Use-Case Analysis

Use-Case Analysis

- What is it?
  - An informal, user-friendly, technique useful for *functional* requirements analysis and specification
- From where did it come?
  - Ivar Jacobson, a Swedish software engineer at Ericsson, now with Rational, in a *method* called OOSE (Object-Oriented Software Engineering). Originally called "Usage cases"
- Now "part of" UML

Definition of "Use Case"

- "The specification of sequences of actions that a system, subsystem, or class can perform by interacting with outside actors"

  (*UML Reference Manual*, Rumbaugh, Jacobson, and Booch).

Purpose of a "Use Case"

- "to define a piece of behavior of a [system or subsystem or class] ***without revealing the internal structure*** of the [system]

  (UML Reference Manual, Rumbaugh, Jacobson, and Booch).

UML References

Importance of Use Cases

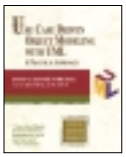
- At least one popular methodology (the Rational Unified Process, based in part on Ivar Jacobson's earlier OOSE) is said to be **Use-Case *Driven***,
- meaning that most development activities are ***traceable*** back to the use cases as defined in agreement with the user or customer.

Nonetheless

- Use cases alone do not constitute a *complete* SRS.
- For example, they focus on the **functional requirements** exclusively.

Use-Case References

recommended:




not recommended:

Other Implications

- Use cases could be used for *other* types of design, and system analysis, **not just software**.
- Once you know about them, it is hard to imagine an engineering project or business process of almost any kind starting without them.

Characteristics of Use-Case Analysis

- **Use-cases**: The specific ways in which the system is used.
- Each use-case expresses a "compete thought" or end-to-end transaction.
- A "black-box" specification; does not deal with internal structure.

Some Key **Components** of Use-Case Analysis

- **Actors**: Entities that use or are used by the system; *typically* people, but could *also* be other *systems* or *devices* as long as they are outside the system being specified.
- **Connections** from Actors to Use-Cases
- **Relationships** between Actors or between Use-Cases

Actors

- Actors are characterized not by the identity of the user or other, but rather by the ***role*** played by the actor.
- One person can be several different actors in different roles.
- One actor can be played (at different times) by several different persons.
- An entire **committee** could be **one** actor.
- An actor need not be a living thing; it could be another subsystem.

More on Actors

- Actors are **not part of the system** in question; they supply input to and receive output from, to the system.
- In other words, actors collectively define the **environment** of the system.
- This does not preclude the possibility of an object in the system **standing for** an actor.

Minimum Requirement for a Use Case

- Verbal description

Common Components of a Use Case

- Name
- Symbolic label
- List of actors
- Initiator
- Verbal description

Initiator

- The initiator of a use case is the actor that starts the flow of events.

Brief Use-Case Description

label for this use-case *name of this use-case*

- OCI: Order from catalog
  - **Actors**: customer, sales rep., shipping dept.
  - **Initiator**: customer
  - **Description**: Customer calls to order items from the catalog. The sales rep. identifies the item numbers, verifies that the items are in stock, and confirms the order with the customer, giving him the order number. The sales rep. then forwards the order to the Shipping dept.

Flow of Events

- Could be more readable for the description to be an enumerated list of events, e.g.:
  1. Customer calls to order from catalog.
  2. Sales representative identifies item numbers.
  3. Sales representative verifies stock.
  4. Sales representative confirms order.
  5. Sales representative gives order number to Customer.
  6. Sales representative passes order to Shipping.

Flow-of-event descriptions could contain *iteration*

- An order could contain multiple items. In this case, the event flow should show something like:
  - ***For each*** item to be ordered:
    - Sales representative checks catalog number.
    - Sales representative verifies stock.
    - Sales representative records item.
- Similarly, flow of events could contain conditional (if-then-else) behaviors.

Use Case Diagrams

- For *visualization* of use case interactions; diagrams are not the use cases themselves.
- Don't tell the whole story
- Useful in brainstorming
- Used in software tools, such as:
  - Rational Rose
  - iLogix Rhapsody

Icon for an Actor

Note: Actors are typically drawn in this "anthropomorphic" way even when the actors aren't people.

Examples of Actors

Customer

Shipping Dept.

Alternate Actor Icons in UML

Customer

Visual Icon

«actor»
Customer

Textual Stereotyping

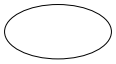
«actor»
Customer

Textual & Visual Stereotyping

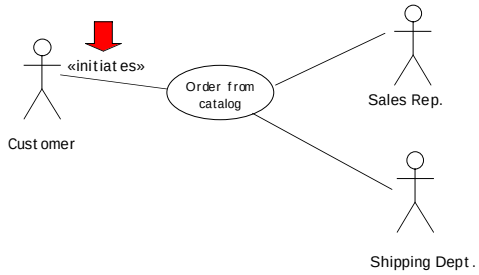
UML Use of “Stereotypes”

- The « … » notation called “guillemets”, (used for quotes in French, Italian, and Spanish)
- These usually indicate the name of a “stereotype”, defined as an *informal extension* of the UML.

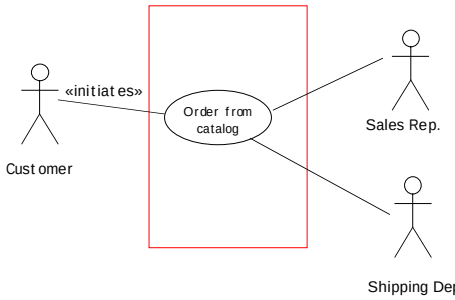
Icon for a Use Case



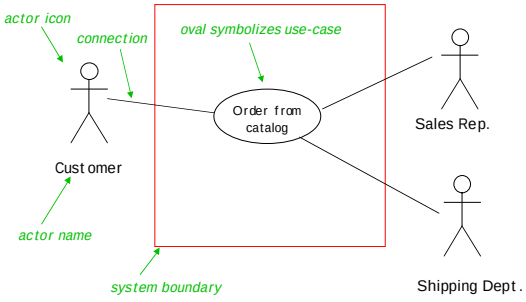
Noting the Initiator



for a mail-order catalog business



Symbology for a simple use-case



Class Exercise

- Identify several other possible use-cases in the catalog-order enterprise.
- For each use case, indicate the actors, initiator, and flow of events.

Steps in Use-Case Analysis

- Identify system boundaries
- Identify actors:
  - Recall: an actor is an entity playing a particular role with respect to the system.

Steps in Use-Case Analysis (cont'd)

- Identify use cases themselves:
  - Every use case has at least one actor.
  - A specific actor initiates the use case.
  - The same actor may participate in multiple use cases, as initiator in some and not in others.
- Create the description including flow of events
- Identify clarifying ***scenarios*** where helpful
- Provide additional information (see later)

Scenarios of a Use Case

- A "scenario" is a *single* path through the event flow. For example, if there is a conditional part, only one branch is taken in the scenario.
- Obviously we can't always **enumerate** all the scenarios; there might be an infinite set of them. If the use case involves iteration, only a finite number of iterations are used in the scenario.

Scenarios (continued)

- Often there will be a "principal" scenario, and several secondary variations.

A Catalog Order Scenario (1 of 3)

- Alice calls company.
- Bert answers the telephone.
- Alice indicates she wishes to place an order.
- Bert asks how the order will be paid.
- Alice indicates via credit card.
- Bert asks for the card number,billing address, and expiration date.
- Alice provides the above info.

A Catalog Order Scenario (2 of 3)

- Bert asks for the first item.
- Alice responds with first item.
- Bert asks for quantity of first item.
- Alice responds with quantity of first item.
- Bert records first item with quantity.
- Bert asks for second item.
- Alice responds with second item.
- Bert indicates second item out of stock; does Alice wish it to be back ordered?
- Alice declines to order item.

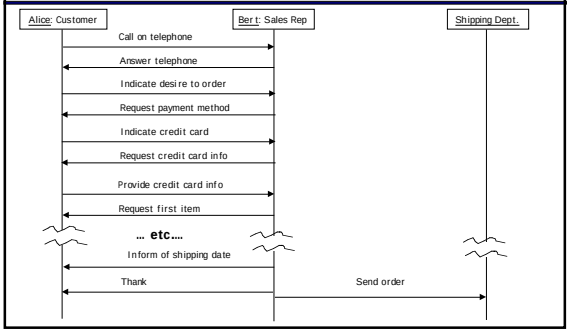
A Catalog Order Scenario (3 of 3)

- Bert asks for third item.
- Alice responds that there are no more items.
- Bert asks for shipping address.
- Alice indicates that it is the same as the billing address.
- Bert informs Alice of expected shipping date and provides order number.
- Bert thanks Alice.
- Alice hangs up.
- Bert transmits order to Shipping dept.

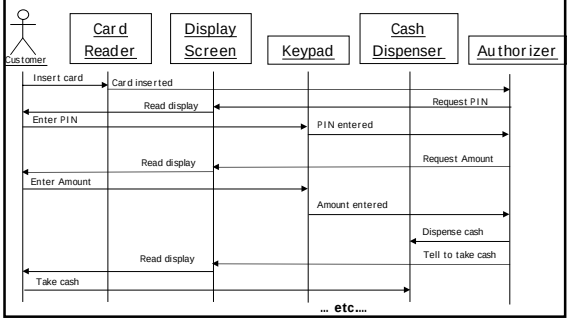
Use-Case Advice (Larry Constantine and others)

- Write in the active voice.
- Pair responses with the events that invoke them.
- Identify **domain objects** that clearly are part of the application context (such as "catalog", "inventory", "fleet" (of automobiles)).
  [A domain dictionary or glossary could be used.]

Sequence Diagram for a Scenario

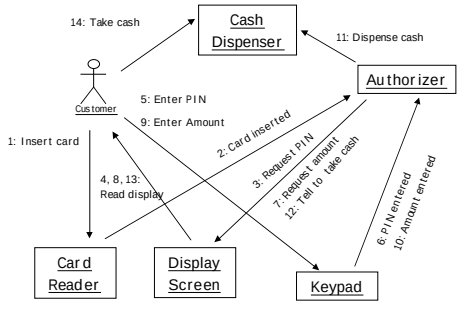


Sequence Diagram for an ATM Withdrawal Use Case



Collaboration Diagram

(= "folded" sequence diagram with message numbers)



Scenarios and Exceptions

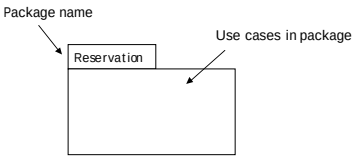
- One possible use of scenarios is in demonstrating representative exceptional or "what if" behaviors (as opposed to the principal behavior).
- **Example**: In the catalog order use case, what if the customer hangs up or connection is lost in the middle of the dialog?

Scenario Types (Bruegge)

- **Visionary** scenario: Describes future scenario
- **Evaluation** scenarios: Describe user tasks against which system is evaluated
- **Training** scenarios: Used for tutorial purposes
- **As-is** scenarios: Describe current situation (during reengineering)

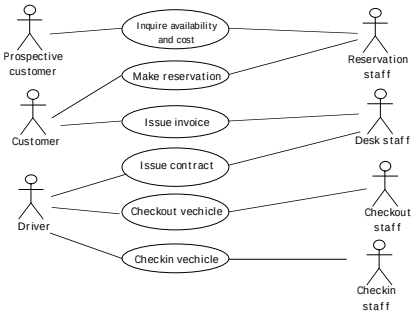
UML **Package Notation**

Used for Grouping; Could be Used to Group Use Cases

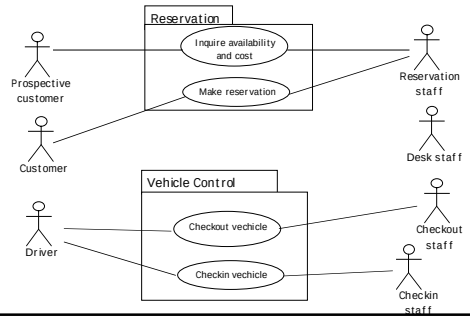


Car Rental Example

How might the use cases be packaged?



Car Rental Example with Two Packages



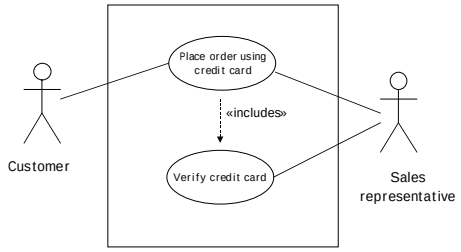
Use of Packages

Packages may be used in the transition to a design, and ultimately coding.

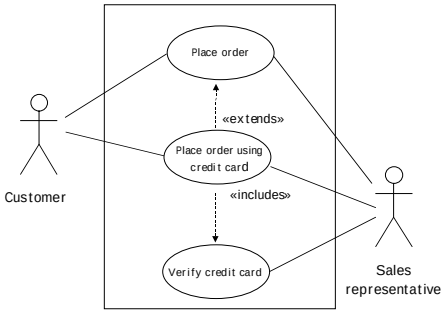
However, these connections would not normally be part of the initial discussion with the customer.

Relations between Use Cases

Inclusion among use-cases



Extension among use-cases

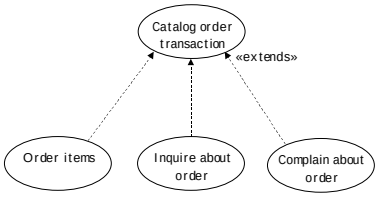


extends vs. *uses*

- **«includes»** means this use-case makes use of another use-case, as if a kind of ***subroutine***. This allows us to not have to repeat the included use-case in the description of the including use-case.
- **«extends»** means that this use-case is a ***specialization*** of another use case.

Note: «includes» was formerly called «uses».

«extends» can be used to impart a hierarchical **abstraction structure** to use cases

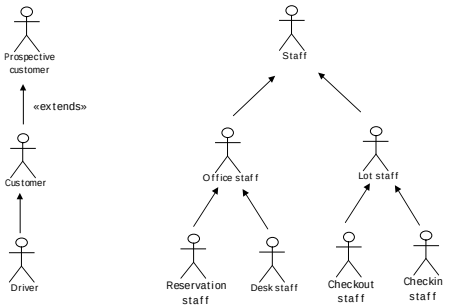


Options of a use case

- **Example**: During order processing, the sales representative offers to tell the customer about current specials.
- Such options should be mentioned as an annex to the other use case items.

Actor Hierarchies

are possible, similar to extensions



Caution about Structuring Use-Cases

- Use-case structuring is obviously *analogous* to structuring in object-oriented systems.
- However, one should ***not*** infer that use-case structure implies anything about *internal* structure of the system.

Use-cases vs. Requirements

- A use-case describes one "unit" of functionality.
- A single informally-specified functional requirement could translate into **multiple use-cases**.
- A single use-case could also be involved in satisfying **multiple requirements**.

Use-cases vs. Requirements (cont'd)

- **Collectively**, the use-cases ideally should account for **all** of the desired **functional** requirements.
- Non-functional requirements may *annotate* use-cases, but don't get represented as use-cases directly.

Further Possible Components of Use Cases

Goals

- A goal describes the **higher purpose** of the execution of the use case.
- Example: Goal for catalog order: A customer wishes to order products from the company.

Pre- and Post-Conditions

- Some use-cases are not meaningful at arbitrary times, but instead only when the system is in a state with certain properties. Such properties are called ***pre-conditions***.
- Similarly, the use-case might leave the system in a state known to satisfy one or more ***post-conditions***.

Example:

- For the car-rental enterprise, the use case "checkin vehicle" has the **pre-condition**
  vehicle is rented to driver
  and the **post-condition**:
  vehicle is on site
  & vehicle is not rented to a driver.
- For the use-case "checkout vehicle", these conditions are reversed.

Invariants

- A condition that is a pre- and post-condition for all use cases is called an ***invariant***.
- Example: Total vehicles =
  vehicles rented
  + vehicles available
  + vehicles in repair
  + vehicles in scrap.

Optional Triggers

- A ***trigger*** is an event that causes the use case to be run.
- Example: A catalog order is **triggered** by a phone call.
- This is similar to a pre-condition, but is a dynamic **event** rather than a **condition**.

Exceptions

- If a use case cannot be completed as described, an *exception* is said to occur.
- The description can indicate aspects of the state and output in such cases.

Alternative to Exceptions

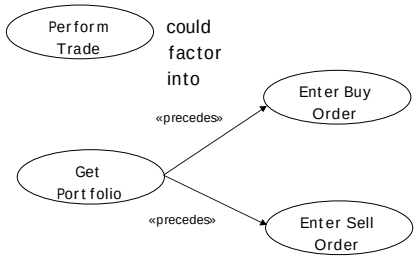
- A use case may be allowed **explicit success and failure outcomes**, each with its own post condition.

Precedence among Use-Cases

- When one use case is used to establish a pre-condition for another, the two may be linked by «precedes».
- One use of precedence is to ***factor*** a use-case into sub-cases, to avoid repetition among different sub-cases.

Precedence Example

Stock-trading example:



Subset of "Top 10 Use-Case Pitfalls"

by Susan Lilly, Software Development Magazine, Jan. 2000
http://www.cs.uwf.edu/~italbot/cen5990/lib/use_case_pitfalls.html

- The use cases aren't written so the customer can understand them.
- The system boundary is undefined or inconsistent.
- The use cases are written from the system's (not the actors') point of view.
- The actor names are inconsistent across use cases.
- The use cases don't correctly separate actors based on functional entitlement.
- The use-case specifications are too long or confusing.

CS 121

Addendum to Assignment

Due Monday

Develop a Set of Use Cases

- Regarding the requirements that were elicited in the previous exercise:
  - Enumerate in name only as many use cases as would be needed to define the scope of the software.
  - Give detailed description of any five of those use cases, using the attached template.
  - Give a use-case diagram showing your five cases together.

Use-Case Template

Use the following template for use cases:

- Label
- Name
- Goal
- Actors
- Initiator
- Description
- Pre-conditions
- Post-conditions
- Options (if present)
- Scenario, if helpful in clarifying

Traceability Matrix

- Provide a Traceability Matrix, which lists each of the requirements stated here and identifies the use cases that cover those requirements.

Additional Points on Use Cases

Do *Not* use Use-Cases to Fully Decompose into a **Design**

- Factoring should be used to ***simplify the description*** of use-cases.
- Avoid the temptation of making use-case decompositions into design.
- Use-cases are ***customer language***, ***not design language*** or pseudo-code. They describe *what*, not *how.*
- There are other tools that are better-suited to the design phase.

Uses of Use-Cases **across** Development Phases (Bruce Douglass, "Doing Hard Time")

- Analysis phase:
  - Suggest large-scale partitioning of the problem domain
  - Provide structuring of analysis objects (i.e. actors and sub-systems)
  - Clarify system and object responsibilities
  - Capture new features as they are added
  - Validate analysis model

Uses of Use-Cases across Development Phases (Bruce Douglass) (cont'd)

- Design phase:
  - ***Validate*** the elaboration of analysis models in the presence of design objects
- Coding phase:
  - ***Clarify*** purpose and role of classes for coders
  - ***Focus*** coding efforts
- Testing phase:
  - Provide test scenarios for ***validation***
- Deployment phase:
  - Suggest ***iterative prototypes*** for spiral development

Levels of Use Cases

- These ideas are from Use Cases: Requirements in Context, Daryl Kulak and Eamonn Guiney, ACM Press, 2000.
- Four **iterative levels** for specifying use cases:
  - **Façade**: Outline and high-level descriptions
  - **Filled**: Broader and deeper descriptions
  - **Focused**: Narrowing and pruning
  - **Finished**: Touch-up and fine-tuning
- See the reference for example worked out at all levels.

Façade Use-Case Components

- Name
- [Goal] (I added this.)
- Summary
- Basic course of events

Filled Use-Case Components

- Name
- [Goal]
- Summary
- Basic course of events
- Alternative paths
- Exception paths

Focused Use-Case Components

- Name
- [Goal]
- Summary
- Basic course of events
- Alternative paths
- Exception paths
- Extension [Option] points
- Trigger
- Assumptions
- Preconditions
- Postconditions
- Related ***business rules***

Business Rules

- ***Business rules*** are requirements that represent ***constraints*** on behaviors, rather than behaviors themselves.
- Examples:
  - All transactions are in U.S. Dollars.
  - Valid operators license is required in order to rent a car.
  - Late-fee is assessed for enrollment after the second week of the semester.

Finished Use-Case Components

- Same components as in the *Focused* iteration, just more polished.

UML Ways of Clarifying Complex Behaviors in Use Cases

- These are more technical and may be more appropriate in the ***design*** phase. However, sometimes they can clarify a use case:
  - **Sequence diagram**: shows messages between actors and sub-systems
  - **Collaboration diagram**: a sequence diagram organized as a directed graph rather than as a linear sequence of messages.
  - **State chart**: Elucidates behavior in terms of properties of state
  - **Timing diagram**: a sequence diagram with a time metric applied to the sequence dimension