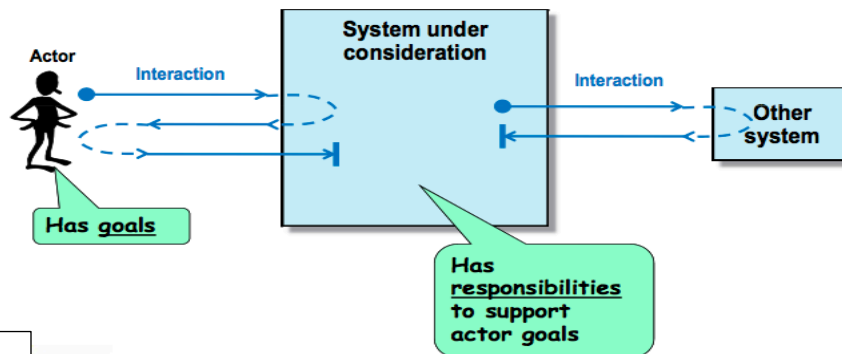


# Use Case Modeling

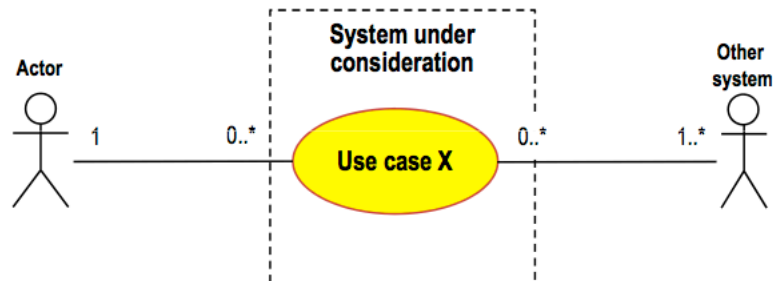
## The Use Case Conceptual Framework

- ♦ An interaction with a system for the purpose of achieving a clearly defined goal
  - Something which is of value to the actor



## UML Representation of Use Cases

- ♦ The UML graphical representation is merely a shorthand rendering of a use case
  - The important (and hard) part is specifying the use case itself



© J.-Pierre Corriveau, 1997- present

© B. Selic

3004 T3a - 3

## Key Concepts

- ♦ **NB:** *There is still no visible consensus on the definitions of key use case concepts*
  - A workshop of 14 leading OO consultants had 14 definitions of "use case"
  - Occurs on the boundary between the informal (i.e., what users want) and formal (i.e., what engineers implement)
- ♦ **Actor:** an entity (person or system) which interacts with a system to achieve one or more goals
- ♦ **Use case:** a collection of possible interactions between actors and a system relating to one or more goals
  - Being a collection of possible scenarios, a use case is similar to a class
  - ..we talk about "instances" (occurrences) of use cases

© J.-Pierre Corriveau, 1997- present

© B. Selic

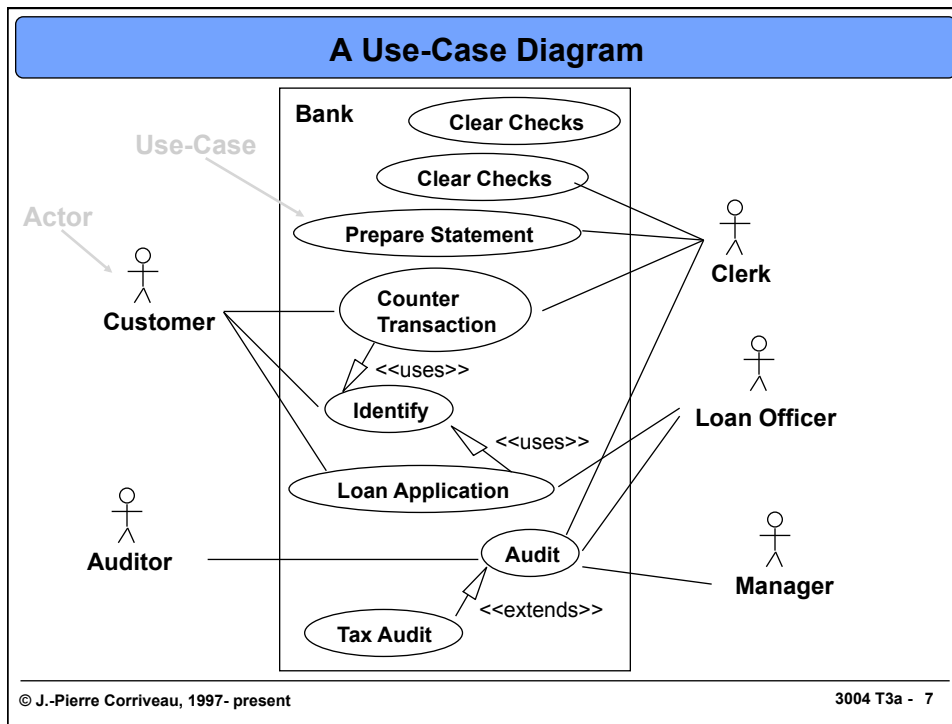
3004 T3a - 4

## Steps of UC Modeling

- **For event-driven systems, UC modeling consists of the following steps:**
  - Scope the system (by considering different Actor perspectives)
  - Identify **events** and **actors**
    - » Actors are **abstractions** generating events
    - » Think of **internal** and **external** events
  - List use-case titles (and prioritize them)
  - Produce a use-case diagram
  - Document use-cases using a scenario textual description (STD) technique
    - » We want the STDs in the design document.

## Identifying External Events

<u>Event</u>	<u>System Resp.</u>	<u>Arrival</u>	<u>Response</u>
offhook -->	dialtone	aperiodic <100/min	<500msec
first digit -->	cancel dialtone	aperiodic <20 sec after dialtone	Digit tone <100msec
last digit -->	translation result	interdigit time = 4sec	a.s.a.p
<--ringing			a.s.a.p after last digit
answer-->	cancel ringing and ringtone	aperiodic	<100msec



### An Example Use Case

**Use Case: Making a successful POTS connection**

- **Actors:** calling party, called party
- **Scenario:**
  - Caller lifts telephone receiver
  - Caller hears dial tone
  - Caller dials digits
  - Caller receives audible ring tone
  - Called party's phone rings
  - Called party lifts receiver
  - Caller and Called party are now connected and can talk
  - Called party hangs up
  - Caller receives dial tone
  - Caller hangs up

**Key points: System as Black Box, event/response**

© J.-Pierre Corriveau, 1997- present 3004 T3a - 8

## How to Start?

- You must start by writing down a list of **verifiable** requirements and going from them to UCs.
- Each use case captures a cluster of scenarios:
  - the scenarios of a UC must be logically clustered together
  - a scenario is formed by a (more or less abstract) sequence of input/output events processed by the system (as a black box)
  - through the use of words such as 'OR', 'AND', 'eventually', 'optionally', 'repeatedly', each step of a UC, each scenario, and ultimately each UC can be viewed as a grammar of events
  - In OO, we use a set of UCs to describe system behavior:
    - » unless otherwise documented, UCs are taken to be independent of and concurrent with each other
    - » inter-UC relationships (annotated with stereotypes) are important to identify: the more the UCs are tied to each other, the less partial the overall specification is!
    - » there is generally no overall grammar to build for the whole system but we do aim for req. coverage (via traceability)

## Organizing Use Cases

- We propose that each use case be documented using an STD that **ideally** contains the following information:
  - » a unique identifier
  - » a brief textual description of the overall objective of the UC
  - » the set of external actors that participate in the UC
  - » a set of possible triggering events
  - » a pre-condition that must be satisfied in order to enable the execution of the UC
  - » a sequence of system *responsibilities (or steps)* for the main scenario (JP: if not for ALL scenarios!!!)
  - » a set of possible resulting events for the UC
  - » a post-condition that must evaluate to true after the execution of the UC
  - » a set of alternative scenarios (optional but important!)
  - » a set of nonfunctional requirements that apply to the UC (optional)
  - » a comment section that may be used by designers as a free format text window to specify different issues related to the UC (e.g., which scenarios were grouped into this UC)

## Example STD (1)

### Use Case Identifier: UC-9 ATM withdraw transaction

**Description:** Describes the steps of a normal withdraw transaction

**External Actors:** User, Central Bank System (CBS)

**Related Use Cases:** UC-15 Transaction

**Precondition:** ATM is idle

**Triggering event:** A user inserts a valid bank card

1. User enters a valid bank card.
2. ATM swallows the bank card and reads the card information.
3. ATM initiates the transaction.
4. ATM asks the user to enter PIN. User enters PIN.
5. CBS validates PIN.
6. ATM asks user to choose a transaction option. User chooses the *withdraw* option.
7. ATM asks for amount to withdraw. User enters amount.
8. ATM sends a withdraw transaction request to CBS.
9. CBS verifies that the user account balance is sufficient to cover the requested amount.
10. CBS registers the withdraw transaction.
11. ATM dispenses cash. User picks up cash.
12. ATM prints a transaction receipt. User picks up the receipt.
13. ATM returns the bank card. User picks up the card.

## Example STD (2)

### Use Case Identifier: ATM withdraw transaction

**Resulting event:** ATM returns the bank card

**Postcondition:** ATM is idle again

**Alternatives:**

- If the user enters three successive invalid PINs, then the transaction is refused and the card is kept.
- If the user's account balance is insufficient, then the transaction is refused.
- If the ATM does not have enough cash, then the transaction is refused.

**Nonfunctional requirements:**

- A transaction must be completed in less than two minutes
- ATM can only handle one transaction at the time.

**Comments:**

- A transaction can be cancelled at any time before the transaction is sent to the CBS.

## UML 's Stereotypes and Packages

- **A *package* can be used to regroup a set of use-cases**
  - a package can also be used to regroup other UML entities, such as classes
  - it constitutes a grouping mechanism for scalability in UML
- **A *stereotype* is a user-defined label that allows extensions to the semantics of UML**
  - this is a key mechanism to introduce your own semantics into the modeling process

## Stereotypes for Use-Cases

- **A *Basic Use-Case*:**
  - must describe a typical usage of the system from end-to-end
  - must keep an external, event-driven perspective
- **An *Extension***
  - captures functionality that is optional or additional to one or more basic use cases
    - » We prefer to list alternatives inside an STD.
  - is related to basic a use-case using an *extends* arrow
- **A *Reference***
  - gives a name to a group of steps repeated in **several** use-cases
  - is related to basic a use-case using a *uses* arrow

## Some Conclusions

## Why OO people like Use-Cases

### Use-cases:

- **constitute a simple, intuitive form of scenario modeling**
  - temporal logic for event specification is much more complicated
- **are not object-oriented**
  - only *solutions* to the requirements are OO!
- **make clear what external functionality is expected**
  - the system is treated as a black-box
  - the interface and the DB functionality are typically separated
- **may be helpful in finding objects**
  - how to do this is discussed later in COMP 3004
  - only *domain* (i.e., problem as opposed to solution) objects should be mentioned in use-cases
- **are traceable to detailed interaction diagrams used later in the design process**
- **may be used as a basis for black-box testing of the system**



## Working with Use-Cases

- **Use-cases proliferate quickly:**
  - It is naive to think you can simply write down all of the use-cases and *exhaustively* describe the behavior of the system
  - We repeat, it is easy to confuse scenarios, their steps, and use-cases
- **Several authors suggest finding “key” scenarios and use-cases**
  - but no one gives good guidelines for selecting such “key” use-cases... See [Wirfs-Brock tutorial](#)
- **Don't forget about scenario interactions (next slide)**

## About Scenario Relationships

- **It is important to group together related scenarios into a single scenario *cluster*:**
  - **A use-case should be thought of as a cluster of related scenarios**
  - Exception handling scenarios can be viewed as extensions or alternatives of basic use-cases
- **Individual scenarios are typically straightforward. It is essential to capture the *relationships* between scenarios! The same holds for use-cases!!**
  - Such relationships typically define at least a temporal, if not a *causal* order between scenarios, and/or between use-cases.
  - A use-case diagram may be used to document inter-UC relationships. But we need lots of stereotypes!

## The Bottom Line

- **Don't do procedural decomposition through the use-cases**
  - don't describe algorithms or specific paths of execution **inside the system**
  - Each scenario of a UC is an end-to-end sequence of events corresponding to a typical use of the system, which is viewed as a black box.
- **Review use-cases with respect to completeness and consistency**
  - trace individual scenarios to requirements
  - inter-scenario and inter-UC relationships are crucial in verifying consistency