Object-Oriented Software Engineering with UML

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 6 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Object-Oriented Analysis and Design

- Object-Oriented Analysis and Design Method combines:
  - Object Modeling
    - J. Rumbaugh et al, "Object-Oriented Modeling and Design", Prentice Hall, 1991
  - Use cases
    - I Jacobson et al, "Object-Oriented Software Engineering", Addison Wesley, Reading MA, 1992.
  - Statecharts (Harel)
  - Sequence Diagrams (Rumbaugh, Jacobson)
  - Object Collaboration Diagrams (Booch)
  - Concurrent, Distributed, & Real-Time Design (Gomaa)
- Unified Modeling Language (UML)
  - Standardized notation for object-oriented development

Object Modeling Technique (OMT)

- Addresses
  - Structural (static) aspects of problem
    - Object Model
      - Information model
  - Dynamic aspects of problem
    - Dynamic Model
      - Uses state transition diagrams and scenarios
  - Functional aspects of problem
    - Functional model
      - Data flow diagrams

Object-Oriented Software Engineering

- Based around *use case* (scenario) concept
- OOSE supports five models:
- Requirements model
  - Defines functional requirements in terms of use cases
- Analysis model
  - Defines objects and how they participate in use cases
- Design model
  - Maps object structure to operational environment
- Implementation model
  - Source code of system
- Test model
  - Testing of system

Unified Modeling Language (UML)

- Standardized notation for object-oriented development
  - Combines notations of OMT, Booch, and use cases
- Needs to be used with an analysis and design method
  - Notation provides more support for analysis than design
- Intended for all types of OO software development
- UML notation used for OO analysis and design method for concurrent, real-time and distributed applications
  - Concurrent Object Modeling and architectural design mEThod (COMET)
- H. Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000.

Unified Modeling Language (UML) Diagrams

Reference: Gomaa text, Chapter 2

- Use Case Diagrams
  - Fig. 2.1
- Class Diagrams
  - Figs. 2.3-2.4
- Collaboration Diagrams
  - Fig. 2.5
- Sequence Diagrams
  - Fig. 2.6
- Statecharts
  - Figs. 2.7-2.8
- Deployment diagrams
  - Fig. 2.13

Object-Oriented Software Life Cycle

Requirements & Analysis Modeling

- Requirements Modeling (Fig. 6.1)
  - Use Case Modeling
    - Define software functional requirements in terms of use cases and actors
- Analysis Modeling (Fig. 6.1)
  - Static Modeling
    - Define structural relationships between classes
    - Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - Define statecharts for state dependent objects
    - Defines how objects participate in use cases using collaboration diagrams or sequence diagrams

Object-Oriented Software Life Cycle

Architectural Design (Fig. 6.1)

- Maps analysis model (emphasis on problem domain) to design model (emphasis on solution domain)
- Structure system into subsystems
- Design each subsystem
- Sequential Applications
  - Emphasis on OO concepts
    - Information hiding, classes, inheritance
- Concurrent, Distributed and Real-Time Applications
  - Emphasis on
    - OO concepts
    - Concurrent tasking

Object-Oriented Software Life Cycle
Incremental Development

- Complete architectural design of software
- Incremental Software Construction (Fig. 6.1)
  - Select subset of system based on use cases
    - Detailed design, code, unit test of components in subset
- Incremental Software Integration (Fig. 6.1)
  - Integration testing of each system increment
    - Integration test based on use cases
  - Develop integration test cases for each use case
  - White box testing
    - Test interfaces between components in use case

Object-Oriented Software Life Cycle
System Testing

- System Testing (Fig. 6.1)
  - Includes functional testing of system
    - Testing of functional requirements
  - Black box testing
    - Based on use cases
- Need system test for each increment released to user
- Independent test team
  - Goal is to break system
  - Thorough systematic test of system before release to users

Steps in Using COMET/UML

1 Develop Object-Oriented Requirements Model
  - Develop Use Case Model (Chapter 7)

2 Develop Object-Oriented Analysis Model
  - Develop static model of problem domain (Chapter 8)
  - Structure system into objects (Chapter 9)
  - Develop statecharts for state dependent objects (Chapter 10)
  - Develop object interaction diagrams for each use case (Chapter 11)

3 Develop Object-Oriented Design Model
  - Design Overall Software Architecture (Chapter 12)
  - Design Distributed Component-based Subsystems (Chapter 13)
  - Structure Subsystems into Concurrent Tasks (Chapter 14)
  - Design Information Hiding Classes (Chapter 15)
  - Develop Detailed Software Design (Chapter 16)

COMET OO Analysis and Design

- UML Notation
  - Supports both Analysis and Design concepts
- COMET/UML method
  - Separate requirements activities, analysis activities and design activities
- Requirements Modeling
  - Consider system as black box
  - Develop Use Case Model

COMET OO Analysis and Design

- Analysis modeling
  - Consider analysis of problem domain
  - Determine problem oriented objects and classes
  - Analyze static viewpoint in Static Model
    - Classes, relationships, attributes
  - Analyze dynamic viewpoint in Dynamic Model
    - Statecharts
    - Object interaction model
      - Consider objects supporting each use case
      - Analyze sequence of interactions between objects
      - Analyze information passed between objects

COMET OO Analysis and Design

- Design Model
  - Consider solution domain
  - Make decisions about overall software architecture
  - Make decisions about distributed component-based subsystems
  - Make decisions about characteristics of objects
    - Active or Passive
  - Make decisions about characteristics of messages
    - Asynchronous or Synchronous (with/without reply)
  - Make decisions about class interfaces
    - Operations and parameters
  - Make detailed design decisions

Use Case Modeling

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 7 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Use Case Modeling

- Use Case
  - Describes sequence of interactions between user (actor) and system
- **Figure 2.1 UML notation for use case diagram**
- Initially developed in Use Case model
  - Shows interaction between actor and black box system
- Use cases refined in Dynamic Model
  - Show objects participating in use case
  - Develop collaboration diagrams or sequence diagrams
- Use cases refined further in Design Model
- Use cases form basis of integration & system test cases

Actors

- Actor models external entities of system
- Actors interact directly with system
  - Human user (**Figure 7.1)**
  - External I/O device (**Figure 7.2)**
  - Timer (**Figure 7.3)**
  - External system (**Figure 7.4)**
- Actor initiates actions by system
  - May use I/O devices or external system to physically interact with system
  - Actor initiates use cases

Use Cases

- Define system functional requirements in terms of Actors and Use cases
  - Narrative description
- Identifying use cases
  - Consider requirements of each actor who interacts with system
  - Use case is a complete sequence of events initiated by an actor
    - Use case starts with input from an actor
    - Describes interactions between actor and system
    - Provides value to actor
  - Basic path
    - Most common sequence
  - Alternative branches
    - Variants of basic path
      - E.g., for error handling
- **Figure 7.5 Banking system actor & use cases**

Documenting Use Cases

- Name
- Summary
  - Short description of use case
- Dependency (on other use cases)
- Actors
- Preconditions
  - Conditions that are true at start of use case
- Description
  - Narrative description of basic path
- Alternatives
  - Narrative description of alternative paths
- Postcondition
  - Condition that is true at end of use case

Use Case Relationships

- **Include** relationship
  - Identify common patterns (sequences) in several use cases
  - Extract common pattern into **abstract use case**
  - Concrete use cases **include** abstract use case
  - **Figure 7.7 Example of abstract use case and include relationship**
- **Extend** relationship
  - Use case A is an extension of use case B
  - Under certain conditions use case B will be extended by description given in use case A
  - Same use case can be extended in different ways
  - **Figure 7.6 Example of extend relationship**

Use Case Package

- Use Case Package
  - Encompasses related group of use cases
  - Represent high level requirements
- Use Case Package Structuring
  - Group use cases into packages based on
    - Major subset of system functionality
    - Related use cases started by the same actor
- **Figure 7.8 Example of use case package**

Static Modeling

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 8 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Object-Oriented Software Life Cycle

Requirements & Analysis Modeling

- Requirements Modeling
  - Use Case Modeling
    - Define software functional requirements in terms of use cases and actors
- Analysis Modeling
  - Static Modeling
    - Define structural relationships between classes
    - Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - Define statecharts for state dependent objects
    - Defines how objects participate in use cases using collaboration diagrams or sequence diagrams

Modeling Objects in the Problem Domain

- Analyze problem domain
  - Map real-world objects to software objects
  - Objects less likely to change than functions of system
- Object structuring criteria
  - Guidelines for structuring objects
- Structural view of objects
  - Static (Object) Model
- Dynamic view of objects
  - Dynamic model
    - Statecharts
    - Sequence Diagrams or Object Collaboration Diagrams

Static Modeling

- Define structural relationships between classes
  - Depict classes and their relationships on class diagrams
- Relationships between classes
  - Associations
  - Composition / Aggregation
  - Generalization / Specialization
- Static Modeling during Analysis
  - System Context Class Diagram
    - Depict external classes and system boundary
- Static Modeling of Entity classes
  - Persistent classes that store data

Static Modeling

- Class
  - Real world entity type about which information is stored
  - Represents a collection of identical objects (instances)
  - Described by means of attributes (data items)
  - Has operations to access data maintained by objects
  - Each object instance can be uniquely identified
- Relationships between classes
  - Associations
  - Composition / Aggregation
  - Generalization / Specialization
- **Figure 2.3 UML notation for relationships on class diagram**

Associations

- Association is
  - static, structural relationship between classes
  - E.g, Employee works in Department
- Multiplicity of Associations
  - Specifies how many instances of one class may relate to a single instance of another class
- 1-to-1 association (**Figure 8.1)**
  - Company has President
- 1-to-many association (**Figure 8.2)**
  - Bank manages Account
- Optional association (0, 1, or many) (**Figure 8.5)**
  - Customer owns Credit Card
- Many-to-Many association (**Figure 8.6)**
  - Course has Student
  - Student attends Course

Composition and Aggregation Hierarchies

- Whole/Part Relationships
  - Show components of more complex class
  - Composition is stronger relationship than aggregation
- Composition Hierarchy
  - Whole and part objects are created, live, die together
  - Often also has a physical association
  - Association between instances
  - **Figure 8.10 Example of composition hierarchy**
- Aggregation Hierarchy
  - Part objects of aggregate object may be created and deleted independently of aggregate object
  - Often used for more abstract whole/part relationships than composite objects
  - **Figure 8.11 Example of aggregation hierarchy**

Generalization / Specialization Hierarchy

- Some classes are similar but not identical
  - Have some attributes in common, others different
- Common attributes abstracted into generalized class (superclass)
  - E.g., Account (Account number, Balance)
- Different attributes are properties of specialized class (subclass)
  - E.g., Savings Account (Interest)
- IS A relationship between subclass and superclass
  - Savings Account IS A Account
  - **Figure 8.12 Generalization / specialization hierarchy**

Static Modeling of Problem Domain

- During Analysis Modeling
  - Conceptual static model
  - Emphasizes real-world classes in the problem domain
  - Does not initially address software classes
  - Emphasis on
    - Physical classes
      - Have physical characteristics (can see, touch)
    - Entity classes
      - Data intensive classes
- **Figure 8.16 Conceptual static model for Banking System**

System Context Class Diagram

- Defines boundary between system and external environment
  - May be depicted on System Context Class Diagram
- External classes
  - External entities that system interfaces to
- Categories of external classes
  - External I/O device
  - External user
  - External system
  - External timer
- **Figure 8.17 Banking System class context diagram**

Static Modeling of Entity Classes

- Entity classes
  - Data intensive classes
  - Store long-living (persistent) data
  - Especially important for Information Systems
    - Many are database intensive
  - Also important for many real-time and distributed applications
- During analysis modeling
  - Model entity classes in the problem domain
  - Attributes
  - Relationships
  - **Figure 8.18 Conceptual static model for Banking System - entity classes**

Object Structuring

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 9 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Object-Oriented Software Life Cycle

Requirements & Analysis Modeling

- Requirements Modeling
  - Use Case Modeling
    - Define software functional requirements in terms of use cases and actors
- Analysis Modeling
  - Static Modeling
    - Define structural relationships between classes
    - Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - Define statecharts for state dependent objects
    - Defines how objects participate in use cases using collaboration diagrams or sequence diagrams

Object Structuring

- Decomposition of problem into objects
  - Based on judgement and characteristics of problem
  - No single correct representation
- Whether objects are in same class or different class depends on nature of problem
- In auto catalog
  - cars, vans, trucks may all be objects in same class
- For vehicle manufacturer
  - cars, vans, trucks may all be objects of different classes

Object Structuring Criteria

- Determine all software objects in system
  - Use Object Structuring Criteria
  - Guidelines for identifying objects
- Structuring criteria depicted using stereotypes (Fig. 9.1)
  - **Stereotype** defines a new building block that is derived from an existing UML modeling element but is tailored to the modeler's problem
  - Depicted using guillemets
    - «entity», «interface», «control»
- Objects are categorized
  - A **category** is a specifically defined division in a system of classification

Object Structuring Criteria

- Interface objects
  - Interface to external environment
  - Each software interface object interfaces to an external (real-world) object (Fig. 9.2)
  - Device interface object
    - Interfaces to I/O device
- Input device interface object
  - E.g., Sensor Interface (Fig. 9.3)
- Output device interface object
  - E.g., Actuator Interface (Fig. 9.4)
- I/O (Input/Output) device interface object
  - E.g., ATM Card Reader Interface (Fig. 9.5)

Object Structuring Criteria

- User interface object (Fig. 9.6)
  - Interfaces to a human user
    - Via standard I/O devices
      - keyboard, visual display, mouse
    - Support simple or complex user interfaces
      - Command line interface
      - Graphical user interface (GUI)
    - Graphical user interface (GUI) object
      - Often a composite object
      - Composed of simpler objects

Object Structuring Criteria

- System interface object
  - Interfaces to an external system
  - Hides details of **how** to communicate with external system
    - E.g., Robot Interface
    - Interfaces to external (real-world) robot
    - Example: Fig. 9.7

Depicting External Classes and Interface Classes

- Start from system context class diagram
  - Shows external classes
  - System (aggregate class)
- Each **external class** must interface to
  - software **interface class**
- Use UML package notation
  - System shown as package
  - External classes are outside the system package
  - Interface classes are inside the system package
  - Example: Fig. 9.8

Object Structuring Criteria

- Entity objects
  - Long lasting objects that store information
    - Same object typically accessed by many use cases
    - Information persists over access by several use cases
      - E.g., Account, Customer
  - Entity classes and relationships shown on static model
  - Entity classes often mapped to relational database during design
  - Examples: Figs. 9.9 – 9.10

Object Structuring Criteria

- Control objects
  - Provides overall coordination for execution of use case
  - Glue that unites other objects that participate in use case
  - Makes overall decisions
  - Decides when, and in what order, other objects participate in use case.
    - Entity objects
    - Interface objects
  - Simple use cases do not need control objects
  - More complex use case usually has at least one control object

Object Structuring Criteria

- Control object
  - Coordinator object
    - Provides sequencing for use case
    - Is not state dependent
    - Example: Fig. 9.11
  - State dependent control object
    - Defined by finite state machine
      - statechart or state transition table
    - Example: Fig. 9.12
  - Timer object
    - Activated periodically
    - Example: Fig. 9.13

Object Structuring Criteria

- Application Logic Objects
  - Business Logic Object
    - Defines business specific application logic (rules) for processing a client request
    - Usually accesses more than one entity object
    - Example: Fig. 9.11
  - Algorithm Object
    - Encapsulates algorithm used in problem domain
    - More usual in scientific, engineering, real-time domains
    - Example: Fig. 9.14

Subsystems

- Subsystem
  - Composite or aggregate object
  - Subsystem may be depicted as package in UML
  - Subsystem dependencies may be depicted
  - For more advanced relationship depict subsystem as aggregate or composite class
  - Examples: Figs. 9.15-9.16
- Object Structuring
  - Only easily identified subsystems are determined
  - Subsystem structuring is addressed in more detail later

Finite State Machines and Statecharts

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 10 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Object-Oriented Software Life Cycle

Requirements & Analysis Modeling

- Requirements Modeling
  - Use Case Modeling
    - – Define software functional requirements in terms of use cases and actors
- Analysis Modeling
  - Static Modeling
    - – Define structural relationships between classes
    - – Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - – Define statecharts for state dependent objects
    - – Defines how objects participate in use cases using collaboration diagrams or sequence diagrams

Finite State Machines

Finite number of states

Only in one state at a time

Transition

Change of state

Caused by event

Transition to same or different state

Action may result from state transition

Notation

State transition diagram
State transition table
Statechart
Examples of statecharts (Figures 10.1 - 10.3)

Finite State Machines and Statecharts

- Statechart
  - Graphical representation of finite state machine
  - States are rounded boxes
  - Transitions are arcs
- Statechart relates events and states
- Event
  - Causes change of state
    - Referred to as state transition
- State
  - A recognizable situation
  - Exists over an interval of time
  - Represents an interval between successive events
- Examples of statecharts (Figures 10.1 - 10.3)

Events

- Event
  - A discrete signal that happens at a point in time
  - Also known as a stimulus
  - Has no duration
- Two events
  - May logically depend on each other
  - E.g, ATM Card inserted before Pin # entered
- Two events
  - May be independent of each other
  - E.g., ATM card read at Alexandria ATM
  - ATM Card read at Fairfax ATM

Events and Conditions

- State transition label
  - Event [Condition]
- Condition is a Boolean function
  - Conditions are optional on statecharts
  - Condition is true for finite period of time
- When event occurs, condition must be *true* for state transition to occur.
- If condition is *false*, state transition does not occur
- Condition may be used to indicate that event has occurred
  - E.g., Closedown Was Requested
- **Figure 10.4 Partial statechart**
- **Figure 10.5 Relationship between events and conditions**
- **Figure 10.6 Use of events and conditions in statechart**
- **Figure 10.7 Example of events and conditions**

Actions

- State transition label
  - Event / action(s)
  - Event [condition] / action(s)
- Action
  - Executed as a result of state transition
  - Executes instantaneously at state transition
  - Terminates itself
  - Is optional
- **Figure 10.8 Example of actions**
- **Figure 10.9 Detailed Cruise Control statechart with actions and conditions**
- Activity
- Entry/exit actions

Activities

- Activity
  - Executes for duration of state
    - Enable Activity on entry to state
    - Disable Activity on exit from state
  - Alternatively
    - do / Activity in state
- Examples of activities
  - Increase Speed
    - Executes for duration of Accelerating state
  - Maintain Speed
    - Executes for duration of Cruising state
  - Resume Cruising
    - Executes for duration of Resuming state
- **Figure 10.10 Cruise Control statechart with activities**

Entry and Exit Actions

- Entry action
  - Action executed on entry into state
    - Entry / action
  - E.g., Display System Down
  - E.g., Display Welcome
  - **Figure 10.11 Example of entry actions**
- Exit action
  - Action executed on exit from state
    - Exit / action
  - E.g, Select Desired Speed
  - **Figure 10.12 Example of exit action**

Hierarchical Statecharts

- Disadvantages of State Transition Diagrams and Flat Statecharts
  - Complex State Transition Diagrams get very cluttered
  - Limited capability for managing complexity
- Hierarchical Statecharts
  - Based on Harel Statecharts
  - Notation for hierarchical decomposition of state transition diagrams
    - Superstate decomposed into substates
    - Default entry states
    - Transition out of superstate corresponds to transition out of every substate

Hierarchical Statecharts

- **OR** decomposition
  - When object is in superstate
    - It is in one and only one of substates
  - Transition into superstate
    - Must be to one and only one of substates
- Aggregation of state transitions
  - If same event causes transition out of every substate
  - Then aggregate into transition out of superstate
- Examples: Fig. 10.14, 19.20-19.23

Hierarchical Statecharts

- Concurrent statecharts
  - State of an object described by more than one statechart
  - Show different aspects of object, may not be concurrent
- Orthogonal statechart
  - Used to depict states of different aspects of object
- **AND** decomposition
  - Object is in one substate on each lower level statechart
  - Object's state is union of all substates
- Same event
  - May cause transitions on more than one statechart
- Output event on one statechart
  - May be input event on other statechart
- Substate on one statechart
  - May be condition on other statechart
- Example: Fig. 10.15

Guidelines on Statecharts

- State name must be passive not active
  - Represents time period when something
    - is happenING, e.g., Elevator Moving
    - Identifiable situation, e.g., Elevator Idle, Initial
- State names must be unique
- Must be able to exit from every state
- Flat statechart
  - Statechart is only in one state at a time
- Hierarchical statechart
  - **or** decomposition
    - Statechart is only in one substate at a time
  - **and** decomposition
    - Statechart is in one substate on each lower level concurrent statechart

Guidelines on Statecharts

- Event is the cause of the state transition
  - Event happens at a moment in time
  - Event name indicates something has just happened
    - e.g, Up Request, Door Closed
- Action is the result of the state transition
  - Action is a command, e.g., Stop, Close Door
  - Action executes instantaneously
  - Activity executes throughout a given state
- More than one action possible with a state transition
  - No sequential dependency between actions
- Condition is a Boolean value
  - Event [Condition]
  - State transition only occurs if
    - Event happens & Condition is True
  - Condition is True over some interval of time
- Actions, Activities and Conditions are optional

Developing Statechart from Use Case

- Develop state dependent use case
- Start with scenario (one path through use case)
  - Consider sequence of interactions between actor and system
- Consider sequence of external events
  - Input event from external environment
  - Causes state transition to new state
  - Action may result from state transition
  - Activity may be enabled / disabled
- Initially develop flat statechart

Example of Developing Statechart from Use Case

- Cruise Control System
- Control Speed use case
  - Scenario of external events
  - Initial state: INITIAL
    a) Driver engages cruise control lever in ACCEL position
    b) Driver releases lever (CRUISE)
    c) Driver presses brake
    d) Driver engages lever in RESUME position
- Example: Fig. 10.16, 10.17

Example of Developing Statechart from Use Case

Control Speed Use Case
Actor: Driver

Summary: This use case describes the automated cruise control of the car, given the driver inputs via the cruise control lever, brake, and engine external input devices.

Precondition: Driver has switched on the engine and is operating the car manually.

Description:

This use case is described in terms of a typical scenario consisting of the following sequence of external events:

1. Driver moves the cruise control lever to the ACCEL position and holds the lever in this position. The system initiates automated acceleration so that the car automatically accelerates.
2. Driver releases the cruise control lever in order to cruise at a constant speed. The system stops automatic acceleration and starts maintaining the speed of the car at the cruising speed. The cruising speed is stored for future reference.
3. Driver presses the brake to disable cruise control. The system disables cruise control so that the car is once more under manual operation.
4. Driver moves the cruise control lever to the RESUME position in order to resume cruising. The system initiates acceleration (or deceleration) toward the previously stored cruising speed.
5. When the system detects that the cruising speed has been reached, it stops automatic acceleration (or deceleration) and starts maintaining the speed of the car at the cruising speed.
6. Driver moves the cruise control lever to the OFF position. The system disables cruise control, so that the car is once more under manual operation.
7. The driver stops the car and switches off the engine.

Developing Statechart from Use Case
(continued)

- Consider alternative external events
  - Could result in additional states
  - Could result in additional state transitions
  - Example: Fig. 10.10
- Develop hierarchical statechart
  - States that can be aggregated to form superstate
  - Event causing transition from several states
    - Create superstate with one transition out of superstate
    - Instead of many transitions out of substates
  - Example: Fig. 10.18-10.19
- Develop orthogonal statechart
  - Model different aspects of state dependent object
  - Example: Fig. 10.20

Example of Developing Statechart from Use Case

(Control Speed use case - continued)

Alternatives:

The driver actor interacts with the system, using three external input devices: the cruise control lever, the brake, and the engine. Following are the complete set of input events initiated by the driver actor using these external devices, and the reaction of the system to them:

- The Accel, Cruise, Resume, and Off external events from the cruise control lever. The Accel event causes automated acceleration, providing the brake is not pressed. The Cruise event may only follow an Accel event. The Resume event may only occur after cruising has been disabled and the desired cruising speed has been stored. The Off event always disables cruise control.
- The Brake Pressed and Brake Released external events from the brake. The Brake Pressed event always disables cruise control. Automated vehicle control is not possible as long as the brake is pressed. After the brake is released, automated vehicle control may be enabled.
- The Engine On and Engine Off external events from the engine. The Engine Off event disables any activity in the system.

Postcondition: The car is stationary, with the engine switched off.

Dynamic Modeling

Hassan Gomaa
Dept of Information & Software Engineering
George Mason University

Reference: H. Gomaa, "Chapter 11 - Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley Object Technology Series, July, 2000

Object-Oriented Software Life Cycle

Requirements & Analysis Modeling

- Requirements Modeling
  - Use Case Modeling
    - Define software functional requirements in terms of use cases and actors
- Analysis Modeling
  - Static Modeling
    - Define structural relationships between classes
    - Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - Define statecharts for state dependent objects
    - Defines how objects participate in use cases using collaboration diagrams or sequence diagrams

Dynamic Modeling

- Use cases refined in Dynamic Model
  - Show objects participating in each use case
- Determine how objects participate in use case
  - Shows sequence of object interactions in use case
  - Develop collaboration diagram
  - Develop sequence diagram
- Message sequence description
  - Narrative description of sequence of object interactions
- State dependent objects
  - Modeled using statecharts
- Dynamic Analysis
  - Approach to determine how objects interact with each other to support use case

Collaboration Diagrams

- Graphically depicts objects participating in a use case
  - Show objects as boxes
  - Show their message interactions as arrows
  - Number sequence of messages
- Message
  - Message = Event + Attributes
    - E.g, ATM card inserted (Card id, expiration date)
- Collaboration Diagram developed for each use case
  - Some objects only appear on one Collaboration Diagram
  - Some objects appear on several Collaboration Diagrams
- Example: Fig. 11.1

Sequence Diagram

- Shows sequence of object interactions in use case
- Emphasis on messages passed between objects
  - Objects represented by vertical lines
    - Actor is on extreme left of page
  - Messages represented by labeled horizontal arrows
    - Only source and destination of arrow are relevant
    - Message is sent from sending object to receiving object
  - Time increases from top of page to bottom
  - Spacing between messages is not relevant
  - Message sequence numbering is optional
- Example: Fig. 11.2

Message Sequence Numbering

- Aa1.1a
  - [first optional letter sequence][numeric sequence] [second optional letter sequence]
- First optional letter sequence - use case id
- Numeric sequence
  - Message sequence starting with external event
  - A1, A2, A3
- Dewey Classification System
  - A1, A1.1, A1.1.1, A1.2
- Interactive System
  - Whole number for external event
    - A1
  - Decimal number for subsequent internal events
    - A1.1, A1.2
- Second optional letter sequence
  - Concurrent event sequences
    - A3, A3a

Message Sequence Description

- Describes how objects participate in use case
  - Narrative description
  - Corresponds to Collaboration Diagram & Sequence Diagram
- Description corresponds to message sequence numbering on diagrams
  - Describe what object does on receiving message
  - E.g, every time the use case references an entity object
    - Describe the object being accessed
    - Identify attributes referenced
- Example: Fig. 11.3

Dynamic Analysis

- Determine how objects interact with each other to support use case
  - Start with external event from actor
  - Determine objects needed to support use case
  - Determine sequence of internal events following external event
  - Depict on collaboration diagram
- Non-state dependent Dynamic Analysis
- State dependent Dynamic Analysis
  - Controlled by statechart
  - Executed by control object
  - Control object activates/deactivates other objects

Non-State Dependent Dynamic Analysis

- Start with black box use case
- Determine interface objects
  - Receives external events from actor
- Determine internal objects
  - Receive messages from interface objects
- Determine object collaboration
  - Sequence of messages passed
- Develop alternative branches
  - E.g, for error handling or less frequently occurring conditions
- Example: Fig. 11.4

State Dependent Dynamic Analysis

- For each black box use case
  - Determine objects participating in use case
    - Determine interface objects
    - Determine internal objects
  - Determine object collaboration
  - Develop statechart(s)
    - Control object executes statechart
  - Iterate till object collaboration diagram consistent with statechart

State Dependent Dynamic Analysis

- For each event that arrives at control object
  - Determine state transition from current state to next state
- For each state transition
  - Determine actions that result from change in state
  - Determine activities to be executed in new state
  - Determine objects required to perform actions and activities
- Develop alternative branches
- Complete analysis use case
  - Must enter every state
  - Must execute every state transition
  - Perform each action and activity

Example of Dynamic Analysis
Banking System - Validate PIN Use Case

- Figs. 11.3, 11.5 – 11.11 , 19.12- 19.14
- Client Objects
  - Interface Objects
    - Card Reader Interface
    - Customer Interface
  - Entity Objects
    - ATM Card
    - ATM Transaction
  - State Dependent Control Object
    - ATM Control

Example of Dynamic Analysis
Banking System - Validate PIN Use Case

- Server Objects
  - Entity Objects
    - Debit Card
    - Card Account
  - Business Logic Objects
    - PIN Validation Transaction Manager
- Example: Fig. 19.15

Example of Dynamic Analysis
Banking System - Withdraw Funds Use Case

- Client Objects
  - Interface Objects
    - Card Reader Interface
    - Receipt Printer Interface
    - Cash Dispenser Interface
    - Customer Interface
  - Entity Objects
    - ATM Transaction
    - ATM Cash
  - State Dependent Control Object
    - ATM Control
- Figs. 19.16- 19.18

Example of Dynamic Analysis
Banking System - Withdraw Funds Use Case

- Server objects
  - Entity Objects
    - Checking Account
    - *or* Savings Account
    - Debit Card
    - Transaction Log
  - Business Logic Objects
    - Withdrawal Transaction Manager
- Fig. 19.19

Example of Dynamic Analysis
Cruise Control System - Control Speed Use Case

- Example from Cruise Control System
- Initial Object Determination
  - State Dependent Control object
    - Cruise Control
  - Input device interface objects
    - Cruise Control Lever Interface, Engine Interface, Brake Interface
  - Output device interface object
    - Throttle Interface

Example of Dynamic Analysis
Cruise Control System - Control Speed Use Case

- Scenario of external events
  - Initial state: INITIAL
    a) Driver engages cruise control lever in ACCEL position
    b) Driver releases lever (CRUISE)
    c) Driver presses brake
    d) Driver engages lever in RESUME position
- Example: Fig. 11.12 – 11.22

Object-Oriented Software Life Cycle
Requirements & Analysis Modeling

- Requirements Modeling
  - Use Case Modeling
    - Define software functional requirements in terms of use cases and actors
- Analysis Modeling
  - Static Modeling
    - Define structural relationships between classes
    - Depict classes and their relationships on class diagrams
  - Dynamic Modeling
    - Define statecharts for state dependent objects
    - Defines how objects participate in use cases using collaboration diagrams or sequence diagrams