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

<table>
<thead>
<tr>
<th>Domain Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Images of various objects representing domain model" /></td>
</tr>
</tbody>
</table>
Interaction Modelling: INPUT

Domain Model

WHAT?
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>juggl</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

<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>comb</td>
<td>dream</td>
<td>give</td>
<td>laugh</td>
<td>open</td>
</tr>
<tr>
<td>call</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Application Model
Interaction Modelling: OUTPUT

HOW?
Interaction Modelling: Overview

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

Code Generation
Interaction Modelling: Overview

Domain (Class) Model

Interaction Modelling

Application (Class) Model

Classes; Attributes; Relations

Code Generation
Interaction Modelling: Overview

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

Instrument for identifying the right operations

Code Generation
Interaction Modelling: Overview

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

Classes; Attributes; Relations; Operations
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
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.
Use Case Models: Actors

- 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.
Use Case Models: Use Cases

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.

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

A use case partitions the functionality of the system into a mainline behavior sequence, variations on normal behavior (e.g., exception conditions, error conditions)

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.

[Diagram showing actor generalization]
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)
- Improves modularity of use cases. The use case that includes another use case is not meaningful without the included one.
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 behavior” to a base use case
  - The 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 “Registration” is meaningful on its own. It could be optionally extended with “Get Help On Registration.”

**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 behavior occurs only if the condition is true when the control reaches the extension point.
Use Case Relationships: Generalization

- The UML notation for an generalization relationship is an arrow with its tail on the child use case and a triangular arrowhead on the parent use case (the same notation that is used for classes)
  - Use case “make trade” can be specialized into child use cases “trade bonds”, “trade stocks” and “trade options” (based on different types of financial items)
  - The parent use case contains steps that are performed for any kind of trade
  - Each child contains additional steps that are particular to a specific kind of trade
Exercise

- Register a manager (with name surname and level), register an employee (with name and surname), register an auditor (with name surname and type) into the system
Exercise

- Register a manager (with name surname and level), register an employee (with name and surname), register an auditor (with name surname and type) into the system.
- Process card transaction, if there is the risk that the transaction is fraudulent start the procedure to block the card.
Exercise

- Register a manager (with name surname and level), register an employee (with name and surname), register an auditor (with name surname and type) into the system
- Process card transaction, if there is the risk that the transaction is fraudulent start the procedure to block the card
- Insert PIN, authorize payment, view account history, change online bank settings
Use Case Descriptions

- A use case diagram is equipped with a description that follows the template described in:
- Each ellipse (use case) in the use case diagram is associated to a different description (one table per use case)
- The *inclusions* are steps consisting in a reference to included cases
- The *extensions* are represented in the extensions section
- All the use cases involved in a *generalization* have in general only one description in which the parts specific for the specific sub-cases must be explicitly specified.
Exercise 1: Use cases?

- A secretary of the CEO office of a company can create a new company in the CompanyManager system with all its parts. Each company is identified by a name. A company consists of departments. Existing departments can be added to the company. When a department is created, it is identified by a name. Each department is located in one or more offices. Each office is in (identified by) a certain address. Each department has a manager and a set of employees. Each manager and employee has a (unique) name and a title. The secretary can add an office, a manager and employees to an existing department. A human resource officer can add employees and a manager to a department (and create them) and promote an employee to a manager. All users can get all the employees with a certain title from a department located at a certain address.
Exercise 2: Use cases?

- A SchoolManagement system contains the list of all the students of a school. A student is associated to a name, a (unique) student ID and a date of birth. Each student can take at most 6 courses. Each course has a title and one professor. Each professor has a (unique) name, an office number and a consultation time. A professor can teach several courses. The system should provide a functionality to first level administrative officers that given a student returns the list of his/her professors with name, office and consultation time. The same functionality should be provided to a student to retrieve the list of his/her professors. The system should also provide a functionality to first level administrative officers and professors to get all the students registered for a certain course. Together with all the functionalities available for first level administrative officers, the system should allow the second level administrative officers to enroll a student, to register an enrolled student for an existing course, to create a professor, to create a new course assigning it to a professor.
Exercise 3: Use cases?

- A CinemaBooking system should store seat bookings for multiple theatres. A theatre has a name and an address. Each theatre has seats arranged in rows. A customer can check the availability of seats for a certain show (i.e., the screening of a given movie at a certain time) in one of the theaters. Shows are at an assigned date and time, and scheduled in the theatre where they are screened. Customers can reserve seats for a show and are given a row number and seat number. They may request bookings of several seats. The system stores the customers telephone numbers. Customers can cancel a booking up to 4 hours before the show. Finally, a theater responsible can ask the system administrator to create a new theater and get the telephone number of a customer that has booked a certain seat.
Exercise 4: Use cases?

- A paper reviewing system can be used to manage several conferences. All the users need to have a registered profile identified by a (unique) name and an affiliation. Each conference has a title and a year and is managed by a chair. A chair can create a conference and add committee members to it. The chair can open the submissions where a deadline for submitting a paper is set. An author can submit a paper with a title before the submission deadline (a paper ID is automatically generated). A conference has several submitted papers, but a paper can be submitted to only one conference. Given a paper ID, the chair and the committee members can retrieve the paper title and the list of authors with their names and affiliations. After the submission deadline, the chair can assign a paper to 3 reviewers taken from the committee members. This can be done after accessing the paper details. A reviewer can set his/her decision about a paper by changing its status from under review to accepted or rejected. This can be done after accessing the paper details. In order to send a reminder to reviewers, the system should provide a functionality to the chair to get the e-mails of all the reviewers of the papers under review. When the chair closes the reviews the status of a paper becomes accept if the paper has at least 2 reviewers giving the score accept to it or reject if the paper has at least 2 reviewers giving the score reject to it. In order to notify the authors about the final decision on a paper the system should provide a functionality to the chair to get the emails of all the authors of the accepted papers and of all the authors of the rejected papers separately.