Object Interactions
UML Sequence Charts

Software Requirements and Design
CITS 4401
Lecture 6

Based on Ch 5 Lecture Notes by Bruegge & Dutoit

The SetTime use case: a sequence diagram

A SimpleWatch: Use case diagram

Sequence Diagrams
- ... describe use cases with objects, showing how the behaviour of a use case is distributed amongst its participating objects.
- ... provide a shift in perspective, allowing developers to find missing objects or uncertainties.
- Developers should focus on problematic or under-specified functionality first, since drawing Sequence Diagrams can be time consuming.
**Start with Flow of Events from the Use Case**

- Flow of events from “Dial a Number”
  - Caller lifts receiver
  - Dial tone begins
  - Caller dials
  - Callee phone rings
  - Callee answers phone
  - Ringing stops
  - ....

**Extracting Events**

- From the flow of events in the use case or scenario proceed to the sequence diagram
- A sequence diagram is a graphical description of objects participating in a use case or scenario
- Relation to object identification:
  - Objects/classes have already been identified during object modelling
  - New Objects are identified as a result of dynamic modelling
- Heuristic:
  - An event always has a sender and a receiver. Find them for each event => These are the objects participating in the use case

**An Example**

- Flow of events in a “GetSeatPosition” use case:
  1. Establish connection between smart card and onboard computer
  2. Establish connection between onboard computer and sensor for seat
  3. Get current seat position and store on smart card
- Which are the objects?

**Sequence Diagram for “Get SeatPosition”**

- Establish connection between smart card and onboard computer
- Establish connection between onboard computer and sensor for seat
- Get current seat position and store on smart card

```
1. Establish Connection
2. Accept Connection
3. Get SeatPosition
```

```
"500,575,300"
```
Heuristics for Sequence Diagrams (1)

- **Layout:**
  - 1st column: Should correspond to the actor who initiated the use case.
  - 2nd column: Should be a boundary object with which the actor interacts to initiate the use case.
  - 3rd column: Should be the control object that manages the rest of the use case.

- **Creation:**
  - Control objects are created by boundary objects initiating the use case.
  - Other boundary objects are created by control objects.

Heuristics for Sequence Diagrams (2)

- **Access:**
  - Entity objects are accessed by control and boundary objects.
  - Entity objects should never access boundary or control objects: This makes it easier to share entity objects across use cases and makes entity objects resilient against technology-induced changes in boundary objects.

Is this a good Sequence Diagram?

- Did the modeller follow the heuristics?

ARENA Sequence Diagram
State Chart Diagram vs Sequence Diagram

- State chart diagrams help to identify:
  - Changes to objects over time

- Sequence diagrams help to identify
  - The temporal relationships between objects over time
  - Sequence of operations as a response to one or more events

Summary (1): Requirements Analysis

- 1. What are the transformations?
  - Create scenarios and use case diagrams
    - Talk to client, observe, get historical records, do thought experiments

- 2. What is the structure of the system?
  - Create object and class diagrams
    - Identify objects. What are the associations between them? What is their multiplicity?
    - What are the attributes of the objects?
    - What operations are defined on the objects?

Summary (2): Requirements Analysis

- 3. What is its control structure?
  - Create state diagrams
    - Only for the dynamically interesting objects.
  - Create sequence diagrams
    - Identify senders and receivers of events
    - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
When is a model dominant?

- **Functional model**: The model performs complicated transformations such as difficult computations consisting of many steps.
- **Object model**: The system has non-trivial data structures.
- **Dynamic model**: The model has many different types of events: Input, output, exceptions, errors, etc.

When is a model dominant? Examples

- **Compiler**: Functional model most important. Dynamic model is trivial because there is only one type input and only a few outputs.
- **Database systems**: Object model most important. Functional model is trivial, because their purpose is usually only to store, organize and retrieve data.
- **Spreadsheet program**: Functional model most important. Object model is trivial, because the spreadsheet values are trivial and cannot be structured further. The only interesting object is the cell.