Object Oriented Analysis and Design

Conceptual Modelling

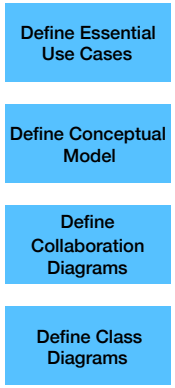
COMP3607
Object Oriented Programming II

Week 3:17-Sept-2018

Outline

- Object-Oriented Analysis
  - Use Case
  - Conceptual Model
  - Collaboration Diagram
  - Class Diagram
- Conceptual Modelling
  - Building a conceptual model
  - Adding associations
  - Adding attributes

Analysis and Design Phases



Object Oriented Analysis and Design

The single most important skill is assigning responsibilities to software components.

This affects the robustness, reusability, and maintainability of the software.

Object Oriented Analysis and Design

Critical questions to ask when designing a system:

- How should responsibilities be allocated to classes of objects?
- How should objects interact?
- What classes should do what?

Representing the Domain

Identifying a rich set of objects or concepts is at the heart of o-o analysis. This leads to an understanding of the meaningful concepts in a problem domain

Use Case

A **use case** describes processes in a domain in narrative format.

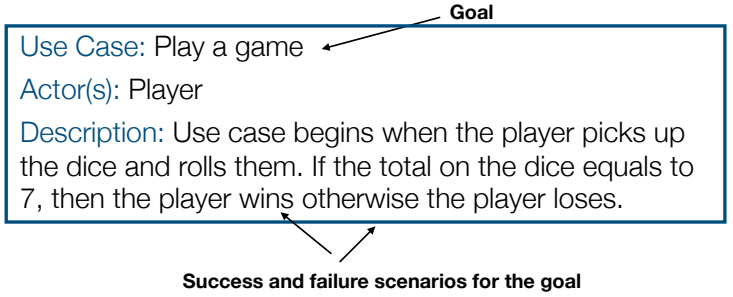
Defining use cases is a preliminary step in describing the requirements of a system.

Use cases are an important requirements analysis artifact, but they are not really object-oriented. They emphasise a **process view** of the domain.

Example: Use Case

Simple example: Dice Game

A player rolls 2 die. If the total on the dice equals to 7, the player wins otherwise the player loses.



Conceptual Model

A **conceptual model** represents real-world things, not software components.

It is represented in UML notation as a static structure diagram and may show:

- Concepts
- Associations between concepts
- Attributes of concepts

Conceptual models illustrates the different categories of things in the domain and it may clarify the roles of people involved in processes.

Example: Conceptual Model

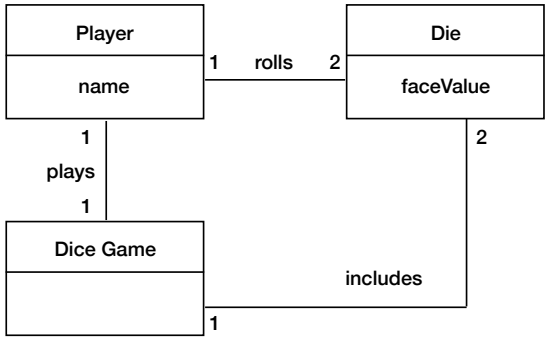


Figure 1: Conceptual Model of the Dice Game

Collaboration Diagram

A **collaboration diagram** illustrates how objects interact via messages. It shows the **flow** of **messages** and the invocation of **methods**.

Sequence diagrams are a type of collaboration diagram.

Example: Sequence Diagram

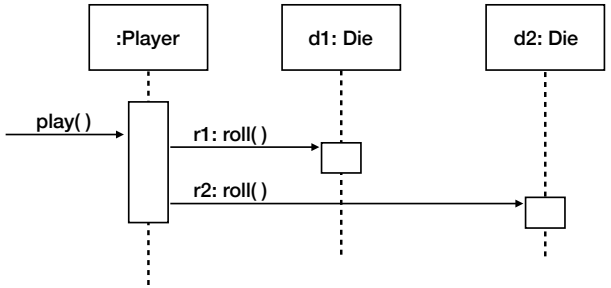


Figure 2: Sequence Diagram of the Dice Game Ilustrating Message Flow and Method Invocation

Class Diagram

A class diagram specifies how objects connect to other objects and what are the operations of the objects.

Class diagrams describe actual software components rather than real-world entities (as in the conceptual model).

Example: Class Diagram

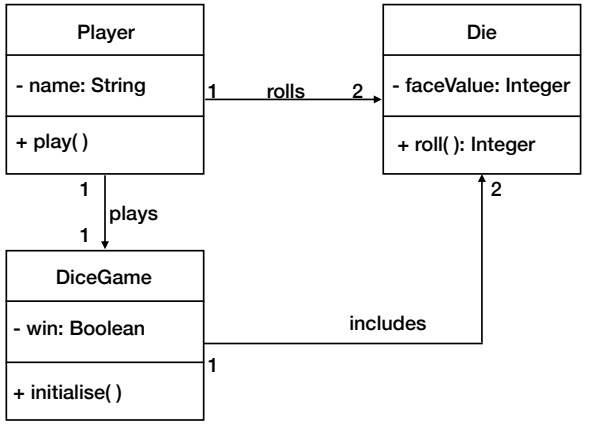


Figure 3: Class Diagram of the Dice Game

Use Cases

A use case is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process.

They are stories or cases of using a system that illustrate the requirements of the system.

Example: High Level Use Case

The following example describes the process of buying things at a store when a point-of-sale terminal is used.

Use Case: **Buy Items**

Actors: Customer, Cashier

Type: primary

Description: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects payment. On completion, the Customer leaves with the items.

Expanded Use Cases

An expanded use case shows more detail than a high level one; they are useful for gaining a deeper understanding of the processes and requirements.

These are done often in conversation style between the actors and the system.

Expanded Use Cases
-Format-

Use case: Name of use case

Actors: List of actors (external agents) indicating who initiates the use case

Purpose: Intention of the use case

Overview: Repetition of the high-level use case, or some similar summary.

Type: 1. Primary, secondary or optional

2. Essential or real

Cross References: Related use cases and system functions

Typical Course of Events

Example: Expanded Use Case

Expanded use case for the Buy Items that has been simplified to handle cash payments only and ignore inventory management (for simplicity).

Use Case: **Buy Items with Cash**

Actors: Customer (initiator), Cashier

Purpose: Capture a sale and its cash payment

Type: primary and essential

Overview: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects a cash payment. On completion, the Customer leaves with the items.

Example: Expanded Use Case

Typical Course of Events

Actor Action	System Response
1. This use case begins when a Customer arrives at a POS checkout with items to purchase	
2. The cashier records the identifier from each item. If there is more than one of the same item, the Cashier can enter the quantity as well.	3. Determines the item price and adds the item information to the running sales transaction. The description and price of the current item are presented
4. On completion of item entry, the Cashier indicates to the POS that the item entry is complete	5. Calculates and presents the sale total

Alternative Courses

- Line 2: Invalid identifier entered. Indicate error

Example: Expanded Use Case

Actor Action	System Response
6. The Cashier tells the Customer the total	
7. The Customer gives cash payment - the cash tendered- possibly greater than the sale total	
8. The Cashier records the cash received amount	9. Shows the balance due back to the Customer Generates a receipt.
10. The Cashier deposits the cash received and extracts the balance owing. The Cashier gives the balance owing and the printed receipt to the Customer	11. Logs the completed sale.
12. The Customer leaves with the items purchased	

Alternative Courses

- Line 7: Customer didn't have enough cash. Cancel sales transaction

Actors

An actor is an entity external to the system who in some way participates in the story of the use case.

An actor typically stimulates the system with input events or receives something from it.

Actors are represented by the role they play in the use case, e.g. Customer, Cashier.

Kinds of actors:

- roles that people play
- computer systems
- electrical or mechanical devices

Identifying Use Cases

- One method used to identify use cases is actor-based:
  - Identify the actors related to the system or organisation
  - For each actor, identify the processes they initiate or participate in.
- A second method to identify use cases is event-based:
  - Identify the external events that a system must respond to
  - Relate the events to actors and use cases

Systems and Boundaries

Typical system boundaries:

- hardware/software boundary of a device or computer system
- department of an organisation
- entire organisation

Defining system boundary is important in order to identify what is internal vs external.

Primary, Secondary and Optional Use Cases

Primary use cases: represent major common processes e.g. Buy Items

Secondary use cases: represent minor or rare processes such as Request for Stocking New Product

Optional use cases: represent processes that may not be tackled

Essential Use Cases

Essential use cases are :

- expanded use cases that are expressed in ideal form
- relatively free of technology and implementation details
- high-level use cases (brevity, abstraction)

Real Use Cases

Real use cases concretely describe the process in terms of :

- its real current design
- specific input and output technologies

These are design artefacts.

Conceptual Modelling

Creating a conceptual model aids in clarifying the terminology or vocabulary of a domain.

It can be viewed as a model that communicates what the important terms are, and how they are related.

Conceptual models describe things in the real world, not software design. The following elements are therefore not suitable for a conceptual model:

- No software artefacts (database, windows)
- No responsibilities or methods

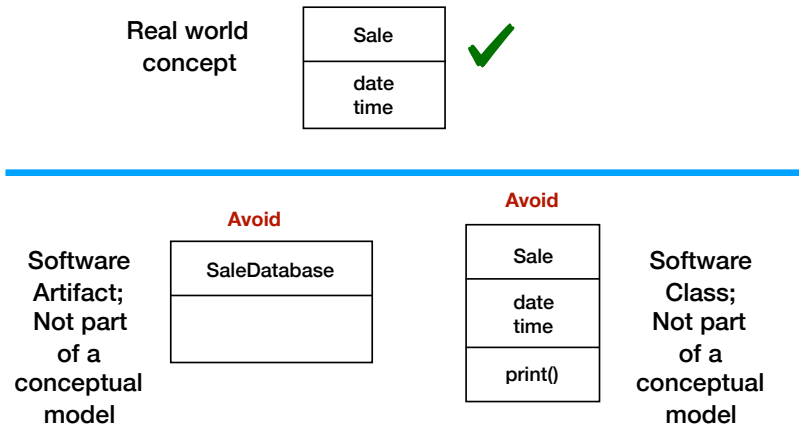
Example

Consider the domain of a retail store where a point-of-sale system is used to record the sales made and handle payments.

The concepts of payments and sales are significant in this problem domain.

Suppose the date and time of each sale is recorded.

Example



Concepts

A **concept** is an idea, thing or object. It may be formally considered in terms of its symbol, intension and extension.

- **Symbol**: words or images representing a concept
- **Intension**: the definition of a concept
- **Extension**: the set of examples to which the concept applies.

Example

Purchase transaction event:

This may be named by the symbol *Sale*.

The intension of a *Sale* may state that it represents the event of a purchase transaction, and has a date and time.

The extension of a *Sale* is all of the examples of sales, i.e., the set of all sales.

Strategies to Identify Concepts

The conceptual model identifies the different concepts in a problem domain.

A useful guideline in identifying concepts is to overspecify with lots of fine-grained concepts rather than to underspecify it.

Strategies to Identify Concepts

- **Concept category list:** physical or tangible objects, places, transactions, roles of people, containers of other things, events, organisations, processes, rules, policies, catalogues, records, services, mechanical/computer systems (external.

Make a list of candidate concepts that are worth considering.

Strategies to Identify Concepts

- **Noun phrase identification**: identify nouns and noun phrases in the textual description of a problem domain and consider whether they should be candidate concepts or attributes.

Expanded use cases are a good description to draw from for this analysis. This works well with the concept category list from the previous slide.

Concepts vs Attributes

A rule of thumb: If we do not think of some concept X as a number or text in the real world, X is probably a concept not an attribute.

Eg. Domain of airline reservations

Should destination be an attribute of *Flight*, or a separate concept, *Airport*?

In the real world, a destination airport is not considered a number or text; it is a massive thing that occupies space. There *Airport* should be a concept.