Interaction Modelling: Use Cases

Fabrizio Maria Maggi
Institute of Computer Science

(these slides are derived from the book “Object-oriented modeling and design with UML”)
Interaction Modelling: INPUT
Interaction Modelling: INPUT
Interaction Modelling: INPUT

WHAT?
Interaction Modelling: INPUT

- To answer this question, the domain model provides classes with attributes and relations among them.
- Operations are not specified.
Interaction Modelling: Overview
Interaction Modelling: Overview

How do objects interact?
Interaction Modelling: Overview
Interaction Modelling: OUTPUT

<table>
<thead>
<tr>
<th>Operations</th>
<th>carry</th>
<th>come</th>
<th>drink</th>
<th>go</th>
<th>lead</th>
</tr>
</thead>
<tbody>
<tr>
<td>ask</td>
<td>catch</td>
<td>cook</td>
<td>drive</td>
<td>hit</td>
<td>lift</td>
</tr>
<tr>
<td>bake</td>
<td>clap</td>
<td>cry</td>
<td>eat</td>
<td>hop</td>
<td>lock</td>
</tr>
<tr>
<td>bite</td>
<td>clean</td>
<td>cut</td>
<td>float</td>
<td>juggle</td>
<td>look</td>
</tr>
<tr>
<td>bounce</td>
<td>climb</td>
<td>dance</td>
<td>fly</td>
<td>jump</td>
<td>march</td>
</tr>
<tr>
<td>brush</td>
<td>close</td>
<td>dig</td>
<td>fold</td>
<td>kick</td>
<td>mix</td>
</tr>
<tr>
<td>build</td>
<td>color</td>
<td>draw</td>
<td>follow</td>
<td>knock</td>
<td>mop</td>
</tr>
<tr>
<td>call</td>
<td>comb</td>
<td>dream</td>
<td>give</td>
<td>laugh</td>
<td>open</td>
</tr>
</tbody>
</table>
Interaction Modelling: OUTPUT

Operations

- carry
- come
- drink
- go
- lead
- ask
- catch
- cook
- drive
- hit
- lift
- bake
- clap
- cry
- eat
- hop
- lock
- bite
- clean
- cut
- float
- juggle
- look
- bounce
- climb
- dance
- fly
- jump
- march
- brush
- close
- dig
- fold
- kick
- mix
- build
- color
- draw
- follow
- knock
- mop
- call
- comb
- dream
- give
- laugh
- open
Interaction Modelling: OUTPUT
Interaction Modelling: Overview

- Domain (Class) Model
- Interaction Modelling
- Application (Class) Model
- Code Generation
Interaction Modelling: Overview

Domain (Class) Model

Classes; Attributes; Relations

Interaction Modelling

Application (Class) Model

Code Generation
Interaction Modelling: Overview

Instrument for identifying the right operations

Domain (Class) Model → Interaction Modelling → Application (Class) Model

Code Generation
Interaction Modelling: Overview

Domain (Class) Model → Interaction Modelling → Application (Class) Model

Code Generation

Classes; Attributes; Relations; Operations
Interaction Modelling

- Interactions can be modeled at different levels of abstraction
  - At a high level use cases describe how a system interacts with outside actors
    - Each use case represents a functionality that a system provides to the user
    - Use cases are helpful for capturing informal requirements
  - Sequence diagrams provide more details about which operations need to be invoked in a specific scenario
Use Case Models: Actors

- An actor is a direct external user of a system
  - An object or a set of objects that communicates directly with the system but that is not part of the system
- Examples
  - Customer and Repair Technician are actors of a vending machine
  - Traveler, Agent and Airline are actors of a travel agency system
  - User and Administrator are actors for a computer database system
Use Case Models: Actors

- Actors can be persons, devices and other systems (anything that interacts directly with the system)
Use Case Models: Actors

- An actor represents a particular facet (i.e., role) of objects in its interaction with a system.
- The same actor can represent different objects that interact similarly with a system.
  - E.g., many individual persons may use a vending machine but their behavior toward the vending machine can be summarized by the actors Customer and Repair Technician.
  - Each actor represents a coherent set of capabilities for its objects.
Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary.

An actor is directly connected to the system.

An indirectly connected object is not an actor and should not be included as part of the system model.

Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine.

Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model.
Modelling the actors helps to define a system by identifying the objects within the system and those on its boundary.

- An actor is directly connected to the system.
  - An indirectly connected object is not an actor and should not be included as part of the system model.
    - Example: the Dispatcher of repair technicians from a service bureau is not an actor of a vending machine.
      - Model a repair service that includes Dispatchers, Repair Technicians and Vending Machines as actors and use a different model for the vending machine model.
A use case is a coherent piece of functionality that a system can provide by interacting with actors.

- **Buy a beverage.** The vending machine delivers a beverage after a customer selects and pays for it.
- **Perform scheduled maintenance.** A repair technician performs the periodic service on the vending machine necessary to keep it in good working condition.
- **Make repairs.** A repair technician performs the unexpected service on the vending machine necessary to repair a problem in its operation.
- **Load items.** A stock clerk adds items into the vending machine to replenish its stock of beverages.

*Figure 7.1 Use case summaries for a vending machine. A use case is a coherent piece of functionality that a system can provide by interacting with actors.*

Use Case Models: Use Cases

- Each use case involves one or more actors as well as the system itself
  - Examples: the use case “Buy a beverage” involves the Customer; the use case “Perform scheduled maintenance” involves the Repair Technician; in a telephone system the use case “Make a call” involves two actors, a Caller and a Receiver

- An actor is not necessarily a person
  - Example: in an online shop the use case “Checkout” involves the Web Customer and the Credit Payment Service

- A use case partitions the functionality of the system into a mainline behavior sequence, variations on normal behavior, exception conditions, error conditions, cancellations of a request

- Use cases should all be at a comparable level of abstraction
  - Examples: “Make telephone call” and “Record voice mail message” are at a comparable level; “Set external speaker volume to high” is too narrow, “Set speaker volume” or even “Set telephone parameters” would be better
Creating Use Case Models

- Use case models include
  - Use case diagrams
  - (Textual) use case descriptions
UML has a graphical notation for summarizing use cases into use case diagrams

- A rectangle contains the use cases for a system with the actors listed on the outside
- The name of the system is written near a side of the rectangle
- A name within an ellipse denotes a use case
- A “stick man” icon denotes an actor with the name placed below the icon
- Solid lines connect use cases to participating actors
Actor Generalization

- The child actor inherits all use case associations from the parent
- Actor generalization should be used if the specific actor has more responsibility than the generalized one (i.e., associated with more use cases)
  - Example: Look at the requirements management use case diagram in the picture and you will see there is duplicate behavior in both the buyer and seller which includes “Create an account” and “Search listings”
  - Rather than having all of this duplication, we will have a more general user that has this behavior and then the actors will “inherit” this behavior from the general user
Use Case Relationships

- For large applications complex use cases can be built from smaller pieces
- Linking enables flexibility in requirements specification
  - Isolating functionality
  - Enabling functionality sharing
  - Breaking functionality into manageable chunks
- Two mechanisms are used:
  - Include
  - Extend
Use Case Relationships: Include

- **Include Relationship**
  - A use case can make use of other smaller use cases
  - The include relationship incorporates the behaviour of another use case (e.g., subroutines)

- Factoring a use case into pieces is appropriate when the pieces represent significant behaviour units
Use Case Relationships: Include

- The UML notation for an include relationship is a dashed arrow from the source (including) to the target (included) use case. The keyword «include» annotates the arrow.
Use Case Relationships: Extend

- Extend Relationship
  - Adds an “extra behaviour” to a base use case
  - Is used in the situation in which some initial capability is defined and later features are added modularly
  - Base use case is meaningful on its own, it is independent of the extension. Extension typically defines optional behavior that is not necessarily meaningful by itself
The UML notation for an extend relationship is a dashed arrow from the extension to the base use case. The keyword <<extend>> annotates the arrow.

- Use case “trade stocks” is meaningful on its own. It could be optionally extended with “margin trading”.
- **Extension Points**: specify the location at which the behavior of the base use case may be extended. Extension points can have a condition attached. The extension behaviour occurs only if the condition is true when the control reaches the extension point.

![Diagram showing use case relationships: Extend](image)
A Note for the Reader

What's is the difference between include and extend in use case diagram?

**Extend** is used when a use case conditionally adds steps to another first class use case. For example, imagine "Withdraw Cash" is a use case of an ATM machine. "Assess Fee" would extend Withdraw Cash and describe the *conditional* "extension point" that is instantiated when the ATM user doesn't bank at the ATM's owning institution. Notice that the basic "Withdraw Cash" use case stands on its own, without the extension.

**Include** is used to extract use case fragments that are *duplicated* in multiple use cases. The included use case cannot stand alone and the original use case is not complete without the included one. This should be used sparingly an only in cases where the duplication is significant and exists by design (rather than by coincidence). For example, the flow of events that occurs at the beginning of every ATM use case (when the user puts in their ATM card, enters their PIN, and is shown the main menu) would be a good candidate for an *include*. 
A Note for the Reader

What's is the difference between include and extend in use case diagram?

**Extend** is used when a use case conditionally adds steps to another first class use case. For example, imagine "Withdraw Cash" is a use case of an ATM machine. "Assess Fee" would extend Withdraw Cash and describe the *conditional* "extension point" that is instantiated when the ATM user doesn't bank at the ATM's owning institution. Notice that the basic "Withdraw Cash" use case stands on its own, without the extension.

**Include** is used to extract use case fragments that are *duplicated* in multiple use cases. The included use case cannot stand alone and the original use case is not complete without the included one. This should be used sparingly an only in cases where the duplication is significant and exists by design (rather than by coincidence). For example, the flow of events that occurs at the beginning of every ATM use case (when the user puts in their ATM card, enters their PIN, and is shown the main menu) would be a good candidate for an **include**.
Use Case Descriptions

- A use case diagram is equipped with a description that follows the template described in: https://courses.cs.ut.ee/MTAT.03.083/2016_fall/uploads/Main/UseCaseDescriptions.pdf

- Each ellipse (use case) in the use case diagram is associated to a different description (one table per use case)