Lecture 9: Modelling System Interactions

Interactions with the new system
- How will people interact with the system?
- When/Why will they interact with the system?

Use Cases
- introduction to use cases
- identifying actors
- identifying cases
- Advanced features

Sequence Diagrams
- Temporal ordering of events involved in a use case

Moving towards specification

What functions will the new system provide?
- How will people interact with it?
- Describe functions from a user’s perspective

UML Use Cases
- Used to show:
  - the functions to be provided by the system
  - which actors will use which functions
- Each Use Case is:
  - a pattern of behavior that the new system is required to exhibit
  - a sequence of related actions performed by an actor and the system via a dialogue.

An actor is:
- anything that needs to interact with the system:
  - a person
  - a role that different people may play
  - another (external) system.

Use Case Diagrams
- Capture the relationships between actors and Use Cases

Notation for Use Cases

Staff contact

Change client contact

Actor

Communication association

System boundary

Use case
**Example**

- Add new staff member
- Add new staff grade
- Change rate for staff grade
- Change grade for staff member
- Calculate staff bonuses

**Sample use cases for a car**

- Driver
- Gas Attendant
- Mechanic
- Drive
  - Fill Up
  - Check Oil
  - Fix Car
  - <<uses>>
  - Turn On Engine
  - <<uses>>

**<<extends>> and <<uses>>**

- <<extends>> when one use case adds behavior to a base case
  - used to model a part of a use case that the user may see as optional system behavior
  - also models a separate sub-case which is executed conditionally.
- <<uses>>: one use case invokes another (like a procedure call);
  - used to avoid describing the same flow of events several times
  - puts the common behavior in a use case of its own.

**Meeting Scheduler Example**

- Initiator
  - Generate Schedule
  - <<uses>>
- Participant
  - Withdraw
  - Edit Constraints
  - Provide constraints
  - Schedule meeting
  - <<uses>>
  - Validate User
  - <<uses>>

- Print Campaign Summary
- Find Campaign
- Check Campaign Budget
- <<extends>>
- <<uses>>
Identifying Actors

- Ask the following questions:
  - Who will be a primary user of the system? (primary actor)
  - Who will need support from the system to do her daily tasks?
  - Who will maintain, administrate, keep the system working? (secondary actor)
  - Which hardware devices does the system need?
  - With which other systems does the system need to interact with?
  - Who or what has an interest in the results that the system produces?

- Look for:
  - the users who directly use the system
  - also others who need services from the system

Finding Use Cases

- For each actor, ask the following questions:
  - Which functions does the actor require from the system?
  - What does the actor need to do?
  - Does the actor need to read, create, destroy, modify, or store some kinds of information in the system?
  - Does the actor have to be notified about events in the system?
  - Does the actor need to notify the system about something?
  - What do those events require in terms of system functionality?
  - Could the actor’s daily work be simplified or made more efficient through new functions provided by the system?

Documenting Use Cases

- For each use case:
  - prepare a “flow of events” document, written from an actor’s point of view.
  - describe what the system must provide to the actor when the use case is executed.

- Typical contents
  - How the use case starts and ends;
  - Normal flow of events;
  - Alternate flow of events;
  - Exceptional flow of events;

- Documentation style:
  - Choice of how to represent the use case:
    - English language description
    - Collaboration Diagrams
    - Sequence Diagrams

Generalizations

- Actor classes
  - It’s sometimes useful to identify classes of actor
    - E.g., where several actors belong to a single class
    - Some use cases are needed by all members in the class
    - Other use cases are only needed by some members of the class
  - Actors inherit use cases from the class

- Use Case classes
  - Sometimes useful to identify a generalization of several use cases
Modelling Sequences of Events

- Objects “own” information and behaviour:
  - They have attributes and operations relevant to their responsibilities.
  - They don’t “know” about other objects’ information, but can ask for it.
  - To carry out business processes, objects have to collaborate.
    - By sending messages to one another to invoke each other’s operations.
    - Objects can only send messages to one another if they “know” each other
      - I.e. if there is an association between them.

- Describe a Use Case using Sequence Diagrams:
  - Sequence diagrams show step-by-step what’s involved in a use case:
    - Which objects are relevant to the use case.
    - How those objects participate in the function.
  - You may need several sequence diagrams to describe a single use case.
  - Each sequence diagram describes one possible scenario for the use case.
  - Sequence diagrams:
    - Should remain easy to read and understand.
    - Do not include complex control logic.

Example Sequence Diagram

Another Example

Branching messages, etc
Don’t forget what we’re modelling

During analysis

- We want to know about the application domain and the requirements
- So we develop a coarse-grained model to show where responsibilities are, and how objects interact
  - Our models show a message being passed, but we don’t worry too much about the contents of each message
  - To keep things clear, use icons to represent external objects and actors, and boxes to represent system objects.

During design

- We want to say how the software should work
- So we develop fine-grained models to show exactly what will happen when the system runs
  - E.g. show the precise details of each method call.