**OO Using UML: Dynamic Models**

Defining how the objects behave

---

**Overview**

- The object model describes the structure of the system (objects, attributes, and operations)
- The dynamic model describes how the objects change state (how the attributes change) and in which order the state changes can take place
- Several models used to find the appropriate dynamic behavior
  - Interaction diagrams
  - Activity diagrams
  - State diagrams
- Uses finite state machines and expresses the changes in terms of events and states

---

**We Will Cover**

- Why interaction diagrams?
- Sequence diagrams
  - Capturing use-cases
  - Dealing with concurrency
- Collaboration diagrams
- When to use what
- When to use interaction diagrams

---

**Interaction Diagrams**

---

**Different Types of Interaction Diagrams**

- An Interaction Diagram typically captures a use-case
  - A sequence of user interactions
- **Sequence diagrams**
  - Highlight the sequencing of the interactions between objects
- Collaboration diagrams
  - Highlight the structure of the components (objects) involved in the interaction

---

**Home Heating Use-Case**

**Use case:** Power Up  
**Actors:** Home Owner (initiator)  
**Type:** Primary and essential  
**Description:** The Home Owner turns the power on. Each room is temperature checked. If a room is below the desired temperature the valve for the room is opened, the water pump started, the fuel valve opened, and the burner ignited. If the temperature in all rooms is above the desired temperature, no actions are taken.  
**Cross Ref.:** Requirements XX, YY, and ZZ  
**Use-Cases:** None
Sequence Diagrams

Example from Fowler

Concurrency

Another Example

Comment the Diagram

Collaboration Diagrams
Conditional Behavior

- Something you will encounter trying to capture complex use-cases
  - The user does something. If this something is X do this... If this something is Y do something else... If this something is Z...
  - Split the diagram into several
  - Split the use-case also
  - Use the conditional message
    - Could become messy
  - Remember, clarity is the goal!

Comparison

- Both diagrams capture the same information
  - People just have different preferences
  - We prefer sequence diagrams
    - They clearly highlight the order of things
    - Invaluable when reasoning about multi-tasking
  - Others like collaboration diagrams
    - Shows the static structure
      - Very useful when organizing classes into packages
  - We get the structure from the Class Diagrams

When to Use Interaction Diagrams

- When you want to clarify and explore single use-cases involving several objects
  - Quickly becomes unruly if you do not watch it
- If you are interested in one object over many use-cases -- state transition diagrams
- If you are interested in many objects over many use cases -- activity diagrams

State Diagrams

We Will Cover

- State Machines
  - An alternate way of capturing scenarios
    - Large classes of scenarios
- Syntax and Semantics
- When to use state machines

Where Do State Diagrams Fit?

- Generally, one state diagram per class
- Describe the entire behavior of class
- All methods in one state diagram
### Events, Conditions, and States

- **Event**: something that happens at a point in time
  - Operator presses self-test button
  - The alarm goes off
- **Condition**: something that has a duration
  - The fuel level is high
  - The alarm is on
- **State**: an abstraction of the attributes and links of an object (or entire system)
  - The controller is in the state self-test after the self-test button has been pressed and the rest-button has not yet been pressed
  - The tank is in the state too-low when the fuel level has been below level-low for alarm-threshold seconds

### Making a Phone Call Scenario

To make a call, the caller lifts receiver. The caller gets a dial tone and the caller dials digit (x). The dial tone ends. The caller completes dialing the number. The callee phone begins ringing at the same time a ringing begins in callee phone. When the callee answers the called phone stops ringing and ringing ends in callee phone. The phones are now connected. The caller hangs up and the phones are disconnected. The callee hangs up.

### Partial Class Diagram

![Partial Class Diagram](image)

### Event Trace

<table>
<thead>
<tr>
<th>Event Trace</th>
<th>The Caller</th>
<th>The Line</th>
<th>The Callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>caller lifts receiver</td>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>callee answers</td>
</tr>
<tr>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>dial tone begins</td>
<td>phone connected</td>
</tr>
<tr>
<td>dial tone ends</td>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>phone disconnected</td>
</tr>
<tr>
<td>caller hangs up</td>
<td></td>
<td></td>
<td>callee hangs up</td>
</tr>
</tbody>
</table>

### State Diagram for Scenario

![State Diagram for Scenario](image)

### Scenario 2

<table>
<thead>
<tr>
<th>Scenario 2</th>
<th>The Caller</th>
<th>The Line</th>
<th>The Callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>caller lifts receiver</td>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>callee answers</td>
</tr>
<tr>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>dial tone begins</td>
<td>phone connected</td>
</tr>
<tr>
<td>dial tone ends</td>
<td>dial tone begins</td>
<td>dial tone ends</td>
<td>phone disconnected</td>
</tr>
<tr>
<td>caller hangs up</td>
<td></td>
<td></td>
<td>callee hangs up</td>
</tr>
</tbody>
</table>
Modified State Machine

Conditions

Operations (AKA Actions)

Hierarchical State Machines

Information Hiding

Event Generalization

- Sometimes the state transitions are conditional
- Group states with similar characteristics
- Enables information hiding
- Simplifies the diagrams

- Related events can inherit properties from each other
- If an event at a lower level occurs - the event at a higher level also occurred
- Event attributes
  - mouse-down.device
  - mouse-down.location
  - mouse-up.character

- Actions are performed when a transition is taken or performed while in a state
- Actions are terminated when leaving the state

- Related events can inherit properties from each other
- If an event at a lower level occurs - the event at a higher level also occurred
- Event attributes
  - mouse-down.device
  - mouse-down.location
  - mouse-up.character
Concurrency

- Some states represent several concurrent concepts
- Concurrency is supported by the state machines
- Concurrent state machines are separated by dashed lines

Ambiguous Semantics 1

Is F transition ever taken? How?

Ambiguous Semantics 2

What happens when G is false after event E?

Are we stuck here?

Ambiguous Semantics 3

How many threads are running here?

What does this mean?

Ambiguous Semantics 4

Does this component get started?

Ambiguous Semantics 5

What is the semantics of message passing?
Queued?
Rendezvous?
Lost if no transition?
Transition Rules

- Find all the transitions with the trigger event
  - If there are none, the event is lost. This is *not* an error.
- Evaluate the guards (if any)
  - No guard = true guard
  - For false guard, ignore this transition
  - Guards can reference attributes of the class
- If more than one transition on a state survives, pick one at random.

More Transition Rules

- Descendants of actions (in a inheritance hierarchy) can trigger a transition
- Transitions in nested states take precedence over enclosing states.
- Null triggers “occur” when the state is done doing whatever it does.
  - A transition with a null trigger and a false guard never fires again.
- Concurrent threads have to be joined or terminated.

Transition Syntax

Event[Guard]/Action1;Action2;…;ActionN

Actions include: send(event)
Events include: timeout(), when(boolean)

Pulse[pulsemode]/count++

Sample triggers:
- Timeout(10s)/send(reset)
- Digit(d)/isvalid(d)/stash(d)

State Machines - Summary

- Events
  - instances in time
- Conditions
  - conditions over time
- States
  - abstraction of the attributes and associations
- Transitions
  - Takes the state machine from one state to the next
  - Triggered by events
  - Guarded by conditions
  - Cause actions to happen

- Internal actions
  - something performed in a state
- Hierarchies
  - allows abstraction and information hiding
- Paralelism
  - models concurrent concepts

When to use State Machines

- When you want to describe the behavior of one object for all (or at least many) scenarios that affect that object
- Not good at showing the interaction between objects
  - Use interaction diagrams or activity diagrams
- Probably not needed for all classes
  - Some methods prescribe this
  - Sometimes time consuming and questionable benefit

Coming up with the State Diagrams
Modeling Approach

- Prepare scenarios
  - Work with the customer
  - Start with normal scenarios
  - Add abnormal scenarios
- Identify events (often messages)
  - Group into event classes
- Draw some sequence diagrams
  - Find objects with complex functionality you want to understand better
- Build a state diagram for the complex classes

Scenario-1

- Room
- Controller
- Fuel Valve
- Burner
- Water Pump

Dynamic Model

- Water Pump
- Fuel Valve
- Burner

More Dynamic Model

- Room
- Water Valve
- Temp-Sensor

Even More Dynamic Model

- Controller
- Temperature
- Home Heating System
- All-Running
Identify Key Operations

- Operations from the object model
  - Accessing and setting attributes and associations (often not shown)
- Operations from events
  - All events represent some operation
- Operations from actions and activities
  - Actions and activities represent some processing activity within some object
- Operations from functions
  - Each function typically represents one or more operations
- Shopping list operations
  - Inherent operations (what should be there)

Complete OO Model

Iterate the Model

- Keep on doing this until you, your customer, and your engineers are happy with the model

Activity Diagrams

We Will Cover

- History of activity diagrams in UML
  - A highly personal perspective
- Activity diagrams
- Swimlanes
- When to use activity diagrams
  - When not to

Activity Diagrams

- Shows how activities are connected together
  - Shows the order of processing
  - Captures parallelism
- Mechanisms to express
  - Processing
  - Synchronization
  - Conditional selection of processing
- A glorified flowchart
Why Activity Diagrams

- Very good question
- Not part of any previous (UML related) method
- Introduced for activities, like business processes
- Introduced to sell products (drawing tools)
- Suitable for modeling of business activities
- UML and OO is becoming more prevalent in business applications
- Object frameworks are making an inroad
- Stay within one development approach and notation
- Generally a flowchart and I do not really see the need in OO modeling
- Probably because I do not do business systems

Coffee Example

HACS Use-Cases

Use case: Distribute Assignments
Actors: Instructor (initiator), Student
Type: Primary and essential
Description: The Instructor completes an assignment and submits it to the system. The instructor will also submit the delivery date, due date, and the class the assignment is assigned for. The system will at the due date mail the assignment to the student.

Cross Ref.: Requirements XX, YY, and ZZ
Use Cases: Configure HACS must be done before any user (Instructor or Student) can use HACS

Activity Diagrams for Use Cases

Swimlanes (Who Does What?)
### Problems with Activity Diagrams

- **NOT** good for design – not bad for biz process
  - “Flow” to OO is hard
- They are glorified flowcharts
  - Very easy to make a traditional data-flow oriented design
- Switching to the OO paradigm is hard enough as it is
  - Extensive use of activity charts can make this shift even harder
- However...
  - Very powerful when you know how to use them correctly

### When to Use Activity Diagrams

- Not clear how useful in OO modeling
  - Particularly when modeling control systems
- Useful when
  - Understanding workflow in an organization
  - Analyzing a use case (or collection of use cases)
  - Working with multi-threaded applications (maybe)
    - For instance, process control applications
  - Do not use activity diagrams
    - To figure out how objects collaborate
    - See how objects behave over time