Interaction Modelling: Sequence Diagrams

Fabrizio Maria Maggi
Institute of Computer Science

(These slides are derived from the book “Object-oriented modeling and design with UML”)
Software Development Methodology

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

Code Generation
Interaction Modelling
Interaction Modelling

How do objects interact?
Interaction Modelling
Software Development Methodology

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

Code Generation
Software Development

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

Domain Classes; Application Classes (e.g., Patterns); Attributes; Relations; Operations

Code
Application (Class) Model
# Application (Class) Model

<table>
<thead>
<tr>
<th>Operations/Interfaces</th>
<th>carry</th>
<th>come</th>
<th>drink</th>
<th>go</th>
<th>lead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ask</td>
<td>catch</td>
<td>cook</td>
<td>drive</td>
<td>hit</td>
</tr>
<tr>
<td></td>
<td>bake</td>
<td>clap</td>
<td>cry</td>
<td>eat</td>
<td>hop</td>
</tr>
<tr>
<td></td>
<td>bite</td>
<td>clean</td>
<td>cut</td>
<td>float</td>
<td>juggle</td>
</tr>
<tr>
<td></td>
<td>bounce</td>
<td>climb</td>
<td>dance</td>
<td>fly</td>
<td>jump</td>
</tr>
<tr>
<td></td>
<td>brush</td>
<td>close</td>
<td>dig</td>
<td>fold</td>
<td>kick</td>
</tr>
<tr>
<td></td>
<td>build</td>
<td>color</td>
<td>draw</td>
<td>follow</td>
<td>knock</td>
</tr>
<tr>
<td></td>
<td>call</td>
<td>comb</td>
<td>dream</td>
<td>give</td>
<td>laugh</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>open</td>
</tr>
</tbody>
</table>
Application (Class) Model
Application (Class) Model
Interaction Modelling: Detailing Use Cases with Scenarios

**Use Case:** Buy a beverage

**Summary:** The vending machine delivers a beverage after a customer selects and pays for it.

**Actors:** Customer

**Preconditions:** The machine is waiting for money to be inserted.

**Description:** The machine starts in the waiting state in which it displays the message “Enter coins.” A customer inserts coins into the machine. The machine displays the total value of money entered and lights up the buttons for the items that can be purchased for the money inserted. The customer presses a button. The machine dispenses the corresponding item and makes change, if the cost of the item is less than the money inserted.

**Exceptions:**

- **Canceled:** If the customer presses the cancel button before an item has been selected, the customer’s money is returned and the machine resets to the waiting state.
- **Out of stock:** If the customer presses a button for an out-of-stock item, the message “That item is out of stock” is displayed. The machine continues to accept coins or a selection.
- **Insufficient money:** If the customer presses a button for an item that costs more than the money inserted, the message “You must insert $nn.nn more for that item” is displayed, where $nn.nn is the amount of additional money needed. The machine continues to accept coins or a selection.
- **No change:** If the customer has inserted enough money to buy the item but the machine cannot make the correct change, the message “Cannot make correct change” is displayed and the machine continues to accept coins or a selection.

**Postconditions:** The machine is waiting for money to be inserted.

---

*Figure 1.2: Use case description. A use case brings together all of the behavior relevant to a role of system functionality.*

The creation of a sequence diagram should be driven by the objective of writing interfaces for the classes of a domain model by reasoning on their interactions in the implementation of the required functionalities.
A sequence diagram is used to support the definition of the interfaces through the identification of the operations that the classes need to expose for implementing the required functionalities.

A sequence diagram is used to document where the identified operations come into play in the implementation of the required functionalities.
Sequence Diagrams

Sequence diagrams show operation calls.
The period of time of an object’s execution is a thin rectangle called *activation* or *focus of control*.

An activation shows the time period during which a call of an operation is processed including the time when the called operation invokes other operations.
Sequence Diagrams

An activation has a call arrow coming into its top and a return arrow leaving its bottom.
Sequence Diagrams

The body of the operation is made of all the interactions that occur between the call arrow and the return arrow.
Sequence Diagrams

The period of time when an object exists but is not active is shown as a dashed line.
The entire period of time when an object exists is called *lifeline*. 
Sequence Diagrams

The notation for a call is an arrow from the calling activation to the activation created by the call.

IN MagicDraw:
Message
A return of a call is a dashed arrow from the bottom of the called activation to the calling activation.
If an object does not exist at the beginning of the sequence diagram, it must be created. UML shows creation by placing the object symbol at the head of the dashed arrow representing the call that creates the object.
Sequence Diagrams

An object can call its own operations (self calls)
Advanced Sequence Diagrams: alt

http://www.ibm.com

Systems modelling – Fabrizio Maria Maggi
Advanced Sequence Diagrams: opt

1. **register**: RegisterOffice
   - **getPastDueBalance** (studentId)
   - pastDueBalance

2. **ar**: AccountsReceivable
   - addStudent (studentId)
   - getClassCost()
   - chargeForClass()

3. **drama**: Class
   - pastDueBalance = 0
   - getClassCost()
Sequence Diagrams: Note for the reader

Useful material about sequence diagrams can be found on the course website:

https://courses.cs.ut.ee/MTAT.03.083/2016_fall/uploads/Main/MaterialSD
The Entity-Control-Boundary Pattern

**Entities**
Objects representing system data, often from the domain model.

**Boundaries**
Objects that interface with system actors (e.g. a user or external service). Windows, screens and menus are examples of boundaries that interface with users.

**Controls**
Objects that mediate between boundaries and entities. These serve as the glue between boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions. It is important to understand that you may decide to implement controllers within your design as something other than objects – many controllers are simple enough to be implemented as a method of an entity or boundary class for example.

Four rules apply to their communication:

1. Actors can only talk to boundary objects.
2. Boundary objects can only talk to controllers and actors.
3. Entity objects can only talk to controllers.
4. Controllers can talk to boundary objects and entity objects, and to other controllers, but not to actors

Communication allowed:

<table>
<thead>
<tr>
<th></th>
<th>Entity</th>
<th>Boundary</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boundary</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
The Entity-Control-Boundary Pattern

```
1. getBalance(accountNumber)

2. retrieveLedger()

3. ledger

4. retrieveAccount(accountNumber)

5. buyerAccount

6. getBalance()

7. balance
```

Functionality to be implemented
The Entity-Control-Boundary Pattern

Functionality to be implemented

Most of the calls come from the controller that plays the role of the "orchestrator" in the implementation of the functionality.

http://www.ibm.com

System modelling – Fabrizio Maria Maggi
From a Domain model to an Application model
From a Domain model to an Application model
# Application (Class) Model

<table>
<thead>
<tr>
<th>Operations/Interfaces</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ask</td>
<td>carry</td>
</tr>
<tr>
<td>catch</td>
<td>come</td>
</tr>
<tr>
<td>cook</td>
<td>drink</td>
</tr>
<tr>
<td>drive</td>
<td>go</td>
</tr>
<tr>
<td>hit</td>
<td>lead</td>
</tr>
<tr>
<td>lift</td>
<td></td>
</tr>
<tr>
<td>bake</td>
<td>clap</td>
</tr>
<tr>
<td>cry</td>
<td>eat</td>
</tr>
<tr>
<td>hop</td>
<td>lock</td>
</tr>
<tr>
<td>bite</td>
<td>clean</td>
</tr>
<tr>
<td>cut</td>
<td>float</td>
</tr>
<tr>
<td>juggle</td>
<td>look</td>
</tr>
<tr>
<td>bounce</td>
<td>dance</td>
</tr>
<tr>
<td>climb</td>
<td>fly</td>
</tr>
<tr>
<td>march</td>
<td></td>
</tr>
<tr>
<td>brush</td>
<td>close</td>
</tr>
<tr>
<td>dig</td>
<td>fold</td>
</tr>
<tr>
<td>kick</td>
<td>mix</td>
</tr>
<tr>
<td>mix</td>
<td></td>
</tr>
<tr>
<td>build</td>
<td>color</td>
</tr>
<tr>
<td>draw</td>
<td>follow</td>
</tr>
<tr>
<td>knock</td>
<td></td>
</tr>
<tr>
<td>call</td>
<td>comb</td>
</tr>
<tr>
<td>dream</td>
<td>give</td>
</tr>
<tr>
<td>laugh</td>
<td></td>
</tr>
<tr>
<td>open</td>
<td></td>
</tr>
</tbody>
</table>
Invoked on an object of type Bank -> it should be provided in the interface of class Bank
Invoked on an object of type AccountLedger -> it should be provided in the interface of class AccountLedger
Invoked on an object of type CheckingAccount -> it should be provided in the interface of class CheckingAccount
Application (Class) Model
From a Domain model to an Application model
Application Model

• For creating an application model, you start from a domain model where the operations of the classes are not specified and you only have classes representing *entities* of the domain. Then, through interaction modeling you understand what operations are needed to implement a certain functionality and what additional application classes (outside the domain) are needed to implement it (e.g., Boundary and Controllers).

• **The application model is the class model obtained by adding operations and application classes to the domain model.**
Application Model

• It is possible to add redundant associations to access the data more efficiently
• It is possible to specify the direction of certain associations
Exercise 1: Interfaces?

- Each company has a name. A company consists of departments. Each department has a name and is located in one or more offices. Each office is in a certain address. Each department has a manager and a set of employees. Each employee has a name and a title. One of the functionalities required in the system is to add a new department with a given name, manager and an office in a given address to a company. Then given a department a new employee can be added to the list of employees of that department. Another functionality requires to promote an employee to manager. Finally it is needed to get all the employees with a certain title from a department located at a certain address.
Exercise 2: Interfaces?

- A school has a name and can have many students. A student is associated to a name, a student ID and a date of birth. Each student has to take a course but one student can take at most 6 courses. Each course has a title. For a course there is at least one student in the school who has taken the course. Each course has one professor. Each professor has a name and a room number. A professor can teach several courses. The system should provide a functionality that given a student returns the list of his/her professors with name, office and consultation time. The system should also provide a functionality to get all the students registered for a certain course. The system should allow the school to enroll a student, to register an enrolled student for an existing course, to create a new course assigning it to a professor identified by a name.
Exercise 2: Interfaces?

- A school has a name and can have many students. A student is associated to a name, a student ID and a date of birth. Each student has to take a course but one student can take at most 6 courses. Each course has a title. For a course there is at least one student in the school who has taken the course. Each course has one professor. Each professor has a name and a room number. A professor can teach several courses. The system should provide a functionality that given a student returns the list of his/her professors with name, office and consultation time. The system should also provide a functionality to get all the students registered for a certain course. The system should allow the school to enroll a student, to register an enrolled student for an existing course, to create a new course assigning it to a professor identified by a name. OTHER USEFUL FUNCTIONALITIES?
Exercise 3: Interfaces?

- The cinema booking system should store seat bookings for multiple theatres. A theatre has a name and an address. Each theatre has seats arranged in rows. Customers can reserve seats and are given a row number and seat number. They may request bookings of several adjacent seats. Each booking is for a particular show (i.e., the screening of a given movie at a certain time). Shows are at an assigned date and time, and scheduled in a theatre where they are screened. The system stores the customers telephone numbers. A user of the system should be allowed to check the availability of seats for a certain show in one of the theaters. Also it should be possible to book a show in one of the theaters and to cancel a booking up to 4 hours before the show. Finally, a functionality is needed to get the telephone number of a customer that has booked a certain seat.
Exercise 4: Interfaces?

- A paper reviewing system has several conferences. Each conference has a title and a year and is managed by a chair and a list of committee members. Committee members and chairs must be assigned to one, but possibly more conferences. They have a name and an affiliation. A conference has several submitted papers, but a paper can be submitted to only one conference. A paper is assigned to 3 reviewers taken from the committee members. A paper can be accepted, rejected, or under review. We also know the paper titles and list of authors with their names and affiliations. In order to notify the authors about the final decision on a paper, the system should provide a functionality to get the emails of all the authors of the accepted papers and of all the authors of the rejected papers separately. In order to send a reminder to reviewers, the system should provide a functionality to get the emails of all the reviewers of the papers under review. It should be possible to set the status of a paper to accept if the paper has at least 2 reviewers giving the score „accept“ to it. It should be possible to set the status of a paper to reject if the paper has at least 2 reviewers giving the score „reject“ to it.