Chapter 8
Analysis Modeling, Part 2/2

Moonzoo Kim
CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/cs550-07
Overview of Ch 8. Building the Analysis Model

- **April 10: ch 8.1- ch 8.5**
  - 8.1 Requirement Analysis
  - 8.2 Analysis Modeling Approaches
  - 8.3 Data Modeling Concepts
  - 8.4 Object-Oriented Analysis
  - 8.5 Scenario-based modeling

- **April 12: ch 8.6- ch 8.8**
  - 8.6 Flow-oriented modeling
  - 8.7 Class-based modeling
  - 8.8 Creating a behavioral model
Flow-Oriented Modeling

• Represents how data objects are transformed as they move through the system.

• A data flow diagram (DFD) is the diagrammatic form that is used.

• Considered by many to be an ‘old school’ approach.
  • Flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements.
The Flow Model

Every computer-based system is an information transform....
Flow Modeling Notation

- external entity
- process
- data flow
- data store
External Entity

A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

*Data must always originate somewhere and must always be sent to something*
Process

A data transformer (changes input to output)

Examples: compute taxes, determine area, format report, display graph

Data must always be processed in some way to achieve system function
Data Flow

Data flows through a system, beginning as input and be transformed into output.

- base
- height
- compute triangle area
- area
Data Stores

Data is often stored for later use.

- sensor #
- report required
- sensor number
- sensor data
- sensor #, type, location, age
- type, location, age
Data Flow Diagramming: Guidelines

- All icons must be labeled with meaningful names
- The DFD evolves through a number of levels of detail
- Always begin with a context level diagram (also called level 0)
- Always show external entities at level 0
- Always label data flow arrows
- Do not represent procedural logic
Constructing a DFD—I

- review the data model to isolate data objects and use a grammatical parse to determine “operations”
- determine external entities (producers and consumers of data)
- create a level 0 DFD
Level 0 DFD Example

user

processing request

digital video processor

requested video signal

monitor

video source

NTSC video signal
Constructing a DFD—I

- Write a narrative describing the transform
- Parse to determine next level transforms
- “balance” the flow to maintain data flow continuity
- Develop a level 1 DFD
- Use a 1:5 (approx.) expansion ratio
The Data Flow Hierarchy

level 0

level 1

x \rightarrow \text{P} \rightarrow y

p1 \rightarrow p2 \rightarrow p4 \rightarrow 5

p3 \rightarrow p4

a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g

level 0

level 1
Flow Modeling Notes

- each bubble is refined until it does just one thing
- the expansion ratio decreases as the number of levels increase
- most systems require between 3 and 7 levels for an adequate flow model
- a single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)
Process Specification (PSPEC)

- narrative
- pseudocode (PDL)
- equations
- tables
- diagrams and/or charts
DFDs: A Look Ahead

Maps into

analysis model

design model
Control Flow Diagrams

- Represents “events” and the processes that manage events
- An “event” is a Boolean condition that can be ascertained by:
  - listing all sensors that are "read" by the software.
  - listing all interrupt conditions.
  - listing all "switches" that are actuated by an operator.
  - listing all data conditions.
  - recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC inputs/outputs.
The Control Model

- the control flow diagram is "superimposed" on the DFD and shows events that control the processes noted in the DFD
- control flows—events and control items—are noted by dashed arrows
- a vertical bar implies an input to or output from a control spec (CSPEC) — a separate specification that describes how control is handled
- a dashed arrow entering a vertical bar is an input to the CSPEC
- a dashed arrow leaving a process implies a data condition
- a dashed arrow entering a process implies a control input read directly by the process
- control flows do not physically activate/deactivate the processes—this is done via the CSPEC
Class-Based Modeling

- Identify analysis classes by examining the problem statement
- Use a “grammatical parse” to isolate potential classes
- Identify the attributes of each class
- Identify operations that manipulate the attributes
Analysis Classes

- **External entities** (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
- **Things** (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
- **Occurrences or events** (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
- **Roles** (e.g., manager, engineer, salesperson) played by people who interact with the system.
- **Organizational units** (e.g., division, group, team) that are relevant to an application.
- **Places** (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
- **Structures** (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.
Selecting Classes—Criteria

- retained information
- needed services
- multiple attributes
- common attributes
- common operations
- essential requirements
Class Diagram

- **RoofPlan**
  - type
  - name
  - outsideDimensions
  - determineType()
  - positionRoofplan
  - scale()
  - changeColor()

- **Camera**
  - type
  - ID
  - location
  - fieldView
  - panAngle
  - ZoomSetting
  - determineType()
  - translateLocation()
  - displayID()
  - displayView()
  - displayZoom()

- **Wall**
  - type
  - wallDimensions
  - determineType()
  - computeDimensions()

- **WallSegment**
  - type
  - startCoordinates
  - stopCoordinates
  - nextWallSegment
  - determineType()
  - draw()

- **Window**
  - type
  - startCoordinates
  - stopCoordinates
  - nextWindow
  - determineType()
  - draw()

- **Door**
  - type
  - startCoordinates
  - stopCoordinates
  - nextDoor
  - determineType()
  - draw()
CRC Modeling

- Analysis classes have “responsibilities”
  - *Responsibilities* are the attributes and operations encapsulated by the class

- Analysis classes collaborate with one another
  - *Collaborators* are those classes that are required to provide a class with the information needed to complete a responsibility.
  - In general, a collaboration implies either a request for information or a request for some action.
## CRC Modeling

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FloorPlan</td>
<td>incorporates walls, doors and windows</td>
</tr>
<tr>
<td></td>
<td>shows position of video cameras</td>
</tr>
<tr>
<td></td>
<td>defines floor plan name/type</td>
</tr>
<tr>
<td></td>
<td>manages floor plan positioning</td>
</tr>
<tr>
<td></td>
<td>scales floor plan for display</td>
</tr>
<tr>
<td></td>
<td>scales floor plan for display</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>defines floor plan name/type</td>
<td></td>
</tr>
<tr>
<td>manages floor plan positioning</td>
<td></td>
</tr>
<tr>
<td>scales floor plan for display</td>
<td></td>
</tr>
<tr>
<td>scales floor plan for display</td>
<td></td>
</tr>
<tr>
<td>incorporates walls, doors and windows</td>
<td>Wall</td>
</tr>
<tr>
<td>shows position of video cameras</td>
<td>Camera</td>
</tr>
</tbody>
</table>
**Class Types**

- *Entity classes*, also called *model* or *business* classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).

- *Boundary classes* are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.

- *Controller classes* manage a “unit of work” [UML03] from start to finish. That is, controller classes can be designed to manage
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.
Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem.
- Each responsibility should be stated as generally as possible.
- Information and the behavior related to it should reside within the same class.
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.
Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- three different generic relationships between classes [WIR90]:
  - the *is-part-of* relationship
  - the *has-knowledge-of* relationship
  - the *depends-upon* relationship
Composite Aggregate Class
Reviewing the CRC Model

- All participants in the review (of the CRC model) are given a subset of the CRC model index cards.
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate).
- All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.
- The review leader reads the use-case deliberately.
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.
  - When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card.
    - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.
Associations and Dependencies

- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called *associations*
  - Associations can be refined by indicating *multiplicity* (the term *cardinality* is used in data modeling)
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a *dependency relationship* is established
Multiplicity

Wall

WallSegment

Window

Door

is used to build

1..*

0..*

is used to build

0..*
Dependencies

DisplayWindow

Camera

<<access>>

{password}
Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping.
- The + sign preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package. A minus sign indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.
Analysis Packages

- Environment
  - Tree
  - Landscape
  - Road
  - Wall
  - Bridge
  - Building
  - Visual Effect
  - Scene

- Rules Of The Game
  - Rules Of Movement
  - Constraints On Action

- Characters
  - Player
  - Protagonist
  - Antagonist
  - Supporting Role
Behavioral Modeling

- The behavioral model indicates how software will respond to external events or stimuli. To create the model, the analyst must perform the following steps:
  - Evaluate all use-cases to fully understand the sequence of interaction within the system.
  - Identify events that drive the interaction sequence and understand how these events relate to specific objects.
  - Create a sequence for each use-case.
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy and consistency.
State Diagram for the ControlPanel Class

Figure 8.20 pg 251 in SEPA
The States of a System

- **State**
  - a set of observable circumstances that characterizes the behavior of a system at a given time.

- **state transition**
  - the movement from one state to another.

- **Event**
  - an occurrence that causes the system to exhibit some predictable form of behavior.

- **Action**
  - process that occurs as a consequence of making a transition.
Behavioral Modeling

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another
  - How does the system change state?
    - indicate event
    - indicate action
- draw a state diagram or a sequence diagram
Sequence Diagram

Figure 8.27 Sequence diagram (partial) for SafeHome security function
Specification Guidelines

- use a layered format that provides increasing detail as the "layers" deepen
- use consistent graphical notation and apply textual terms consistently (stay away from aliases)
- be sure to define all acronyms
- be sure to include a table of contents; ideally, include an index and/or a glossary
- write in a simple, unambiguous style (see "editing suggestions" on the following pages)
- always put yourself in the reader's position, "Would I be able to understand this if I wasn't intimately familiar with the system?"
Specification Guidelines

Be on the lookout for persuasive connectors, ask why?
   keys: certainly, therefore, clearly, obviously, it follows that ...

Watch out for vague terms
   keys: some, sometimes, often, usually, ordinarily, most, mostly ...

When lists are given, but not completed, be sure all items are understood
   keys: etc., and so forth, and so on, such as

Be sure stated ranges don't contain unstated assumptions
   e.g., Valid codes range from 10 to 100. Integer? Real? Hex?

Beware of vague verbs such as handled, rejected, processed, ...

Beware "passive voice" statements
   e.g., The parameters are initialized. By what?

Beware "dangling" pronouns
   e.g., The I/O module communicated with the data validation module and
   its control flag is set. Whose control flag?
Specification Guidelines

1. When a term is explicitly defined in one place, try substituting the definition for other occurrences of the term.

2. When a structure is described in words, draw a picture.

3. When a structure is described with a picture, try to redraw the picture to emphasize different elements of the structure.

4. When symbolic equations are used, try expressing their meaning in words.

5. When a calculation is specified, work at least two examples.

6. Look for statements that imply certainty, then ask for proof keys; always, every, all, none, never.

7. Search behind certainty statements to ensure restrictions or limitations are realistic.