Formal System Design Process with UML

Use a formal process & tools to facilitate and automate design steps:

- Requirements
- Specification
- System architecture
- Coding/chip design
- Testing

Text: Chapter 1.3, 1.4
Object-Oriented Design

- Describe system/design as interacting objects
  - Across multiple levels of abstraction
  - Visualize elements of a design
- **Object** = state + methods.
  - **State** defined by set of “attributes”
    - each object has its own identity.
    - user cannot access state directly
  - **Methods** provide an abstract interface to the object attributes.
- Objects map to system HW/SW elements
Objects and classes

- **Class**: an object **type** that defines
  - **state** elements for all objects of this type.
    - Each object has its own state.
    - Elements not directly accessible from outside
    - State values may change over time.
- **methods** (operations) used to interact with all objects of this type.
  - State elements accessed through methods
Object-oriented design principles

- Some objects closely correspond to real-world objects.
  - Other objects may be useful only for description or implementation.

- **Abstraction**: list only info needed
  - Limit listed object info for each given purpose

- **Encapsulation**: mask internal op’s/info
  - Objects provide interfaces to read/write the object state
  - Hide object’s implementation from the rest of the system.
Unified Modeling Language (UML)

• Developed by Grady Booch et al.
  • Version 1.0 in 1997 (current version 2.4.1)
  • Maintained by Object Management Group (OMG) – www.omg.org
  • Resources (tutorials, tools): www.uml.org

• Goals:
  • object-oriented;
  • visual;
  • useful at many levels of abstraction;
  • usable for all aspects of design.

• Encourage design by successive refinement
  • Don’t rethink at each level
  • CASE tools assist refinement/design
UML Elements

- **Model elements**
  - classes, objects, interfaces, components, use cases, etc.

- **Relationships**
  - associations, generalization, dependencies, etc.

- **Diagrams**
  - class diagrams, use case diagrams, interaction diagrams, etc.
  - constructed of model elements and relationships
Structural vs. Behavioral Models

- **Structural**: describe system components and relationships
  - *static* models
  - objects of various classes

- **Behavioral**: describe the behavior of the system, as it relates to the structure
  - *dynamic* models
UML Diagram Types

- **Use-case**: help visualize functional requirements (user-system interaction)
- **Class**: types of objects & their relationships
- **Object**: specific instances of classes
- **Interaction diagrams** (dynamic)
  - **Sequence**: how sequences of events occur (message-driven)
  - **Collaboration**: focus on object roles
- **Statechart**: describe behavior of system/objects
- **Component**: physical view of system (code, HW)
- **Others ...
UML use-case diagrams

- Describe behavior user sees/expects ("what" – not "how")
- Describe user interactions with system objects
- Users = actors (anyone/anything using the system)

Example: Data acquisition system

- Translate to algorithm for system design
DAQ system use-case description

- **User**
  - Select measure volts mode
  - Select measurement range or autorange

- **System**
  - If range specified
    - Configure to specified gain
    - Make measurement
      - If in range – display results
      - If exceed range – display largest value and flash display
  - If auto range
    - Configure to midrange gain
    - Make measurement
      - If in range – display mode
      - If above/below range – adjust gain to next range and repeat
      - If exceed range – display largest value and flash display
UML class

Display

- pixels
- elements
- menu_items

- mouse_click()
- draw_box

class name
Attributes/
State elements
operations/
methods

Class diagram: shows relationship between classes
Object diagram: static configuration of objects in a system

UML object

- **Object name**: d1: Display
- **Class name**: Display
- **Attributes**:
  - pixels: array[] of pixels
  - elements
  - menu_items

**Comment**:
- pixels is a 2-D array

**Object Diagram**:
- Green box: comment
- Red box: object name
- Yellow box: attributes
The class interface

- Implementation of the object is hidden by the class interface (encapsulation)
- Operations provide the abstract interface between the class’ implementation and other classes.
- An operation can examine/modify the object’s state.
- Operations may have arguments, return values.
Choose your interface properly

• If the interface is too small/specialized:
  • object is hard to use for even one application;
  • even harder to *reuse*.

• If the interface is too large:
  • class becomes too cumbersome for designers to understand;
  • implementation may be too slow;
  • spec and implementation are probably buggy.
Relationships between classes and objects

- **Association**: objects “related” but one does not own the other.

  ![Diagram: Association]

- **Aggregation**: complex object is made of several smaller objects.

  ![Diagram: Aggregation]

- **Composition**: *strong* aggregation: part may belong to only one whole – deleting whole deletes parts.

  ![Diagram: Composition]

- **Generalization**: define one class in terms of another. Derived class inherits properties.

  ![Diagram: Generalization]
Association Example

Optionally – show “direction” of association

SendsNumberTo
Aggregation/Composition Examples

- **List**
  - **Atom**
    - Atoms may be in other modules

- **Rectangle**
  - **Point**
    - Points can only be on one rectangle

aggregation

composition

Aggregation/Composition Examples

- **n..m** - between $n$ and $m$ instances
- **0..* - any number of instances (or none)
- **1..* - at least one instance
- **1** - exactly one instance
Generalization/Class derivation

- May want to define one class in terms of another more “general” class.
  - Instead of creating a new class
- Derived class inherits attributes & operations of base class.

![UML generalization diagram]

- Derived_class
- Base_class

(child class)

(parent class)

UML generalization
Class derivation example

Parent class

Display

- pixels
- elements
- menu_items

derived classes

BW_display

child class

draw_box

Color_display

child class

generalizations

set_pixel() mouse_click()
Multiple inheritance

- Speaker
- Display

Multimedia_display

derived class inherits properties of both base classes
Generalization example
Links and associations

- **Association**: describes relationship between **classes**.
  - Association & class = **abstract**

- **Link**: describes relationships between **objects**.
  - Link & object = **physical**
Association & link examples

Class Diagram

- message
  - msg: ADPCM_stream
  - length: integer

- message set
  - count: integer

# contained messages
0..* 1
Contains
(association)

Object Diagram

- m1: message
  - msg = msg1
  - length = 1102

- m2: message
  - msg = msg2
  - length = 2114

- Msg: message set
  - count = 2
Object & Class Diagram Example

Object diagram

Class diagram
/* Define the Display class */
class Display {
    pixels : pixelttype[IMAX,JMAX]; /* attributes */
public:
    /* methods */
    Display() { } /* create instance */
pixelttype pixel(int i, int j) { return pixels[i,j]; }
    void set_pixel(pixelttype val, int i, int j) { pixels[i,j] = val; }
}
Instantiating an object of a class in C++

/* instantiate Display object d1 */
Display d1;

/* manipulate object d1 */
apixel = d1.pixel(0,0);
Behavioral description

- Several ways to describe behavior:
  - internal view;
  - external view.

- Dynamic models:
  - **State diagram**: state-dependent responses to events
  - **Sequence diagram**: message flow between objects over time
  - **Collaboration diagram**: relationships between objects

- Specify:
  - intermodule interactions
  - order of task executions
  - what can be done in parallel
  - alternate execution paths
  - when tasks active/inactive
State machines

Similar to sequential circuit state diagrams
Event-driven state machines

- Behavioral descriptions are written as event-driven state machines.
  - Machine changes state when receiving an input.
- An event may come from inside or outside of the system.
  - **Signal**: asynchronous event.
  - **Call**: synchronized communication.
  - **Timer**: activated by time.
- May also have state changes without events
  - When some condition is satisfied
Signal event

{{{signal}}}

`mouse_click`

lefortright: button
x, y: position

event declaration

```
mouse_click(x,y,button)
```
event description
Call event

draw_box(10,5,3,2,blue)

c ←→ d

Timer event

tm(time-value)

e ←→ f
Example: click on a display

start

mouse_click(x,y,button)/find_region(region)

region = menu/which_menu(i)
call_menu(I)

called menu item

region = drawing/__
find_object(objid)

region = menu/
which_menu(i)
call_menu(I)

got menu item

highlight(objid)

object highlighted

found object

finish
Sequence diagram

- Shows sequence of operations over time.
- Relates behaviors of multiple objects.
- Objects listed at top from left to right
  - Each object has a time line (shown as dashed line)
  - Focus of control (shown as a rectangle) indicates when object is “active”
  - Actions between objects shown as horizontal lines/arrow
Sequence diagram example

Programs on a CPU: only one has control at a time

m: Main

f1: Function

f2: Function

f1(p1) → f2(p2) → return(r2)

time

box = “focus of control”

return(r1)
Sequence diagram example

m: Mouse
d1: Display
u: Menu

mouse_click(x,y,button)
which_menu(x,y,i)
call_menu(i)

box = "focus of control"
lifelines

time
Collaboration Diagram

- Show relationship between object messages passed between them
  - Objects as icons
  - Messages as arrows
  - Arrows labeled with sequence numbers to show order of events
Example: Cell phone class diagram

Cell phone use case: Make call

1. User enters number (presses buttons)
2. Update display with digits
3. Dialer generates tones for digits – emit from speaker
4. User presses “send”
5. “In use” indicator lights on display
6. Cell radio connects to network
7. Digits sent to network
8. Connection made to called party

1.2 EmitTone(code)
1* Digit(code)
2:Send()
2.1 Connect(pno)
1.1 DisplayDigit(code)

Summary

- Object-oriented design helps us organize a design.
- UML is a transportable system design language.
  - Provides structural and behavioral description primitives.

- Example: Model train set (Section 1.4)