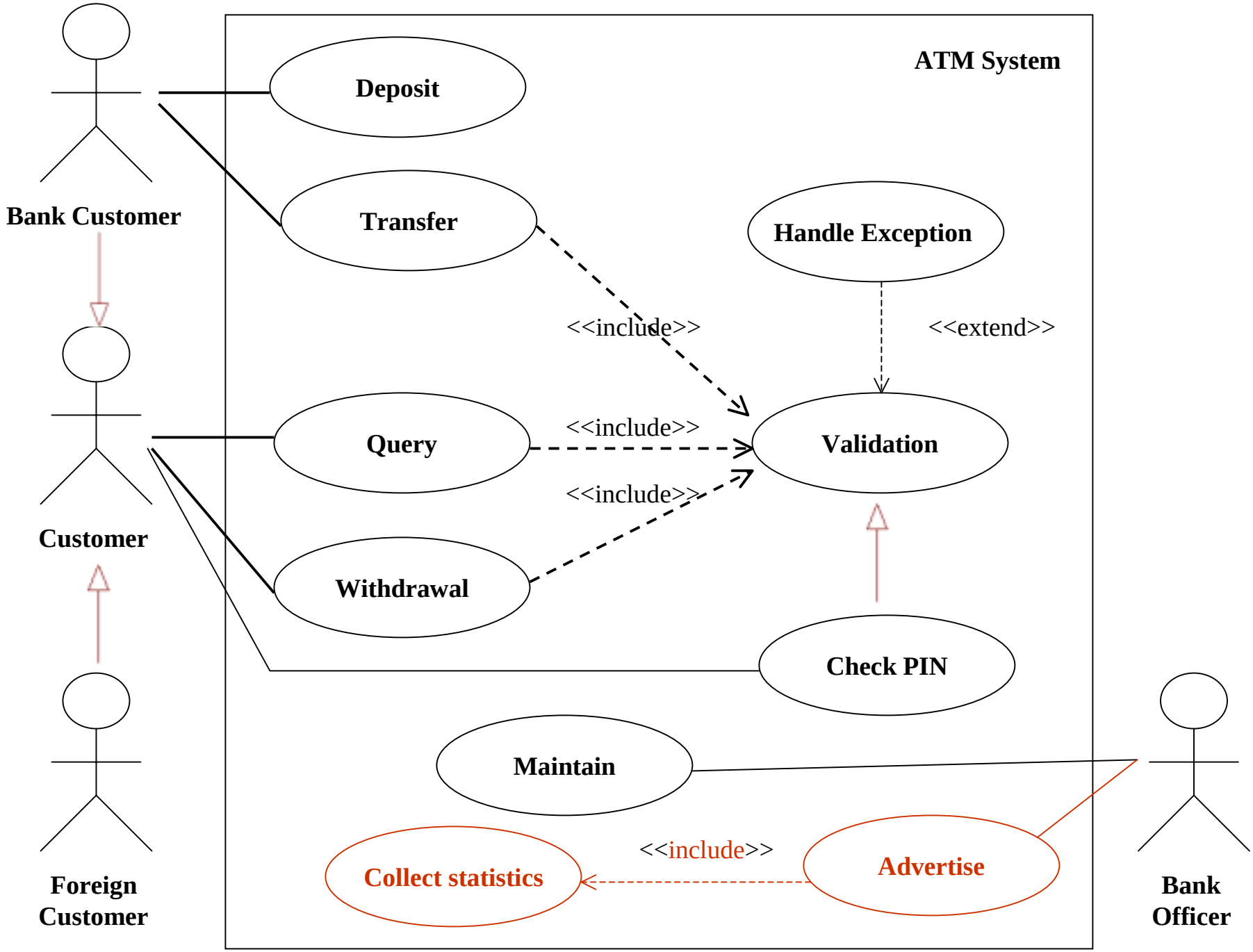
The Interbank Consortium, a hypothetical financial institution, has directed its software development subsidiary, Interbank Software, to develop new services that support a collection of automated teller machines (ATMs).

Customers use ATMs to make queries, withdrawals, deposits and funds transfers involving their accounts. Thieves or crooks must be prevented from interfering with these actions.

Interactions with the ATM would work like this:

1. *The customer inserts his/her bank card into the ATM.*
2. *The ATM prompts the customer for a "password" which the user enters at the ATM.*
3. *The customer then selects an action to be performed; the selected action is then performed by the branch (perhaps causing dispersal of cash at the ATM).*

Additionally, the bank would like to be able to use the system to maintain statistics about customers' behaviour in order to adapt its services to their needs, and also to send them some advertisements when they are using the system.


ATM System
Deposit
Transfer
Handle Exception
<<include>>
<<extend>>
Query
<<include>>
Validation
<<include>>
Withdrawal
Check PIN
Maintain
Collect statistics
<<include>>
Advertise
Bank Customer
Customer
Foreign Customer
Bank Officer

☞Example: Textual Description for Check PIN Use Case

X Flo

Wher

Flow of Events for *Check PIN* Use Case

Main flow of events: The use case starts when the system prompts the *User* for a PIN number. The *User* can now enter a PIN number via the keypad **(E1)**. The *User* commits the entry by pressing the Enter button **(E2)**. The system then checks this PIN number to see if it is valid **(S1, E3)**. If the PIN number is valid, the system acknowledges the entry, thus ending the use case.

Subflows:

S1: The system invokes ***Validate*** use case.

Alternative flow of events:

E1: The *User* can clear a PIN number any time before committing it and reenter a new PIN number.

E2: The *User* can cancel a transaction at any time by pressing the Cancel button, thus restarting the use case. No changes are made to the *User's* account.

E3: If the *User* enters an invalid PIN number, the use case restarts. If this happens three times in a row, the system cancels the entire transaction, preventing the *User* from interacting with the ATM for 30 minutes.