OO System Models
Dynamic View

UML Sequence, Communication, State Diagrams
Objective

- Describe the evolutionary approach for using sequence diagrams to complete class diagrams
- Introduce the syntax of sequence diagrams
- Explain how message are interleaved between objects to build sequence diagrams
- Introduce CRUD Analysis & CRUD matrix
- Introduce communication (collaboration) Diagrams
- Explain state machine models for real time systems
The Dynamic View of a system may be described using UML diagrams:

- UML Sequence Diagrams
- UML Communication ‘UML 2.0’ (Collaboration) Diagrams
- UML State Diagrams
- UML Activity Diagrams
From Use Cases to: Objects, Attributes, Operations (methods) - “evolutionary”
Identifying objects

- Look for **nouns** in the SRS (System Requirements Specifications) document

- Look for **NOUNS** in use cases descriptions

- A **NOUN** may be
  - Object
  - Attribute of an object
Identifying Operations ‘methods’

- Look for verbs in the SRS (System Requirements Specifications) document

- Look for **VERBS** in use cases descriptions

- **A VERB** may be
  - translated to an **operation** or set of operations
  - A method is the code implementation of an operation.
**OO: Visibility of attributes or operations**

- Visibility: specifies the extent to which other classes can use a given class's attributes or operations.

Three levels of visibility:

- **+**: public level (usability extends to other classes)
- **#**: protected level (usability is open only to classes that inherit from original class)
- **-**: private level (only the original class can use the attribute or operation)
# OO: Visibility

Ex: *Public and private operations in a Hard Disk*

<table>
<thead>
<tr>
<th>HardDisk</th>
</tr>
</thead>
<tbody>
<tr>
<td>+modelName</td>
</tr>
<tr>
<td>+capacity</td>
</tr>
<tr>
<td>+producer</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>+read()</td>
</tr>
<tr>
<td>+write()</td>
</tr>
<tr>
<td>-adjustHeads()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
OOA and OOD

<table>
<thead>
<tr>
<th><strong>Analysis Model</strong></th>
<th><strong>Design Model</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>classes</td>
<td>objects</td>
</tr>
<tr>
<td>attributes</td>
<td>data structures</td>
</tr>
<tr>
<td>methods</td>
<td>algorithms</td>
</tr>
<tr>
<td>relationships</td>
<td>messaging</td>
</tr>
<tr>
<td>behavior</td>
<td>control</td>
</tr>
</tbody>
</table>
Object behaviour modelling

- A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case.

- **Sequence diagrams** (or collaboration diagrams) in the UML are used to model interaction between objects.
Sequence Diagram

- Class diagrams and object diagrams represent **static** information.

- In a functioning system, however, objects interact with one another, and these interactions occur over time.

- The UML sequence diagram shows the **time-based dynamics of the interaction**
System Sequence Diagram (SSD) for a UC

- For any UC Scenario (e.g. Happy Path)

Time

1 2
3 4
5

System

System as a whole
(no details of collaborating objects)
Message type in a sequence diagram

- **Simple** This is a transfer of control from one object to another.

- **Synchronous** If an object sends a synchronous message, it waits for an answer to that message before it proceeds with its business.

- **Asynchronous** If an object sends an asynchronous message, it doesn't wait for an answer before it proceeds.
Sequence Diagrams (SD)

- A SD represents time in the vertical direction.
  - Time starts at the top and progresses toward the bottom.
  - A message that's closer to the top occurs earlier in time than a message that's closer to the bottom.

- Horizontal direction shows the objects participating in the interaction

- More objects may be created during the interaction

- Objects may be destroyed when not needed any more
Sequence diagrams

- The actor-symbol initiates the sequence,
Object and class naming

- Named object: object name : class name
- Anonymous object (any object of the same class): class name
- Or less used: class name
Object and class naming

Named objects Ahmad and Hassan of class Student:

Ahmad : Student  Hassan : Student

• Anonymous (any) student obj

: Student

Or less used  : Student
Sequence Diagram Syntax

<table>
<thead>
<tr>
<th>AN ACTOR</th>
<th>![Actor Symbol]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN OBJECT</td>
<td>anObject:aClass</td>
</tr>
<tr>
<td>A LIFELINE</td>
<td></td>
</tr>
<tr>
<td>A FOCUS OF CONTROL</td>
<td></td>
</tr>
<tr>
<td>(Object activation)</td>
<td></td>
</tr>
<tr>
<td>A MESSAGE</td>
<td>aMessage()</td>
</tr>
<tr>
<td>OBJECT DESTRUCTION</td>
<td>X</td>
</tr>
</tbody>
</table>
Sequence Diagram
Ex: Issue of electronic items

:Library User

Ecat: Catalog
:Library Item
Lib1: NetServer

Issue
Display
Lookup
Message
Object Destruction
Compress
Accept licence
Issue licence
Deliver

Participating (collaborating) Objects
Life line
Activation period

Time
Activation period
Object Destruction
Compress
Deliver
Accept licence
Issue licence
Message
Display
Lookup

:Library User

Ecat: Catalog
:Library Item
Lib1: NetServer

Issue
Display
Lookup
Message
Object Destruction
Compress
Deliver
Accept licence
Issue licence
Message
Display
Lookup
Sequence Diagram: Inquire Item

Clerk

inquireOnItem (catalogID, prodID, size)

:AvailabilityHandler

inquireOnItem (prodID, size)

:Catalog

desc := getDescription ()

price := getPrice ()

quantity := getQty (size)

:ProductItem

quantity := getQOH ()

:CatalogProduct

(desc, price, quantity)

:InventoryItem
Request Appointment SD

Condition: { --- }
If a patient exists

Dennis: SAD
Fig: 8-1  W-30  100% of size
Fine Line Illustrations (516) 501-0400
Building a Sequence Diagram

1. Determine the **context** of the sequence diagram
2. Identify the **participating objects**
3. Set the lifeline for each object
4. Add **messages**
5. Place the focus of control on each object’s lifeline
6. Validate the sequence diagram
Use Case & Sequence diagram

- How many SDs for one UC?
- One SD or more
- Build a SD for each scenario in the UC
  - Happy path scenario
  - Alternative scenario(s)
  - Exception scenario(s)
Communication ‘UML 2.0’ (Collaboration) Diagrams

- Object diagram that shows message passing relationships instead of aggregation or generalization associations.

- Emphasize the flow of messages among objects
  - Recall: SD emphasize timing of messages
Ex: Communication (Collaboration) Diagram
## Elements of Communication (collaboration) Diagram

<table>
<thead>
<tr>
<th>Element</th>
<th>Icon Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>anActor</td>
</tr>
<tr>
<td>Object</td>
<td>anObject : aClass</td>
</tr>
<tr>
<td>Association</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>1: a Message()</td>
</tr>
<tr>
<td>Frame</td>
<td>Context</td>
</tr>
</tbody>
</table>
CRUD Analysis & CRUD matrix

- Interaction among objects (instances of classes)

- UC CRUD matrix:
  - for single Use Case

- System CRUD matrix
CRUD Analysis & CRUD matrix

<table>
<thead>
<tr>
<th></th>
<th>Customer</th>
<th>SearchReq</th>
<th>CDList</th>
<th>CD</th>
<th>Mkt Info</th>
<th>Review</th>
<th>Artist Info</th>
<th>Sample Clip</th>
<th>Shopping Cart</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
</tr>
<tr>
<td>SearchReq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>CDList</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mkt Info</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
</tr>
<tr>
<td>Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
</tr>
<tr>
<td>Artist Info</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
</tr>
<tr>
<td>Sample Clip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shopping Cart</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 8-9** CRUD Matrix for the Place Order Use Case
Building a Communication (Collaboration) Diagram

1. Determine the context of the diagram

2. Identify the participating objects and their associations

3. Layout objects and associations

4. Add messages

5. Validate the sequence diagram
State machine models

- These model the behaviour of the system in response to external and internal events.
- They show the system’s responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- Statecharts are an integral part of the UML.
Microwave oven model

- **Full power**
  - Full power
    - do: set power = 600
- **Half power**
  - Half power
    - do: set power = 300
  - do: display 'Ready'
    - do: display 'Waiting'
  - do: operate
  - do: operate oven
  - do: display time
- **Waiting**
  - do: display time
  - do: display 'Ready'
- **Enabled**
  - do: display 'Ready'
  - do: display 'Waiting'
  - do: operate
  - do: operate oven
  - do: display time
- **Set time**
  - do: get number
  - exit: set time
  - do: set power = 300
  - do: set power = 600
  - do: display time
- **Operation**
  - do: operate oven
  - do: display time
  - Cancel
  - Waiting

**States:**
- Waiting
- Half power
- Full power
- Enabled
- Set time
- Operation
Statecharts

- Allow the decomposition of a model into sub-models (see following slide)
- A brief description of the actions is included following the ‘do’ in each state
- Can be complemented by tables describing the states and the stimuli
## Microwave oven state description

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waiting</td>
<td>The oven is waiting for input. The display shows the current time.</td>
</tr>
<tr>
<td>Half power</td>
<td>The oven power is set to 300 watts. The display shows ‘Half power’.</td>
</tr>
<tr>
<td>Full power</td>
<td>The oven power is set to 600 watts. The display shows ‘Full power’.</td>
</tr>
<tr>
<td>Set time</td>
<td>The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.</td>
</tr>
<tr>
<td>Disabled</td>
<td>Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.</td>
</tr>
<tr>
<td>Operation</td>
<td>Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows ‘Cooking complete’ while buzzer is sounding.</td>
</tr>
</tbody>
</table>
**Microwave oven stimuli**

<table>
<thead>
<tr>
<th>Stimulus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half power</td>
<td>The user has pressed the half power button</td>
</tr>
<tr>
<td>Full power</td>
<td>The user has pressed the full power button</td>
</tr>
<tr>
<td>Timer</td>
<td>The user has pressed one of the timer buttons</td>
</tr>
<tr>
<td>Number</td>
<td>The user has pressed a numeric key</td>
</tr>
<tr>
<td>Door open</td>
<td>The oven door switch is not closed</td>
</tr>
<tr>
<td>Door closed</td>
<td>The oven door switch is closed</td>
</tr>
<tr>
<td>Start</td>
<td>The user has pressed the start button</td>
</tr>
<tr>
<td>Cancel</td>
<td>The user has pressed the cancel button</td>
</tr>
</tbody>
</table>
Microwave oven operation

- **Operation**
  - **Cook**
    - do: run generator
  - **Done**
    - do: buzzer on for 5 secs.
  - **Alarm**
    - do: display event
  - **Disabled**
  - **Waiting**
  - **Checking**
    - do: check status
  - **Turntable fault**
  - **Emitter fault**
  - **Timeout**
  - **OK**
  - **Disable**
  - **Delay**
  - **Open**
  - **Cancel**

- **States**
  - **Cook**
  - **Done**
  - **Alarm**
  - **Disabled**
  - **Waiting**

- **Events**
  - **Do**
  - **Display**
  - **Run**
  - **Buzzer**
  - **Status**
  - **Turntable**
  - **Emitter**
  - **Fault**
  - **OK**
  - **Timeout**
  - **Open**
  - **Cancel**
UML: State Transition

- **password entered**
  - **compare password = incorrect**
    - **control panel**
    - **"reenter"**
    - **compare password = incorrect**
    - **compare password = correct**
  - **"selecting"**
  - **"selecting"**
  - **activation successful**

- **"at rest"**
  - **control panel**
  - **password entered**
  - **compare password = incorrect**
  - **"comparing"**
  - **control panel**