Chapter 8
Analysis Modeling

Moonzoo Kim
CS Division of EECS Dept.
KAIST

Overview of Ch 8. Building the Analysis Model

- 8.1 Requirement Analysis
- 8.2 Analysis Modeling Approaches
- 8.3 Data Modeling Concepts
- 8.4 Object-Oriented Analysis
- 8.5 Scenario-based modeling
- 8.6 Flow-oriented modeling
- 8.7 Class-based modeling
- 8.8 Creating a behavioral model

KAIST

Requirements Analysis

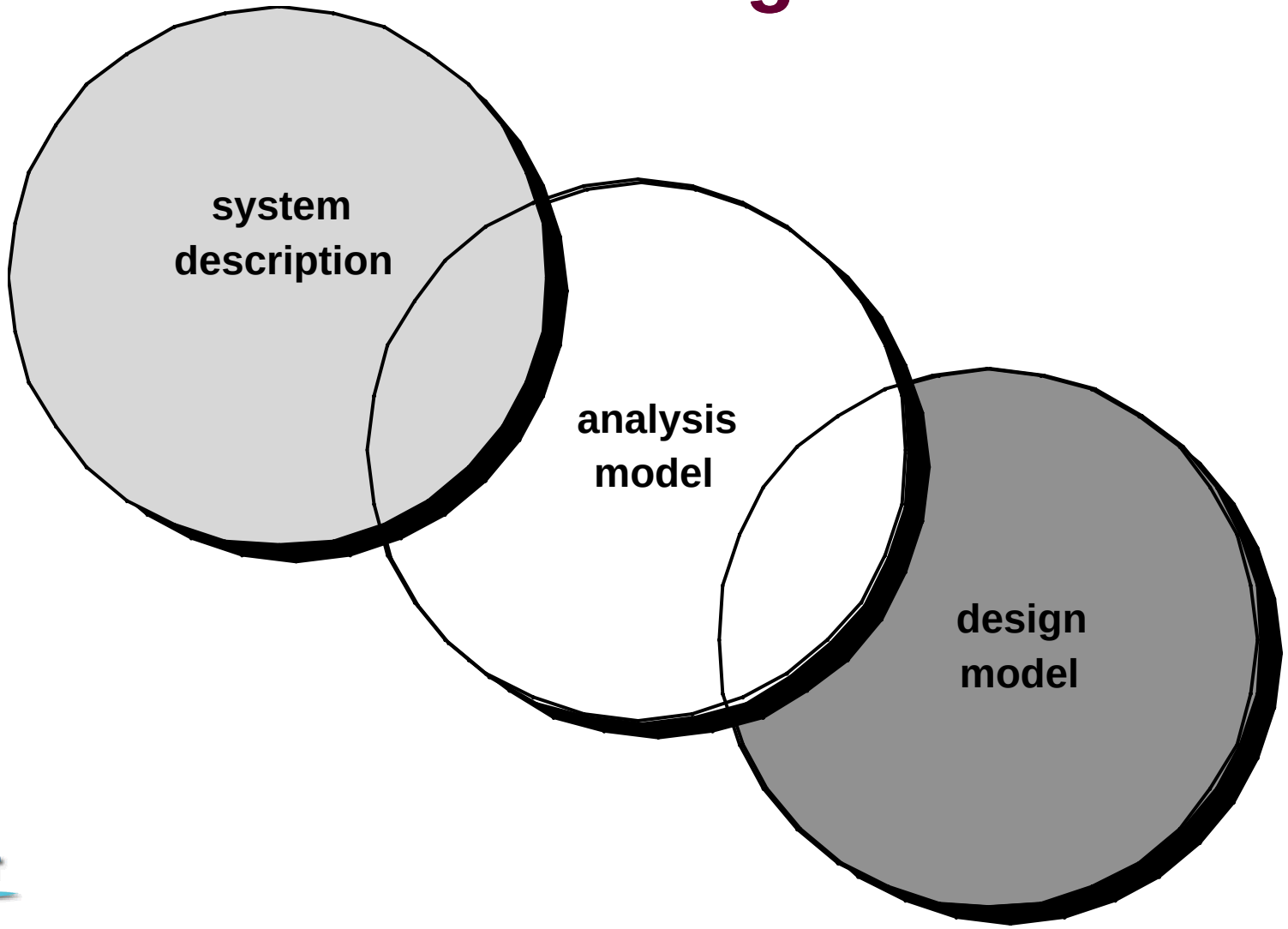
- At a technical level, SE begins with a building an analysis model of a target system
- Requirements analysis
  - specifies software's operational characteristics
  - indicates software's interface with other system elements
  - establishes constraints that software must meet
- Objectives
  1. To describe what the customer requires
  2. Establish a basis for the creation of a SW design
  3. To define a set of requirements that can be validated once the software is built

KAIST

Requirements Analysis

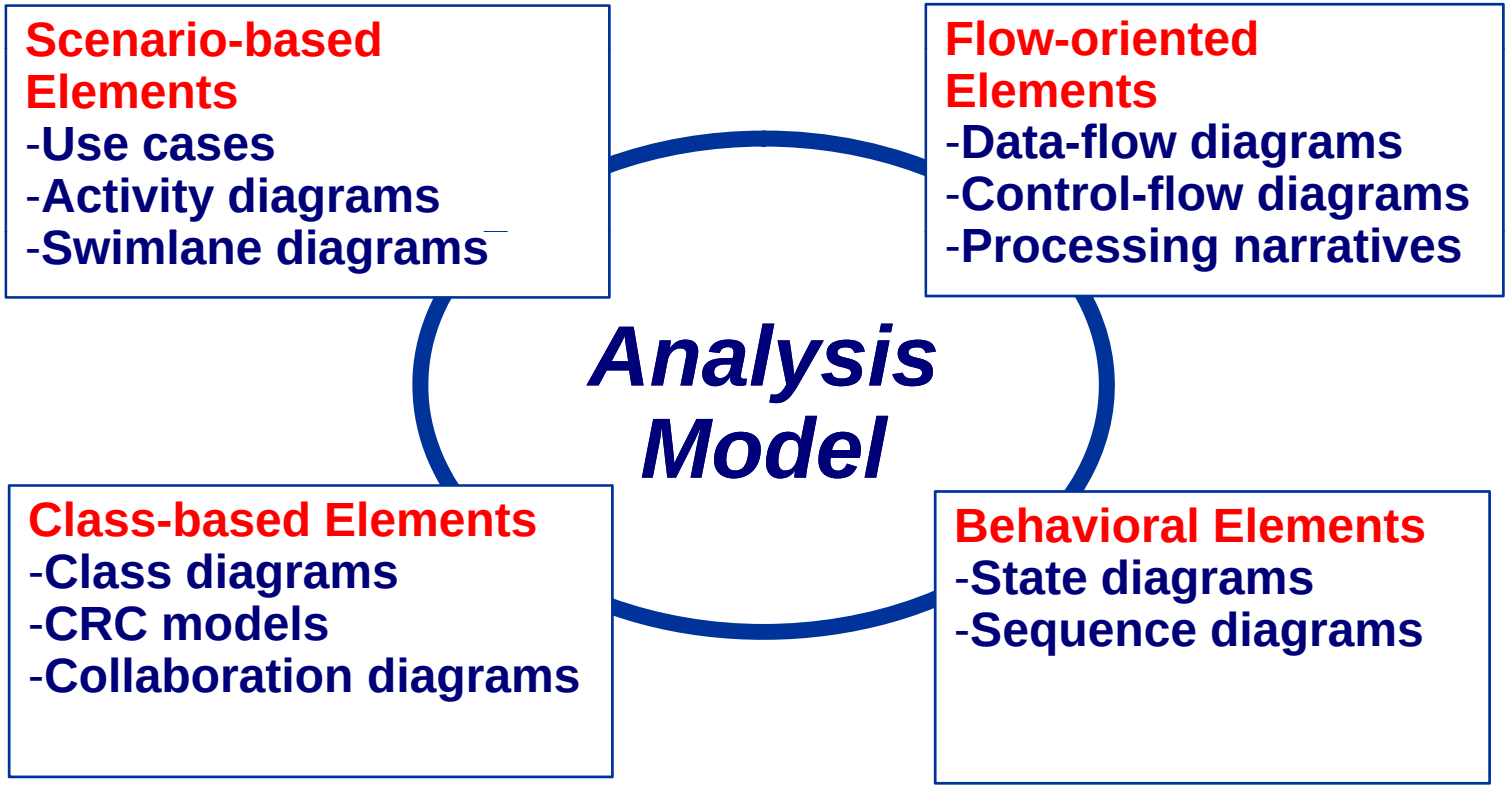
- Requirements analysis allows the software engineer to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
    - see Ch 7. “Requirements Engineering”
  - build models that depict
    - user scenarios
    - functional activities
    - problem classes and their relationships
    - system and class behavior
    - the flow of data as it is transformed.

A Bridge



KAIST

Elements of the Analysis Model



Rules of Thumb

1. The model should focus on requirements that are visible within the problem or business domain.
   - The level of abstraction should be relatively high.
2. Each element of the analysis model should
   - add to an overall understanding of software requirements
   - provide insight into the
     - information domain
     - function of the system
     - behavior of the system
3. Delay consideration of infrastructure and other non-functional models until design.
4. Minimize coupling throughout the system.
5. Be certain that the analysis model provides value to all stakeholders.
6. Keep the model as simple as it can be.

Domain Analysis

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain . . . [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks . . .

Donald Firesmith

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.

KAIST

Data Modeling

- Analysis modeling often begins with data modeling
  - Examines data objects independently of processing
  - Focuses attention on the data domain
  - Indicates how data objects relate to one another
- Relationship among data objects can be expressed in UML very well

- Typical data objects
  - External entities
    - printer, user, sensor
  - Things
    - reports, displays, signals
  - Occurrences or events
    - interrupt, alarm
  - Roles
    - manager, engineer, salesperson
  - Organizational units
    - division, team
  - Places
    - manufacturing floor
  - Structures
    - employee record

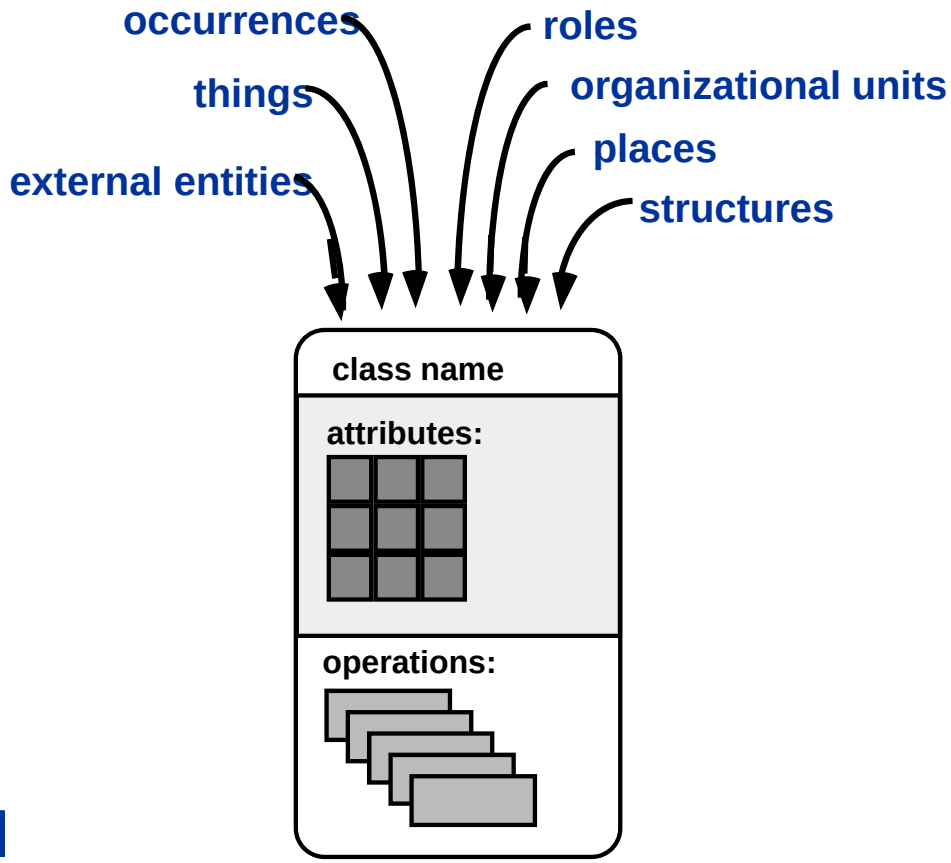
KAIST

Object-Oriented Concepts

- Must be understood to apply class-based elements of the analysis model
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance

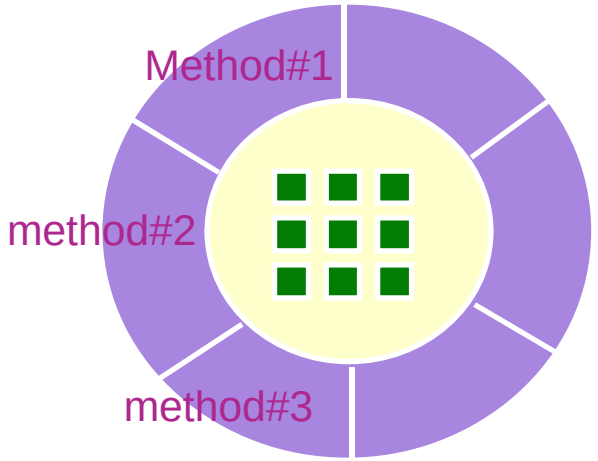
Classes

- Object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - “blueprint” ... describing a collection of similar items
- A superclass establishes a hierarchy of classes
- Once a class of items is defined, a specific instance of the class can be identified



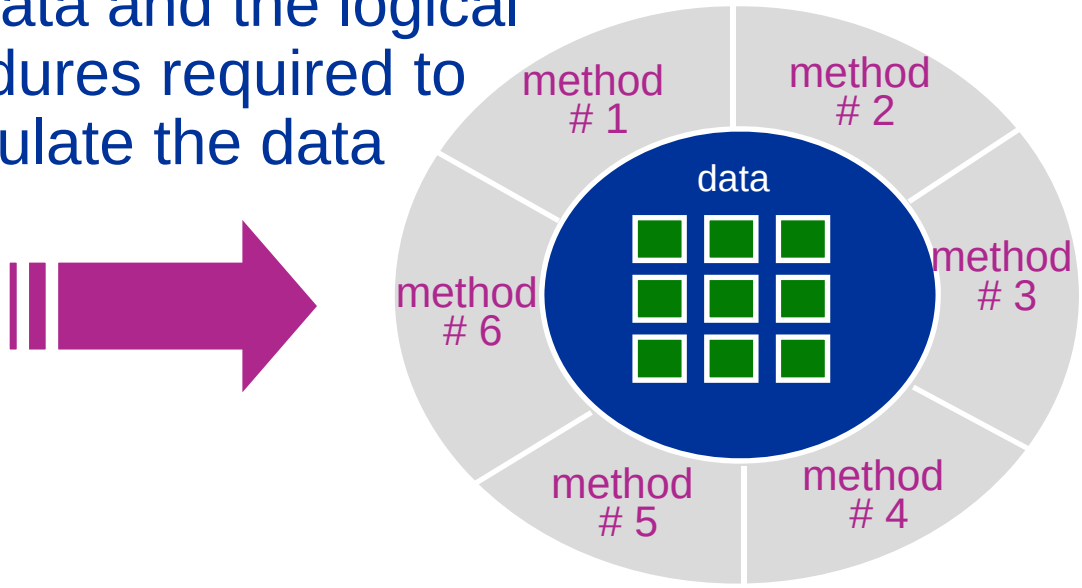
Methods
(a.k.a. Operations, Services)

An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class.



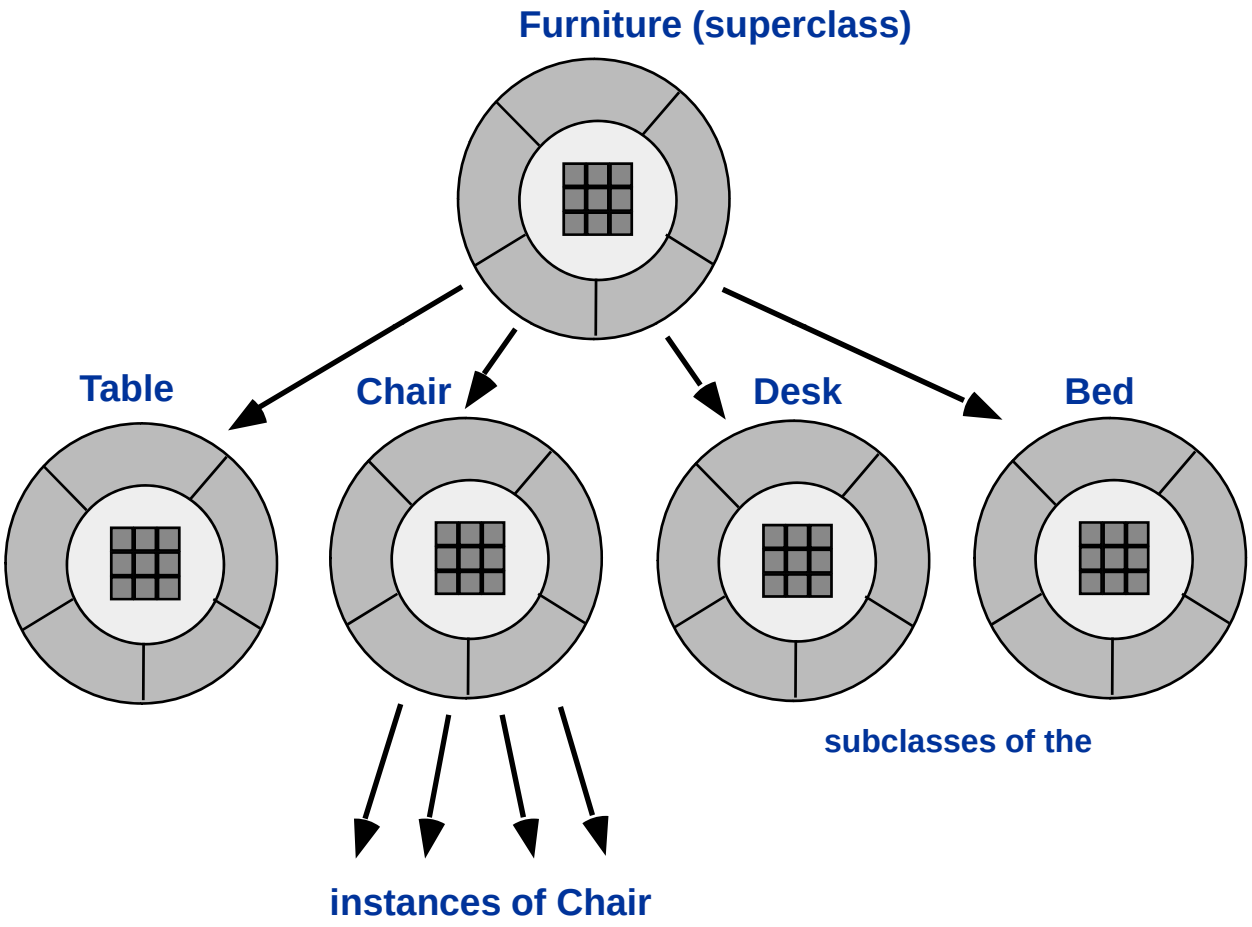
Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data



Achieves “information hiding”

Class Hierarchy



How to Define All Classes

1. Basic user requirements must be communicated between the customer and the SW engineer
2. Classes must be identified
   - Attributes and methods are to be defined
3. A class hierarchy is defined
4. Object-to-object relationships should be represented
5. Object behavior must be modeled
6. Tasks 1 through 5 are repeated until the model is complete

KAIST

Scenario-Based Modeling

"[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases)." Ivar Jacobson

(1) What should we write about?

(2) How much should we write about it?

~~(3) How detailed should we make our description?~~

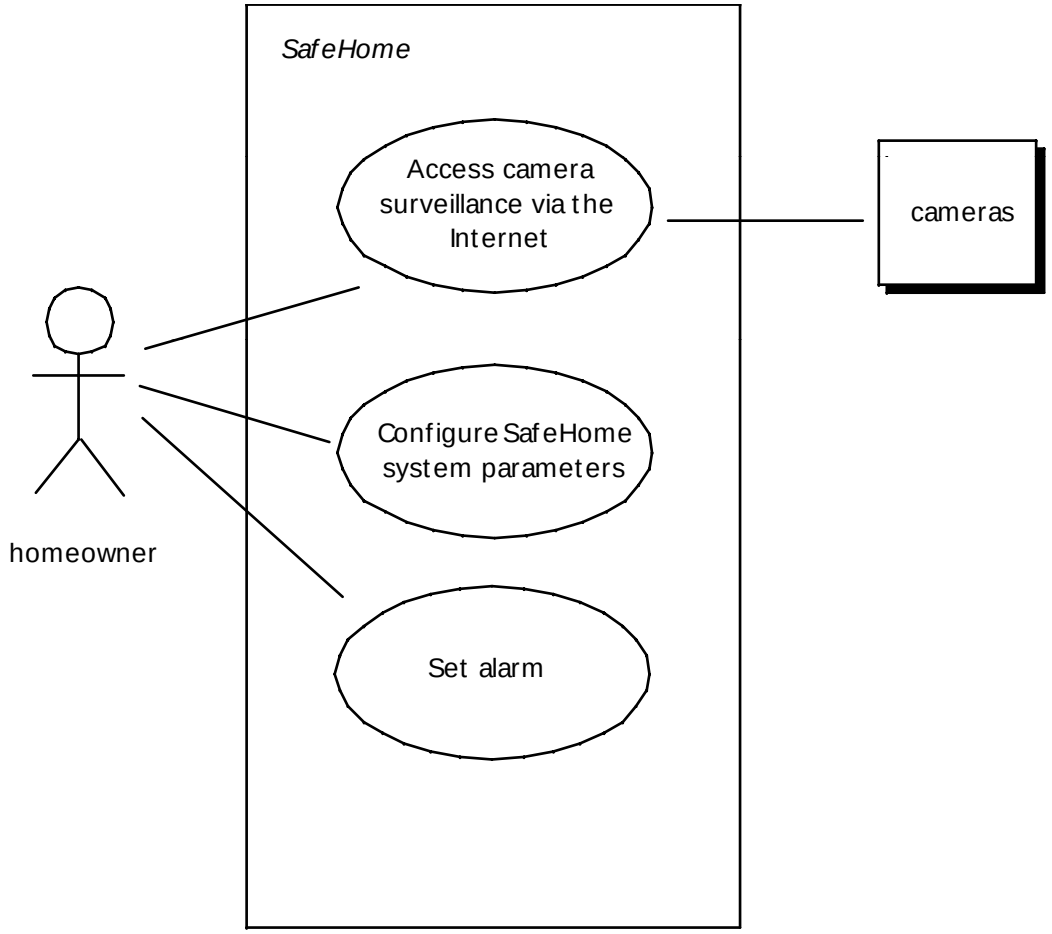
(4) How should we organize the description?

KAIST

Use-Cases

- a scenario that describes a “thread of usage” for a system
- *actors* represent roles people or devices play as the system functions
- *users* can play a number of different roles for a given scenario
- Developing a use case
  - What are the main tasks or functions that are performed by the actor?
  - What system information will the actor acquire, produce or change?
  - What information does the actor desire from the system?

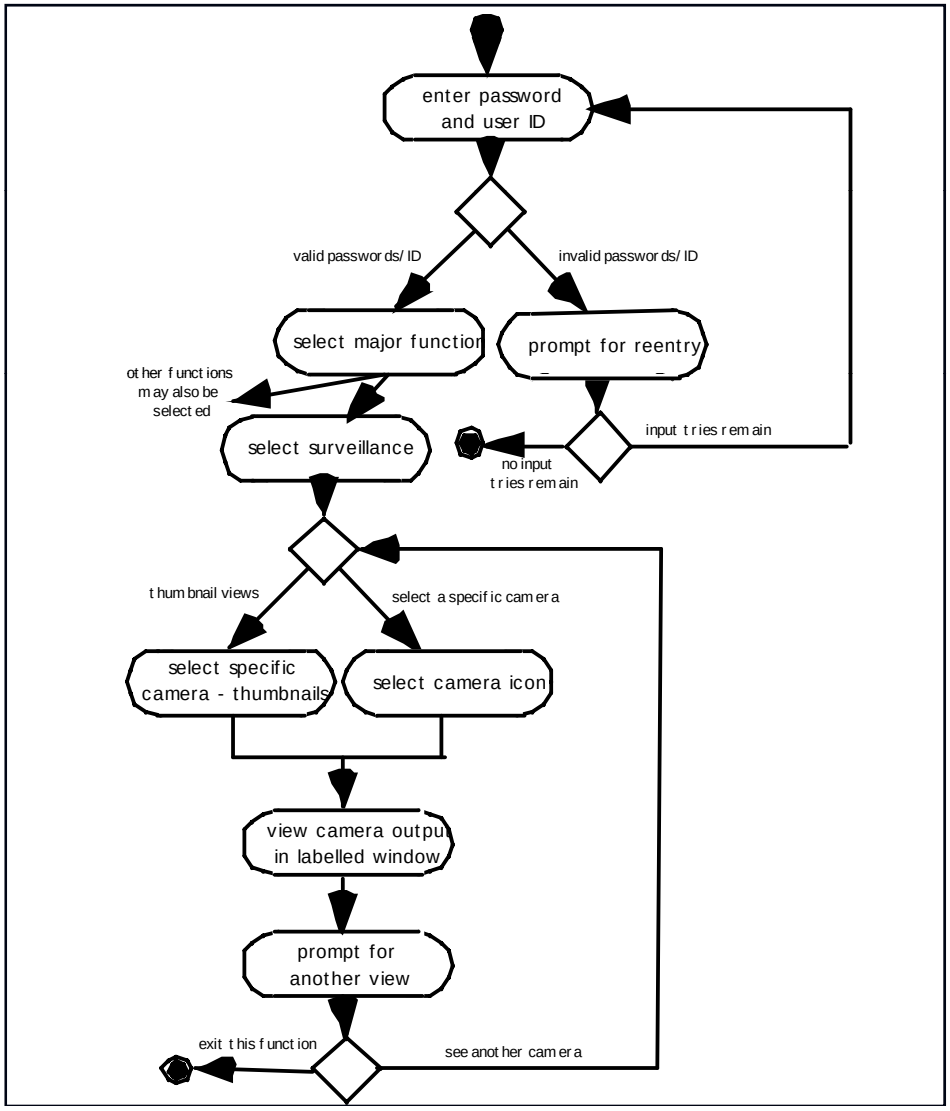
Use-Case Diagram



Activity Diagram

- Supplements the use-case by providing a diagrammatic representation of procedural flow

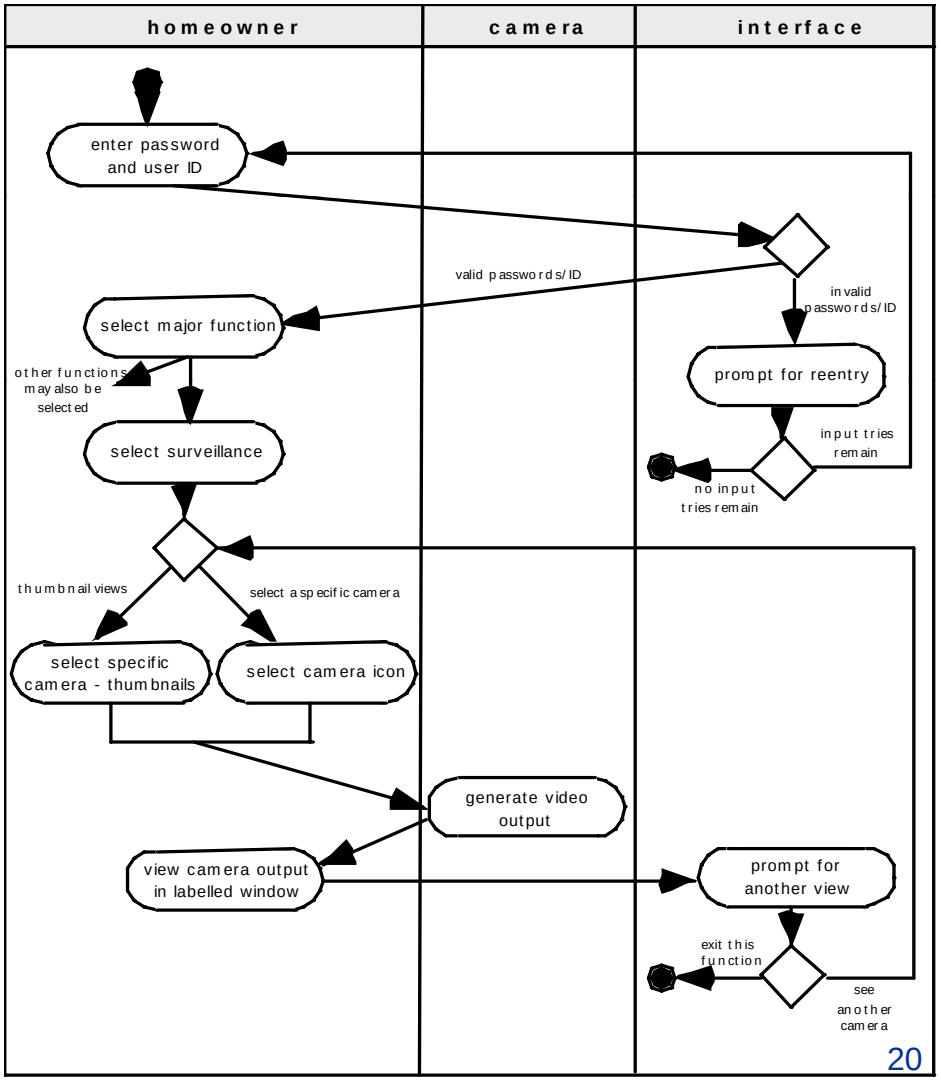
  (Fig 8.7 of 224 pg)



Swimlane Diagrams

- Allows the modeler to represent the flow of activities described by the use-case
- This diagram indicates which actor or analysis class has responsibility for the action described by an activity rectangle

(Fig 8.8 of 225 pg)

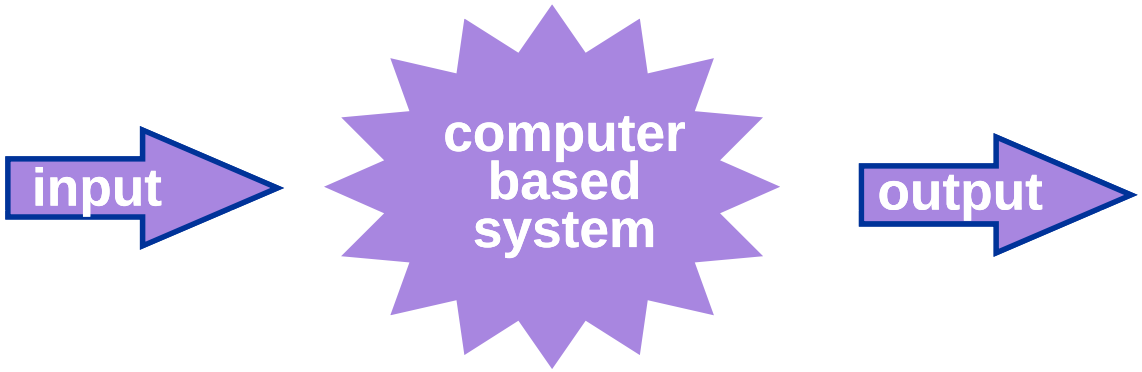


Flow-Oriented Modeling

- Represents how data objects are transformed at 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 (1/2)

external entity

A producer (origin) or consumer (sink) 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

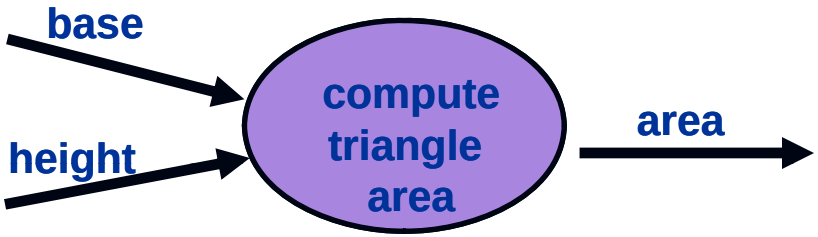
A data transformer (changes input to output)
Examples: compute taxes, determine area, format report, display graph
Data must always be processed in some way to achieve system function

KAIST

Flow Modeling Notation (2/2)

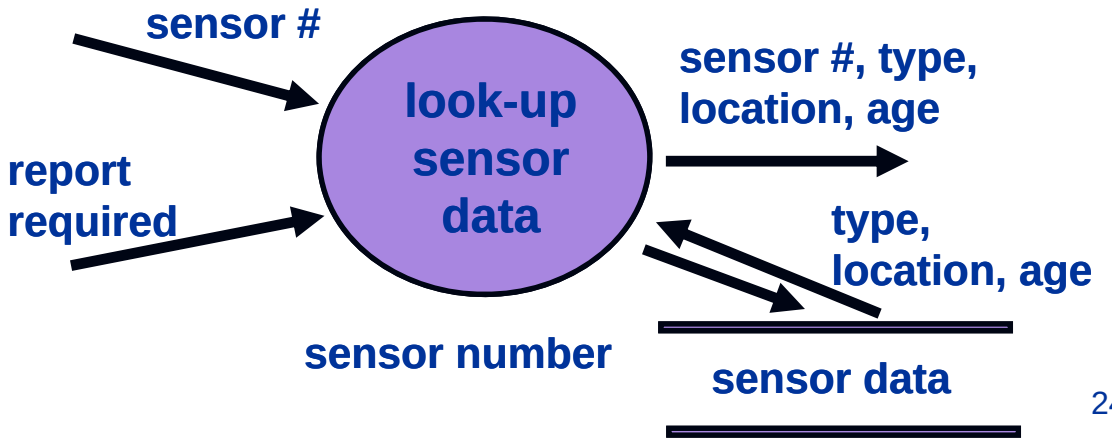
Data flow

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



Data store

Data is often stored for later use.



Data Flow Diagramming: Guidelines

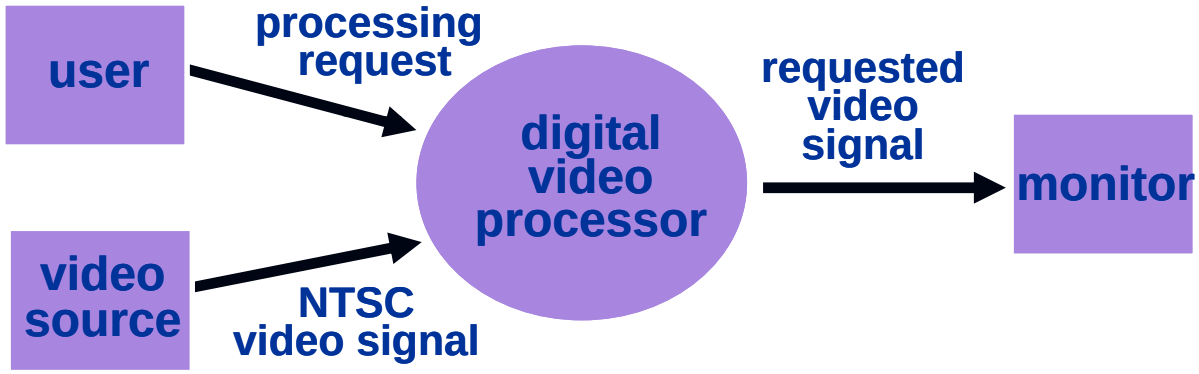
- All icons must be labeled with meaningful names
- The DFD evolves through a number of levels of detail
- Always begin with a context level diagram (also called level 0)
  - Top-down approach
- Always show external entities at level 0
- Always label data flow arrows
- Do not represent procedural logic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

KAIST

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

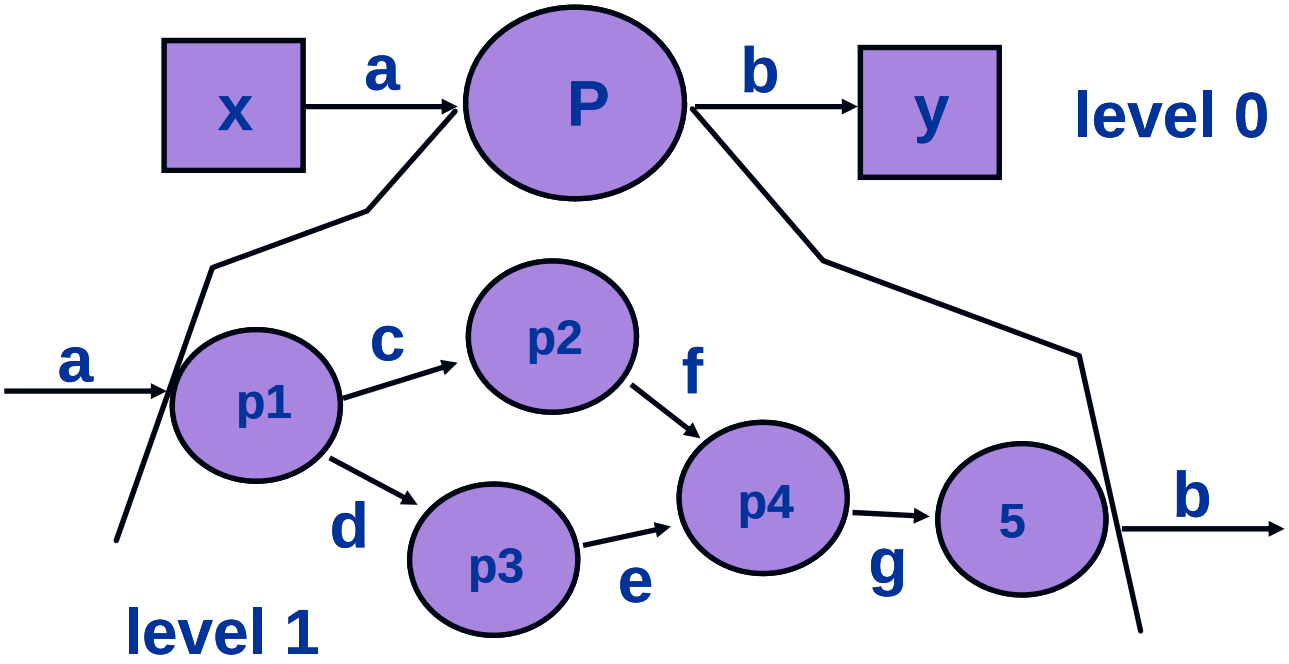
Level 0 DFD Example



Constructing a DFD—II

- 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

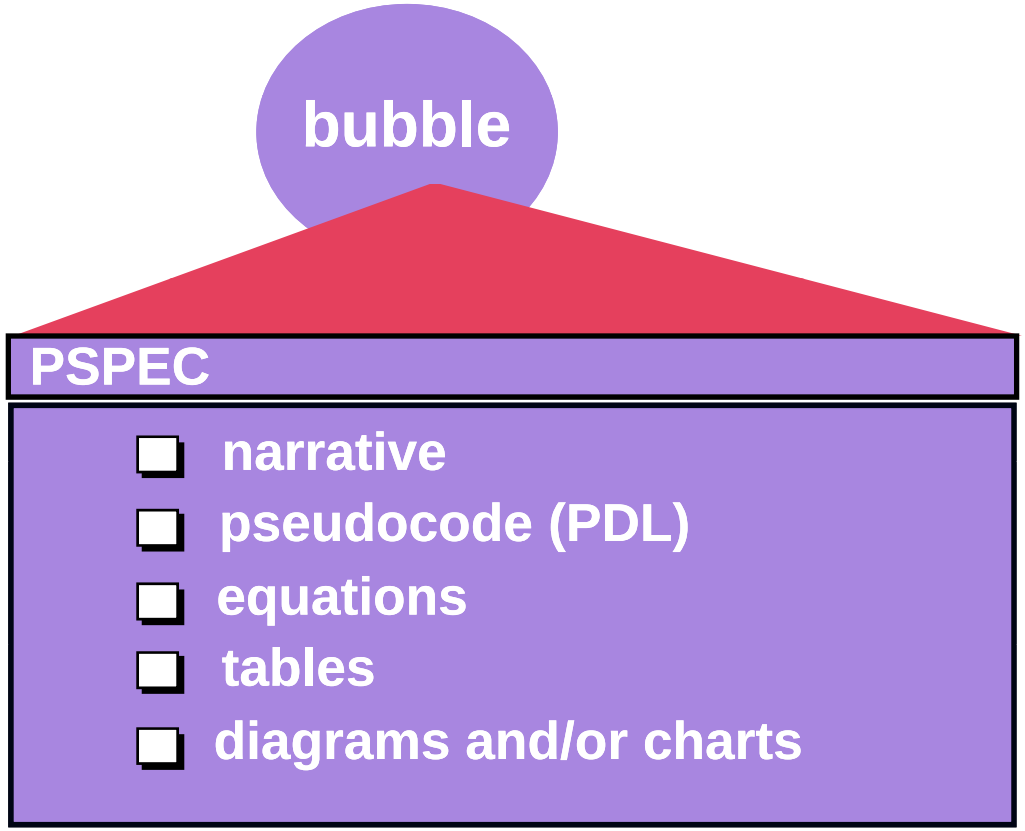


Flow Modeling Notes

- 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)

Appearing at the final level of refinement



Class-Based Modeling

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

Analysis Classes

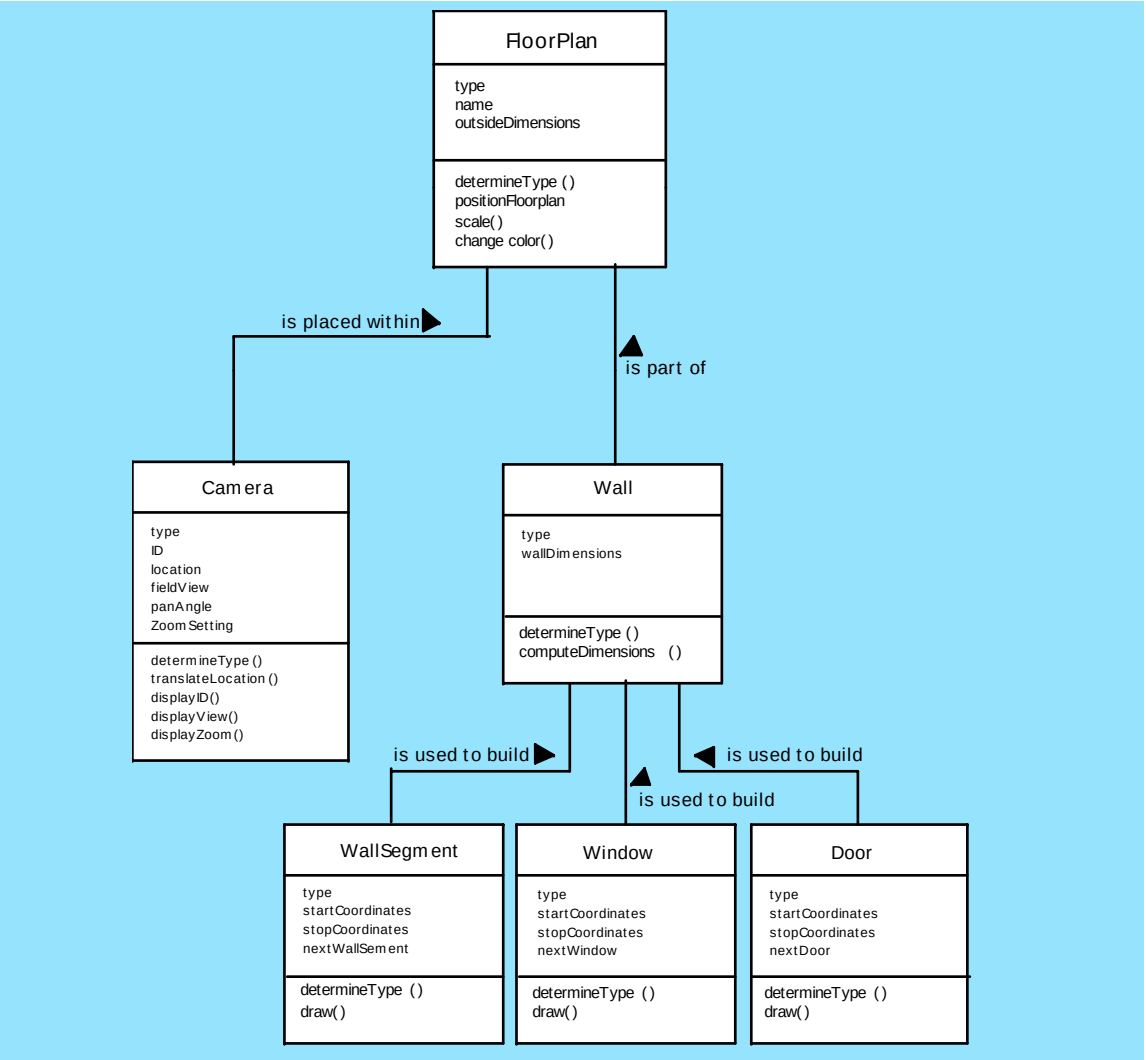
- *External entities* (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
- *Things* (e.g, reports, displays, letters, signals) that are part of the information domain for the problem.
- *Occurrences or events* (e.g., a property transfer or 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

KAIST

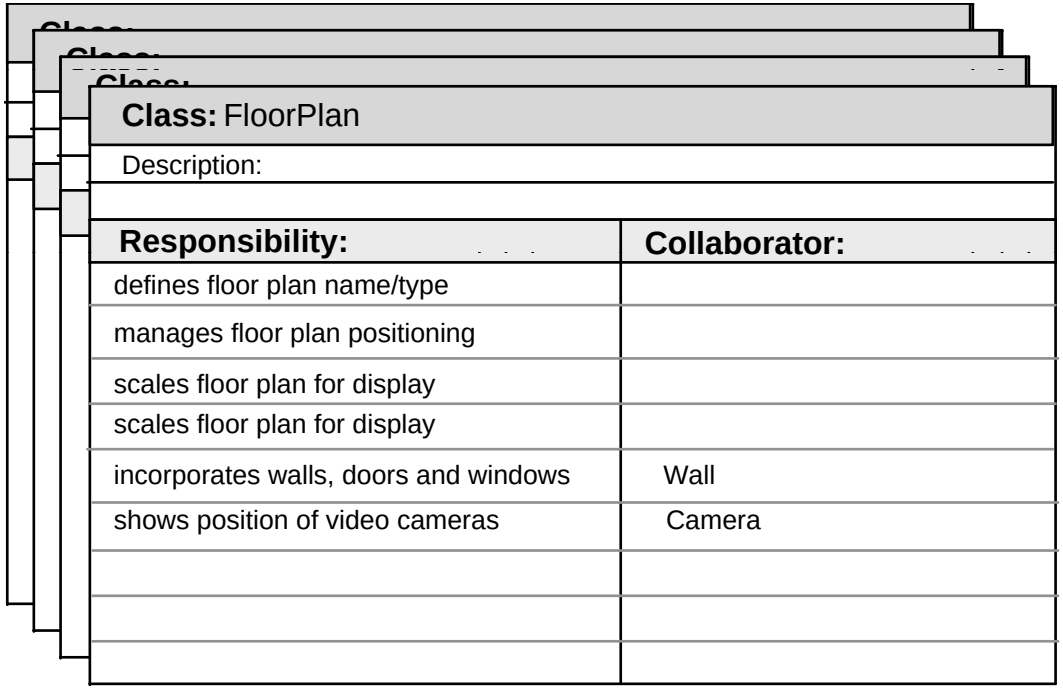
Class Diagram



Class-Responsibility-Collaborator (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



Class: FloorPlan

Description:

Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

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

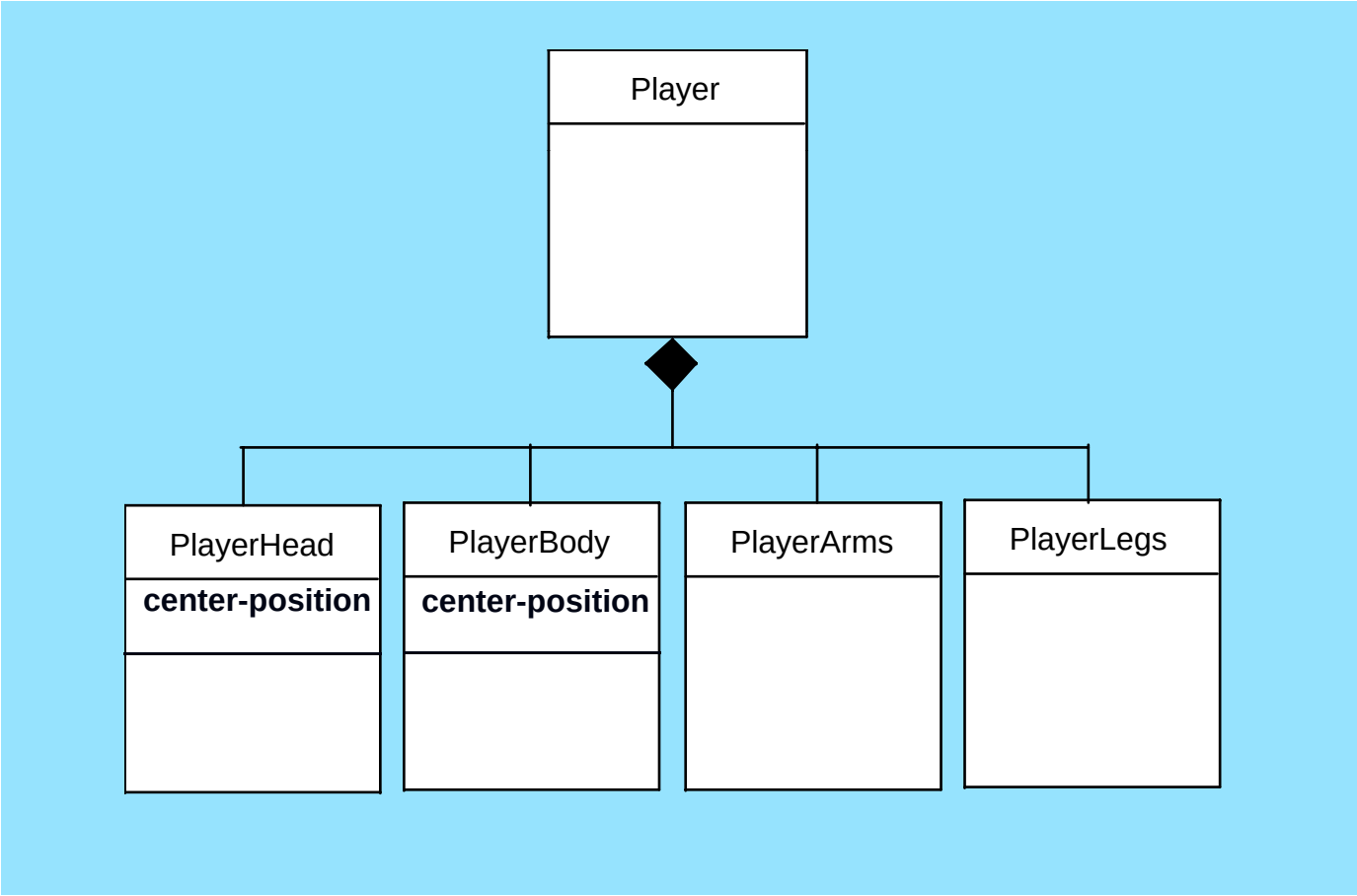
- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible (for higher reuse)
- Information and the behavior related to it should reside within the same class
- Information about one thing should 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

KAIST

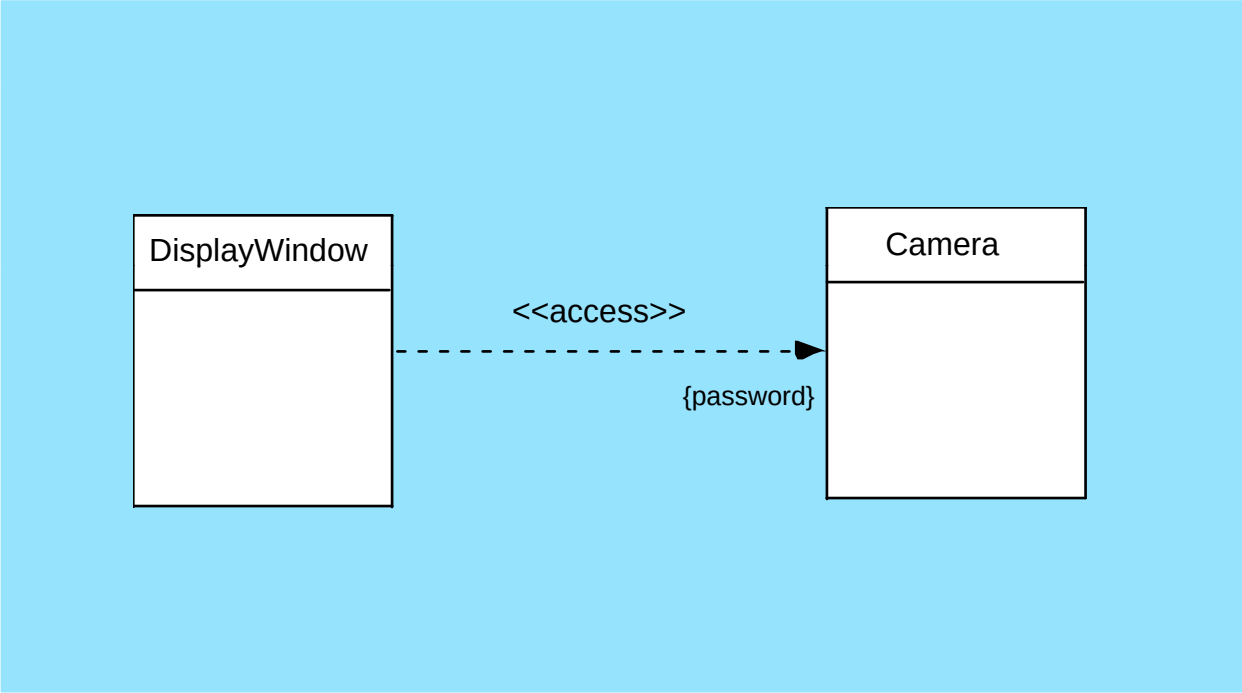
Composite Aggregate Class



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.

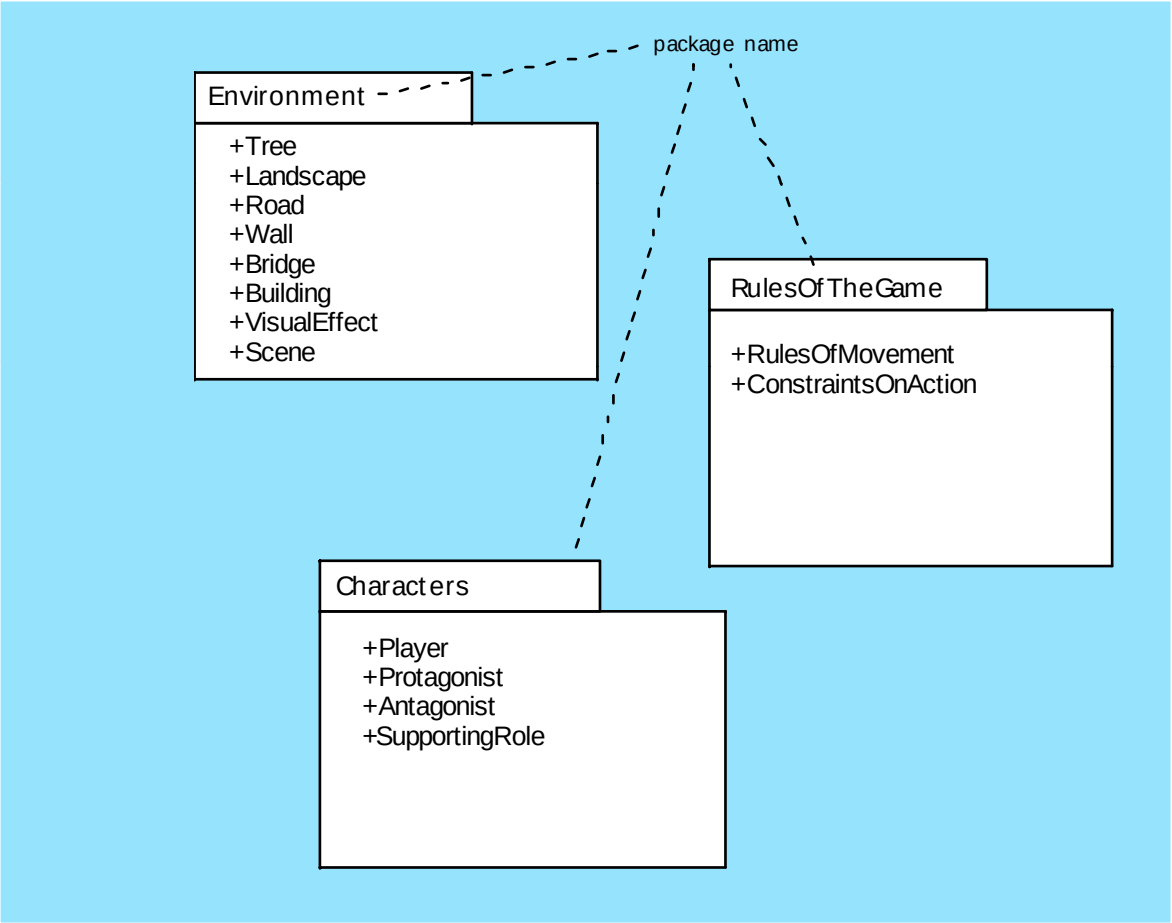
Dependencies



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 - 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



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.

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

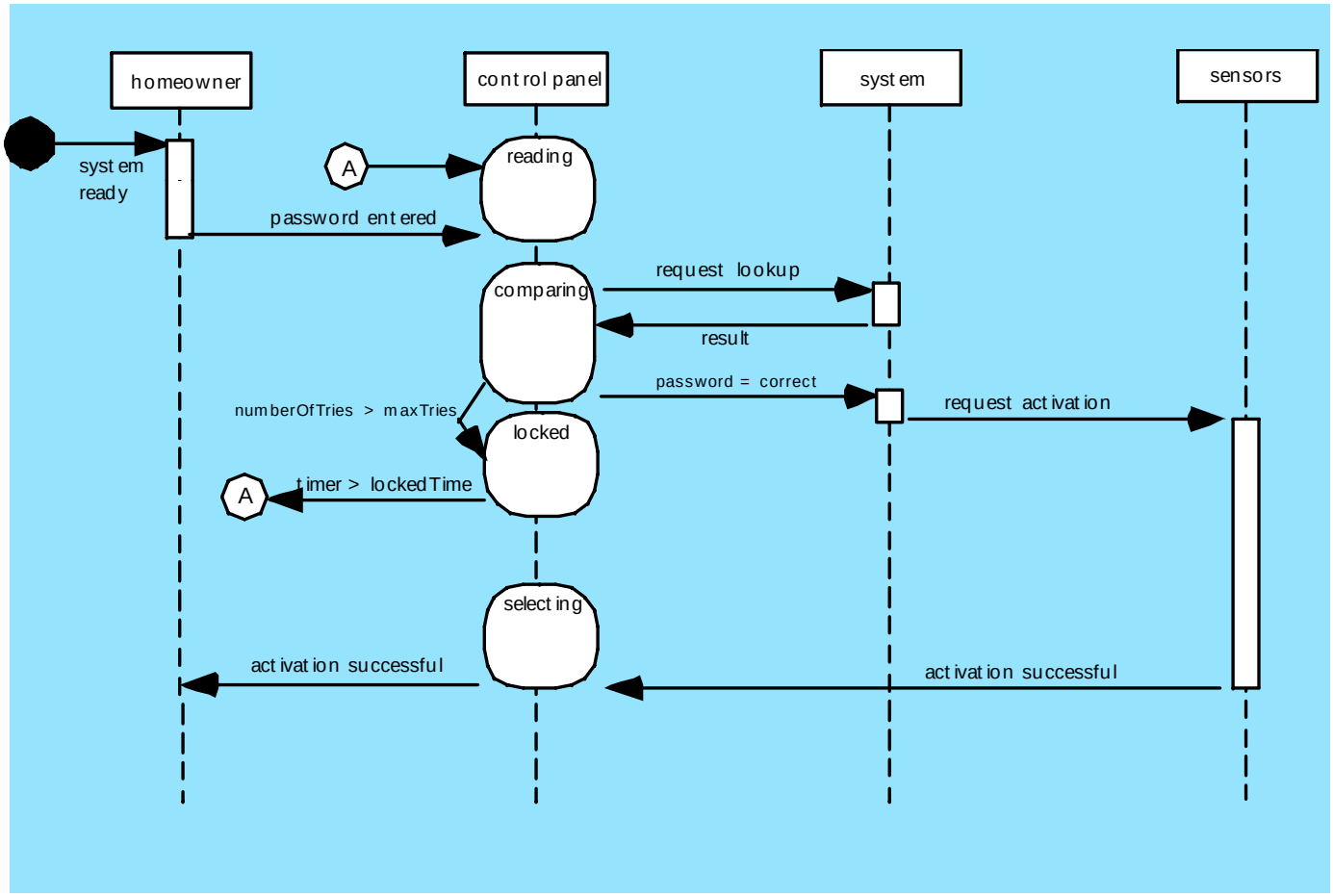


Figure 8.27 Sequence diagram (partial) for *SafeHome* security function

State Diagram for the ControlPanel Class

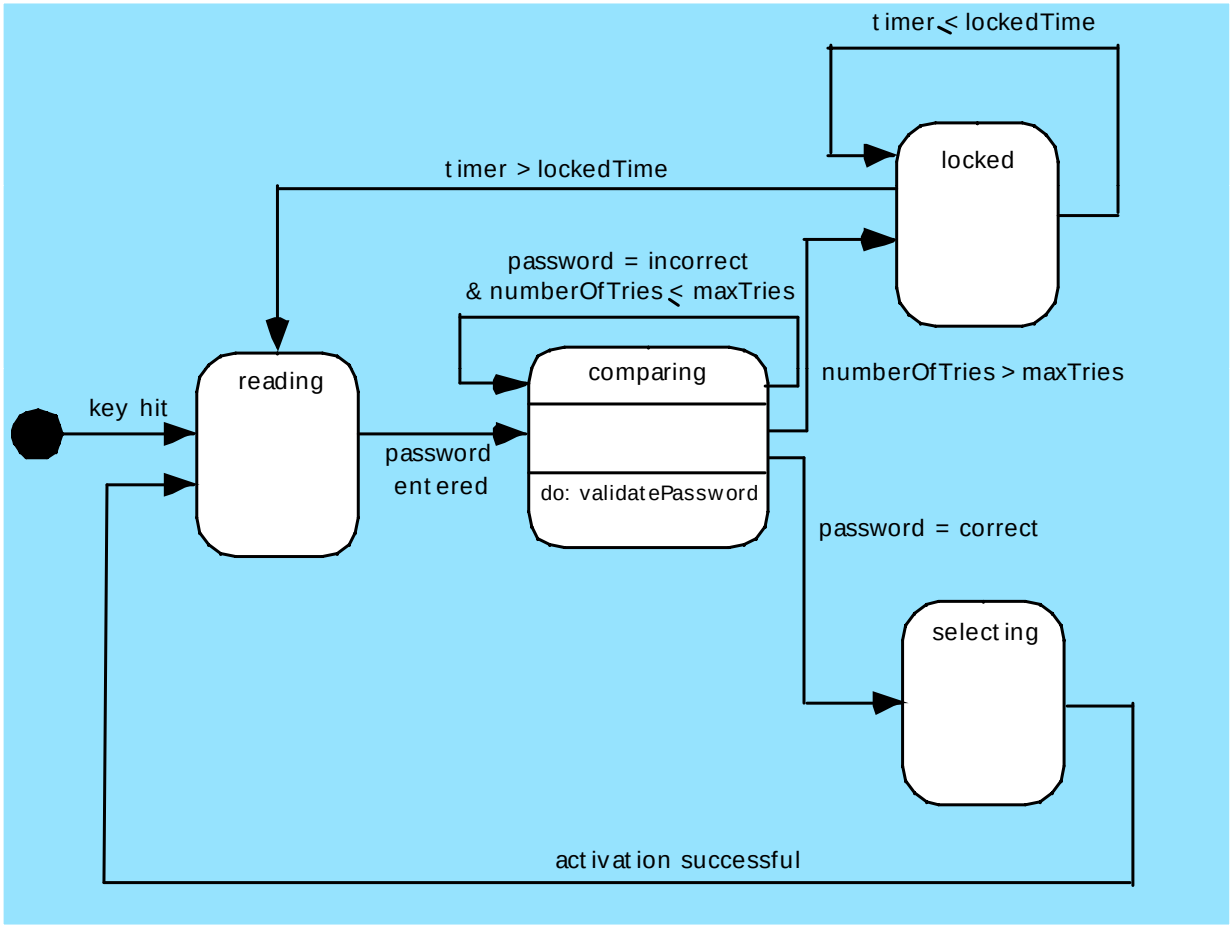


Fig 8.20 pg 251 in SEPA