Chapter 5, Analysis: Dynamic Modeling

Outline
- Dynamic modeling
  - Sequence diagrams
  - State diagrams
- Using dynamic modeling for the design of user interfaces
- Analysis example
- Requirements analysis document template

Example of use case format

Use case name
ReportEmergency

Entry condition
1. The FieldOfficer activates the “Report Emergency” function of her terminal.

Flow of events
2. FRIEND responds by presenting a form to the officer...
3. The FieldOfficer fills the form....
4. The Dispatcher reviews the information submitted by the FieldOfficer ...

Exit condition
5. The FieldOfficer receives the acknowledgment and the selected response.

How do you find classes?
- From previous lectures
  - Application domain analysis: Talk to client to identify abstractions
  - Apply general world knowledge and intuition
  - Scenarios
    - Natural language formulation of a concrete usage of the system
  - Use Cases
    - Natural language formulation of the functions of the system
  - Textual analysis of problem statement (Abbot)
- From this lecture
  - Dynamic model
    - Events: Candidates for operations to be offered by classes
  - Sequence diagrams as sources for objects
- From future lectures
  - Design Patterns

Dynamic Modeling with UML
- Diagrams for dynamic modeling
  - Interaction diagrams describe the dynamic behavior between objects
  - Statecharts describe the dynamic behavior of a single object
- Interaction diagrams
  - Sequence Diagram:
    - Dynamic behavior of a set of objects arranged in time sequence.
    - Good for real-time specifications and complex scenarios
  - Collaboration Diagram:
    - Shows the relationship among objects. Does not show time
- State Charts:
  - A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).
- Activity Diagram:
  - Special type of statechart where all states are action states

Dynamic Modeling
- Definition of dynamic model:
  - A collection of multiple statechart diagrams, one state chart diagram for each class with important dynamic behavior.
- Purpose:
  - Detect and supply methods for the object model
- How do we do this?
  - Start with use case or scenario
  - Model interaction between objects => sequence diagram
  - Model dynamic behavior of single objects => statechart diagram
**Start with Flow of Events from Use Case**

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

**What is an Event?**

- Something that happens at a point in time
- Relation of events to each other:
  - Causally related: Before, after,
  - Causally unrelated: concurrent
- An event sends information from one object to another
- Events can be grouped in event classes with a hierarchical structure. ‘Event’ is often used in two ways:
  - Instance of an event class: “New IETM issued on Thursday September 14 at 9:30 AM”.
  - Event class: “New IETM”, Subclass “Figure Change”
  - Attribute of an event class
    - IETM Update (9:30 AM, 9/14/99)
    - Car starts at (4:45pm, Monroeville Mall, Parking Lot 23a)
    - Mouse button down(button#, tablet-location)

**Sequence Diagram**

- 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 using a DAG notation
- Relation to object identification:
  - Objects/classes have already been identified during object modeling
  - Objects are identified as a result of dynamic modeling
- 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 “Get SeatPosition” 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”**

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

**Heuristics for Sequence Diagrams**

- Layout:
  - 1st column: Should correspond to the actor who initiated the use case
  - 2nd column: Should be a boundary object
  - 3rd column: Should be the control object that manages the rest of the use case
- Creation:
  - Control objects are created at the initiation of a use case
  - Boundary objects are created by control objects
- Access:
  - Entity objects are accessed by control and boundary objects,
  - Entity objects should never call 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 modeler follow the heuristics?

UML Statechart Diagram Notation

- Notation based on work by Harel
  - Added are a few object-oriented modifications
  - A UML statechart diagram can be mapped into a finite state machine

Statechart Diagrams

- Graph whose nodes are states and whose directed arcs are transitions labeled by event names.
- Distinguish between two types of operations:
  - Activity: Operation that takes time to complete
    - associated with states
  - Action: Instantaneous operation
    - associated with events
    - associated with states (reduces drawing complexity): Entry, Exit, Internal Action
- A statechart diagram relates events and states for one class
  - An object model with a set of objects has a set of state diagrams

State

- An abstraction of the attribute of a class
  - State is the aggregation of several attributes a class
- Basically an equivalence class of all those attribute values and links that do not need to be distinguished as far as the control structure of the system is concerned
  - Example: State of a bank
    - A bank is either solvent or insolvent
  - State has duration

Example of a StateChart Diagram

Nested State Diagram

- Activities in states are composite items denoting other lower-level state diagrams
- A lower-level state diagram corresponds to a sequence of lower-level states and events that are invisible in the higher-level diagram.
- Sets of substates in a nested state diagram denoting a superstate are enclosed by a large rounded box, also called contour.
Example of a Nested Statechart Diagram

Superstate

Example of Concurrency within an Object

State Chart Diagram vs Sequence Diagram
Dynamic Modeling of User Interfaces

- Statechart diagrams can be used for the design of user interfaces
- Also called Navigation Path
- States: Name of screens
- Graphical layout of the screens associated with the states helps when presenting the dynamic model of a user interface
- Activities/actions are shown as bullets under screen name
- Often only the exit action is shown
- State transitions: Result of exit action
- Button click
- Menu selection
- Cursor movements
- Good for web-based user interface design

Practical Tips for Dynamic Modeling

- Construct dynamic models only for classes with significant dynamic behavior
- Avoid “analysis paralysis”
- Consider only relevant attributes
- Use abstraction if necessary
- Look at the granularity of the application when deciding on actions and activities
- Reduce notational clutter
- Try to put actions into state boxes (look for identical actions on events leading to the same state)
- Reduce notational clutter
- Try to put actions into state boxes (look for identical actions on events leading to the same state)

Summary: Requirements Analysis

1. What are the transformations?
   - Create scenarios and use case diagrams
     - Talk to client, observe, get historical records, do thought experiments
   - Identify nonfunctional requirements
   - Identify constraints (pseudo requirements)
2. What is the structure of the system?
   - Create 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?
   - Create object diagrams
   - Only for the dynamically interesting objects.
3. What is its control structure?
   - Create sequence diagrams
     - Identify senders and receivers
     - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
   - Create state diagrams
   - Only for the dynamically interesting objects.

Let’s Do Analysis

1. Analyze the problem statement
   - Identify functional requirements
   - Identify nonfunctional requirements
   - Identify constraints (pseudo requirements)
2. Build the functional model:
   - Develop use cases to illustrate functionality requirements
3. Build the dynamic model:
   - Develop sequence diagrams to illustrate the interaction between objects
   - Develop state diagrams for objects with interesting behavior
4. Build the object model:
   - Develop class diagrams showing the structure of the system

Problem Statement: Direction Control for a Toy Car

- Power is turned on
  - Car moves forward and car headlight shines
  - Power is turned off
  - Car stops and headlight goes out
  - Power is turned off
  - Headlight shines
  - Power is turned off
  - Headlight goes out
  - Power is turned on
  - Car runs forward with its headlight shining.

- Power is turned off
  - Car stops and headlight goes out
  - Power is turned on
  - Headlight shines
  - Power is turned off
  - Headlight goes out
  - Power is turned off
  - Car runs forward with its headlight shining.
**Find the Functional Model: Do Use Case Modeling**

- **Use case 1: System Initialization**
  - Entry condition: Power is off, car is not moving
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves forward, headlight is on

- **Use case 2: Turn headlight off**
  - Entry condition: Car moves forward with headlights on
  - Flow of events:
    - Driver turns power off, car stops and headlight goes out.
    - Driver turns power off, headlight shines and car does not move.
  - Exit condition: Car does not move, headlight is out

- **Use case 3: Move car backward**
  - Entry condition: Car is stationary, headlights off
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves backward, headlight is on

- **Use case 4: Stop backward moving car**
  - Entry condition: Car moves backward, headlights on
  - Flow of events:
    - Driver turns power off, car stops, headlight goes out.
    - Power is turned on, headlight shines and car does not move.
    - Power is turned off, headlight goes out.
  - Exit condition: Car does not move, headlight is out.

- **Use case 5: Move car forward**
  - Entry condition: Car does not move, headlight is out
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car runs forward with its headlight shining.

**Use Case Pruning**

- Do we need use case 5?

- **Use case 1: System Initialization**
  - Entry condition: Power is off, car is not moving
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car moves forward, headlight is on

- **Use case 5: Move car forward**
  - Entry condition: Car does not move, headlight is out
  - Flow of events:
    - Driver turns power on
  - Exit condition: Car runs forward with its headlight shining.

**Find the Dynamic Model: Create sequence diagram**

- **Name: Drive Car**
- **Sequence of events:**
  - Billy turns power on
  - Headlight goes on
  - Wheels starts moving forward
  - Wheels keeps moving forward
  - Billy turns power off
  - Headlight goes off
  - Wheels stops moving
  - . . .

**Sequence Diagram for Drive Car Scenario**

**Toy Car: Dynamic Model**

- Headlight
- Wheel
- Off
- On
- Power on
- Power off
- Forward
- Stationary
- Backward
**Toy Car: Object Model**

- **Car**
  - **Power**
    - Status: (On, Off)
  - **Headlight**
    - Status: (On, Off)
  - **Wheel**
    - Motion: (Forward, Stationary, Backward)
    - Start_Moving(), Stop_Moving()
Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
   3.1 Overview
   3.2 Functional requirements
   3.3 Nonfunctional requirements
   3.4 Constraints ("Pseudo requirements")
3.5 System models
   3.5.1 Scenarios
   3.5.2 Use case model
   3.5.3 Object model
   3.5.3.1 Data dictionary
   3.5.3.2 Class diagrams
   3.5.4 Dynamic models
   3.5.5 User interface
4. Glossary

Section 3.5 System Model

3.5.1 Scenarios
   - As-is scenarios, visionary scenarios
3.5.2 Use case model
   - Actors and use cases
3.5.3 Object model
   - Data dictionary
   - Class diagrams (classes, associations, attributes and operations)
3.5.4 Dynamic model
   - State diagrams for classes with significant dynamic behavior
   - Sequence diagrams for collaborating objects (protocol)
3.5.5 User Interface
   - Navigational Paths, Screen mockups

Section 3.3 Nonfunctional Requirements

3.3.1 User interface and human factors
3.3.2 Documentation
3.3.3 Hardware considerations
3.3.4 Performance characteristics
3.3.5 Error handling and extreme conditions
3.3.6 System interfacing
3.3.7 Quality issues
3.3.8 System modifications
3.3.9 Physical environment
3.3.10 Security issues
3.3.11 Resources and management issues

Nonfunctional Requirements (continued)

3.3.4 Performance characteristics
   - Are there any speed, throughput, or response time constraints on the system?
   - Are there size or capacity constraints on the data to be processed by the system?
3.3.5 Error handling and extreme conditions
   - How should the system respond to input errors?
   - How should the system respond to extreme conditions?
3.3.6 System interfacing
   - Is input coming from systems outside the proposed system?
   - Is output going to systems outside the proposed system?
   - Are there restrictions on the format or medium that must be used for input or output?

Nonfunctional Requirements, ctd

- 3.3.7 Quality issues
   - What are the requirements for reliability?
   - Must the system trap faults?
   - Is there a maximum acceptable time for restarting the system after a failure?
   - What is the acceptable system downtime per 24-hour period?
   - Is it important that the system be portable (able to move to different hardware or operating system environments)?
3.3.8 System Modifications
   - What parts of the system are likely candidates for later modification?
   - What sorts of modifications are expected?
3.3.9 Physical Environment
   - Where will the target equipment operate?
   - Will the target equipment be in one or several locations?
   - Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, ...)?
Nonfunctional Requirements, ctd

- 3.3.10 Security Issues
  - Must access to any data or the system itself be controlled?
  - Is physical security an issue?
- 3.3.11 Resources and Management Issues
  - How often will the system be backed up?
  - Who will be responsible for the back up?
  - Who is responsible for system installation?
  - Who will be responsible for system maintenance?

Pseudo Requirements (Constraints)

- Pseudo requirement:
  - Any client restriction on the solution domain
- Examples:
  - The target platform must be an IBM/360
  - The implementation language must be COBOL
  - The documentation standard X must be used
  - A dataglove must be used
  - ActiveX must be used
  - The system must interface to a papertape reader

Project Agreement

- The project agreement represents the acceptance of the analysis model (as documented by the requirements analysis document) by the client.
- The client and the developers converge on a single idea and agree about the functions and features that the system will have. In addition, they agree on:
  - a list of priorities
  - a revision process
  - a list of criteria that will be used to accept or reject the system
  - a schedule, and a budget

Prioritizing requirements

- High priority ("Core requirements")
  - Must be addressed during analysis, design, and implementation.
  - A high-priority feature must be demonstrated successfully during client acceptance.
- Medium priority ("Optional requirements")
  - Must be addressed during analysis and design.
  - Usually implemented and demonstrated in the second iteration of the system development.
- Low priority ("Fancy requirements")
  - Must be addressed during analysis ("very visionary scenarios").
  - Illustrates how the system is going to be used in the future if not yet available technology enablers are

Summary

In this lecture, we reviewed the construction of the dynamic model from use case and object models. In particular, we described: In particular, we described:
- Sequence diagrams for identifying missing objects and operations.
- Statechart diagrams for identifying missing attributes.
- Definition of an event hierarchy.

In addition, we described the requirements analysis document and its use when interacting with the client.