SOFTWARE ENGINEERING

CHAPTER-6
REQUIREMENTS MODELING: SCENARIOS, INFORMATION, AND ANALYSIS CLASSES

Software Engineering: A Practitioner's Approach, 7th edition

Originated by Roger S. Pressman

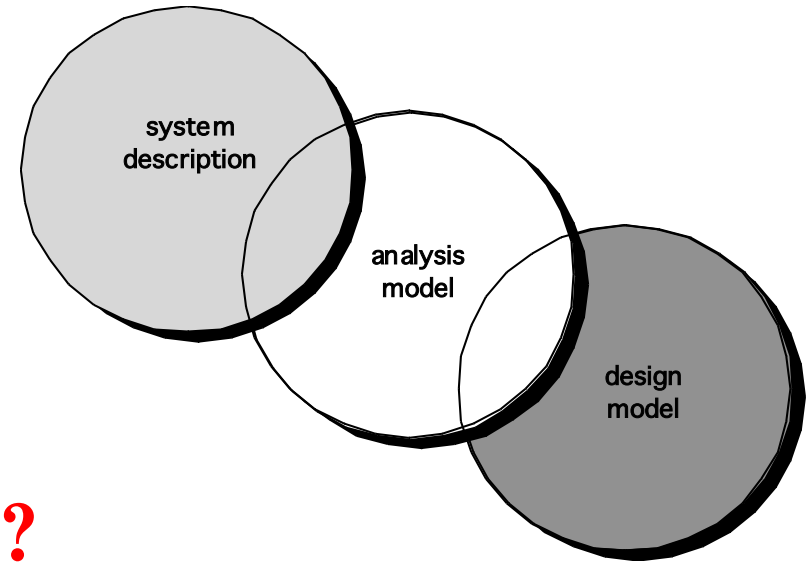
Software School, Fudan University
Spring Semester, 2016

Requirements Analysis

- Requirements analysis
  - specifies software's operational characteristics
  - indicates software's interface with other system elements
  - establishes constraints that software must meet
- Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

A Bridge

- To describe what the customer requires
- And further
  - To establish a basis for the creation of a software design
  - To define a set of requirements that can be validated once the software is built



What? **How?**

Requirement Analysis

- Requirement analysis results in
  - Specification of software's ***operational*** characteristics
  - Software's ***interface*** with other system elements
  - ***Constraints*** that software must meet
- Analysis model and requirements specification
  - Provide a means for *assessing quality* once software is built

Requirement Analysis: An Example

- POS Software for Supermarket
  - ***Operational characteristics***
    - product input, order generation, storage update
    - ……
  - ***Interface***
    - with banking system: support payment by bank cards
    - with human (clerk): support input by both scan and keyboard
    - with hardware (printer): print receipt
    - ……
  - ***Constraints***
    - data precision: error per day < 0.1 (RMB)
    - can run on low-cost clients, use open-source OS and DB
    - ……

Principles in Requirement Analysis (Tom DeMarco)

- The products of analysis (requirement specification) must be highly maintainable
- Problems of size must be dealt with using an effective method of partitioning (decompose and conquer)
- Graphics should be used whenever possible
- Differentiate between logical [essential] and physical [implementation] considerations…
  - E.g. the customer told you "recognize the handwritten names on the answer sheet and …"
    - Essential: automatically recognize the examinee of each answer sheet
    - Implementation: recognize examinee by handwritten names
    - The real best solution: using barcode
  - It is our (developer) work to design and implement the best solution (realistic, most satisfying the requirements) to meet the customer's requirements

Analysis Rule of Thumb (Arlow and Neustadt)

- Focus on requirements that are visible within the problem or business domain, in a relatively high level of abstraction (Don't get bogged down in details)
- Each element should add to an overall understanding of software requirements (No redundancy or ambiguity)
- Delay consideration of infrastructure and other non-functional models until design (Focus on the problem)
- Minimize coupling throughout the system (The more coupling, the more complex)
- Should provide value to all stakeholders (Make the analysis model useful for business, designer and QA)
- KIS – Keep the model as simple as it can be

Domain Analysis

- The same analysis patterns often reoccur across many applications with a specific business domain (commonality)
  - E.g. in e-business domain, "online payment" is a common requirement involved in many different applications
- Analysis *for* reuse in multiple projects
  - Not only the analysis model (or segment) itself
  - But also corresponding design and implementation
    - Design patterns
    - Executable software components

Object-Oriented Domain Analysis

- Identification, analysis and specification of common, reusable capabilities
- In terms of common objects, classes, subassemblies and frameworks
- May involve some application-specific variation or extension points
- For example: online payment
  - Common objects/classes: order, bill, bank card…
  - Common process: confirm order and amount, send billing data, input card number and password, return payment results…

Analysis Modeling Approaches

- Structured Analysis (SA)
  - Data
    - Data objects with attributes and relationships
  - Process that transform the data as separate entities
    - How data objects flow through the system
  - Model Representation (Graphics)
    - Entity-Relationship Diagrams (ERD)
    - Data Flow Diagrams (DFD)
- Object-oriented analysis (OOA)
  - Classes: unified encapsulation of data and behaviors
  - Collaboration between classes
  - Model Representation (Graphics) : UML Diagrams

Data Modeling

- Analysis modeling often begins with data modeling
- Defines all data objects processed in the system, the relationships and other pertinent information
- Data object: a representation of any composite information that must be understood by the software
  - external entity, thing, an occurrence or event, a role, an organization unit, a place or a structure…

Naming Attributes

Identifier | Descriptive Attributes | Referential Attributes

Instance

Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

Data Objects Vs. OO Classes

- Data Object
  - Defines a composite data item
- OO Class
  - Encapsulates both data attributes and operations
  - Definition of classes implies a comprehensive infrastructure that is part of the object-oriented software engineering approach
    - Communicate with one another via messages
    - Can be organized into hierarchies and provide inheritance characteristics for objects

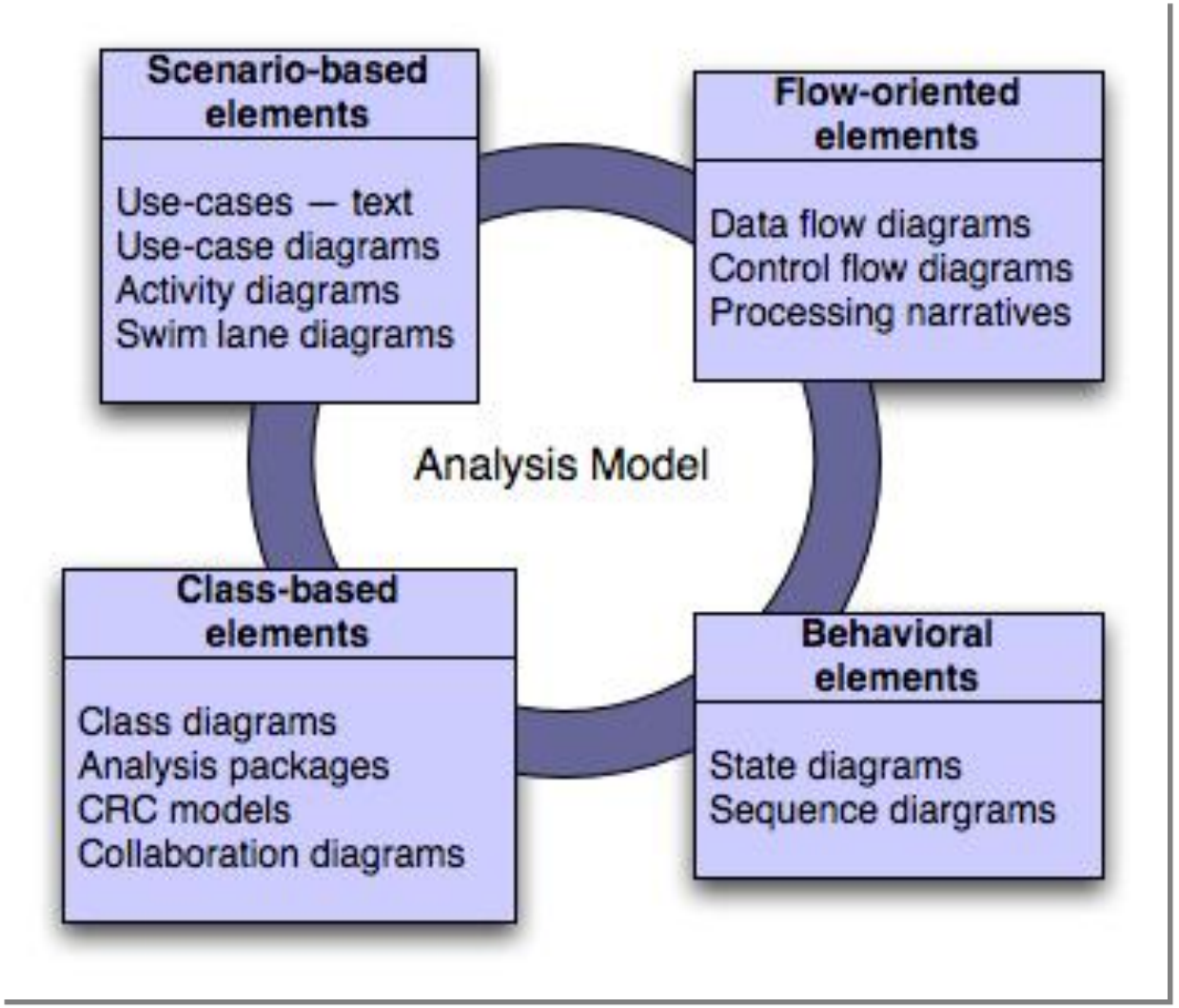
Object-Oriented Analysis (OOA)

- Intent: define all classes (and the relationships and behaviors) that are relevant to the problems
  - Basic user requirements communicated between customer and software engineer
  - Classes are identified (attributes and methods)
  - Class hierarchy is defined
  - Object-to-object relationships be represented
  - Object behaviors be modeled
  - The above tasks are reapplied iteratively until the model is complete

Object-Oriented Concepts

- Attribute
- Class
- Objects
- Operations (method or service)
- Subclass/Superclass
- Aggregation

Elements of the Analysis Model (Object-Oriented)



Requirement models are multi-dimensional
Different kinds of diagrams should be combined to provide a complete representation

Scenario-Based Modeling

USE CASE (用况)

- A use-case captures the interactions that occur between producers and consumers of information and the system itself
  - E.g. a POS sale use case describes the interactions among clerk, client and the POS system
- A use-case describes a specific usage scenario in straightforward language from the point of view of a defined actor
  - *actors* represent roles people or devices play as the system functions
  - *users* can play a number of different roles for a given scenario

Why Scenario? An Example

- Problems: House Design
  - Scenarios
    - Hosting Guests
    - Cooking
    - Bathroom using in the morning
    - ……
  - A series of requirements can be derived from these scenarios
    - The structure of the house
    - The path connecting kitchen and living room
    - How many bathrooms needed and their position
    - ……

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?

What to Write About?

- Inception and elicitation—provide you with the information you'll need to begin writing use cases.
- Requirements gathering meetings, QFD, and other requirements engineering mechanisms are used to
  - identify stakeholders
  - define the scope of the problem
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.

How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

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?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Use Case and Scenario

The scenarios of a use case can be thought as its instances

- **Primary (main) scenarios** of a use-case describe a sequence of actions where everything goes according to plan

- **Secondary (alternative) scenarios** of a use-case deal with exceptional circumstances where things do NOT go according to plan

USE-CASES (CONT.)

- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor(s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What exceptions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

Refining Use Cases: Identify Alternative Interactions

- A description of alternative interactions is essential for a complete understanding of the use cases
- Identify alternative interactions by asking the following questions about each step in the primary scenario
  - *Can the actor take some other action at this point?*
  - *Is it possible that the actor will encounter some error condition at this point?*
  - *Is it possible that the actor will encounter some other behavior at this point?*

Use Case Description

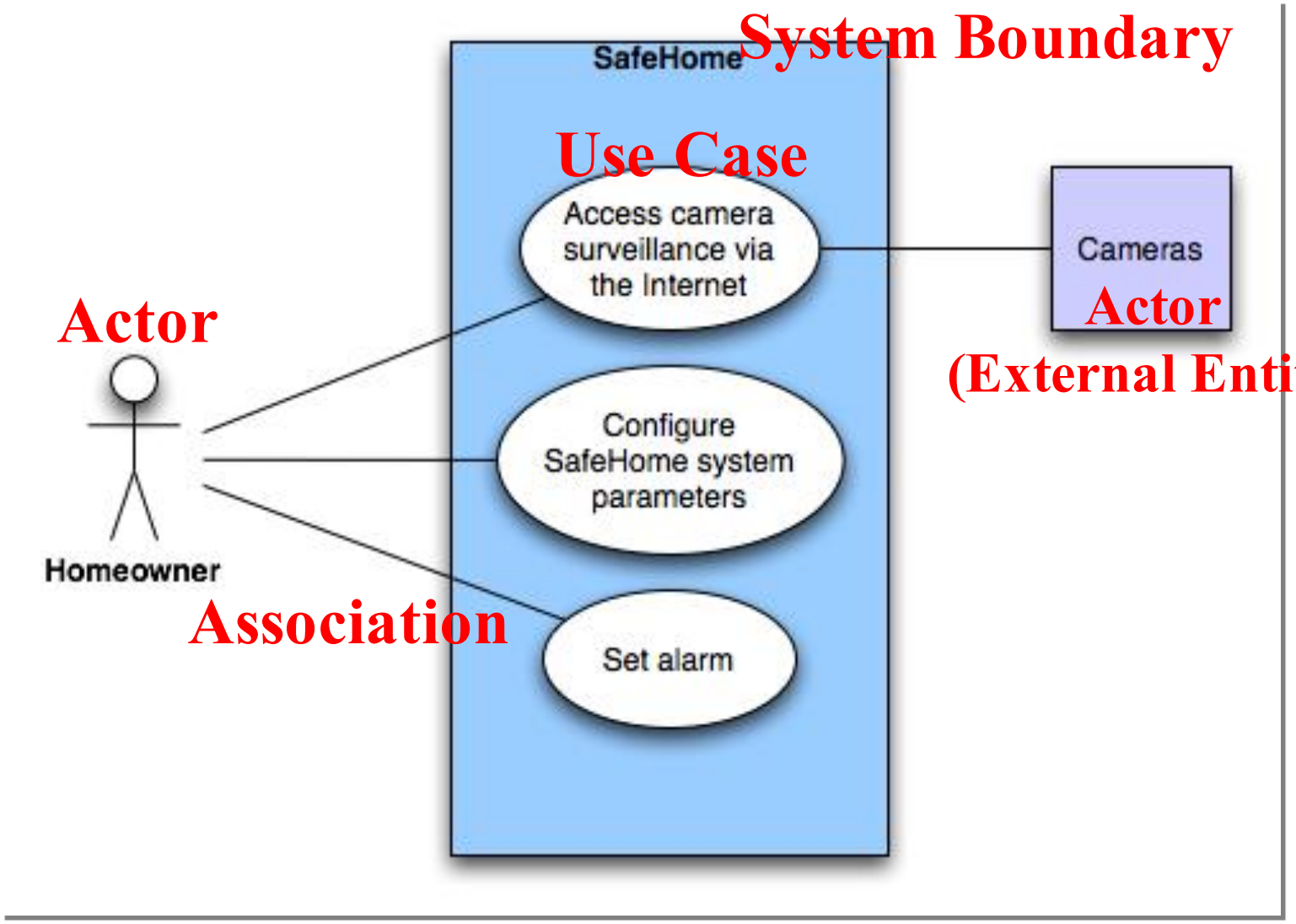
- Text-based Description

- UML Diagrams
  - Use Case Diagrams
  - Activity Diagrams
  - Swim Lane Diagrams
    - variation of activity diagrams

Text-based Use Case Description (Templete)

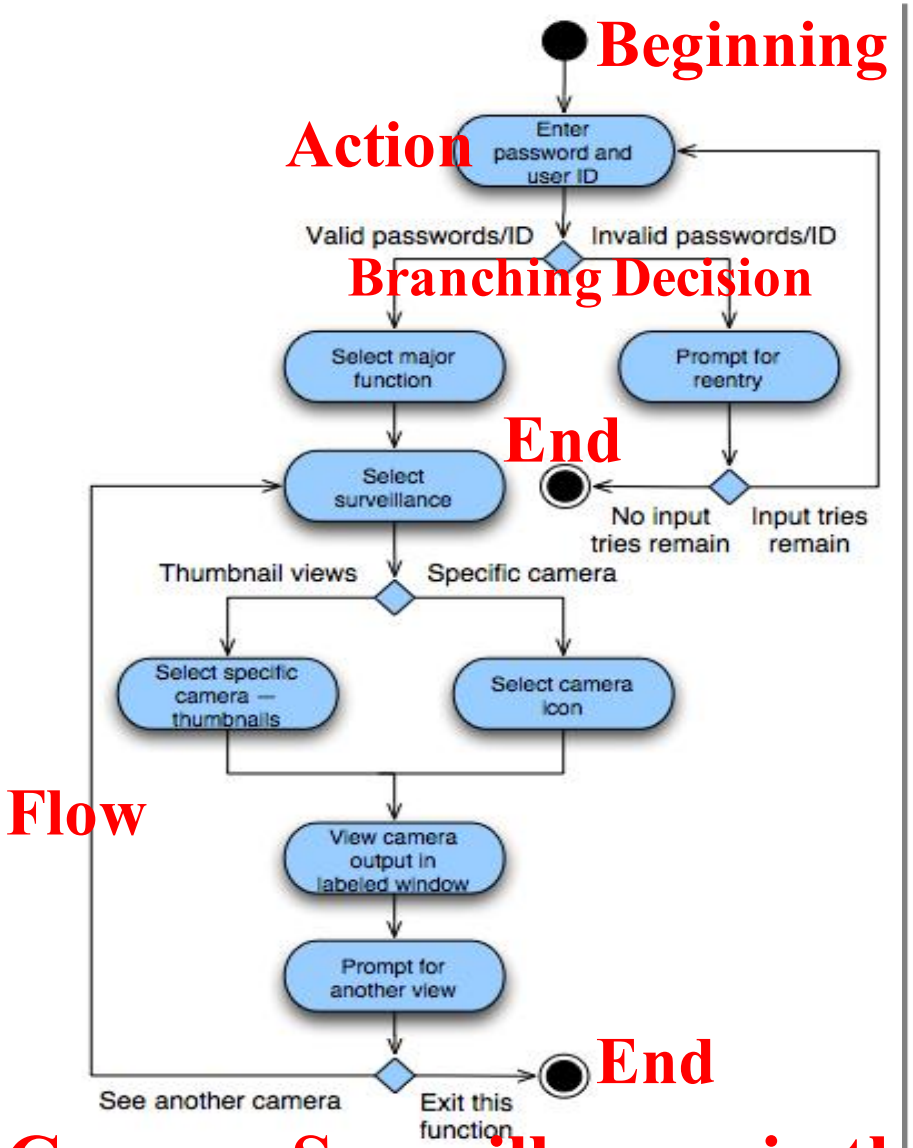
- Use case name
- Primary actor
- Goal in context
- Preconditions
- Trigger
- Main Scenario (a series of steps)
- Exceptions (leading to alternative scenarios)
- Priority
- Frequency of use
- ……

USE-CASE DIAGRAM



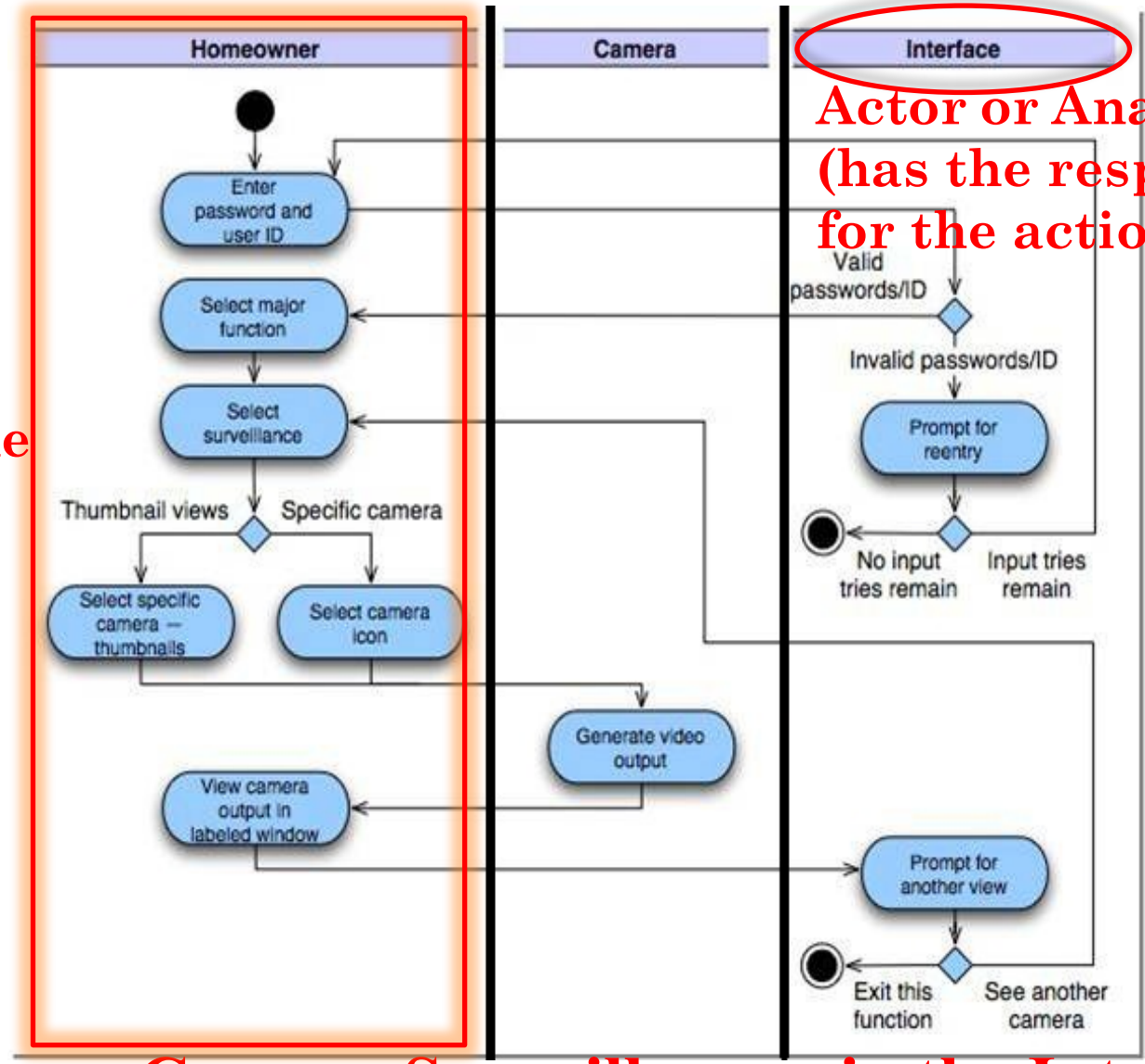
Use-case diagram for surveillance function

Activity Diagram



Access Camera Surveillance via the Internet

Swim Lane Diagram

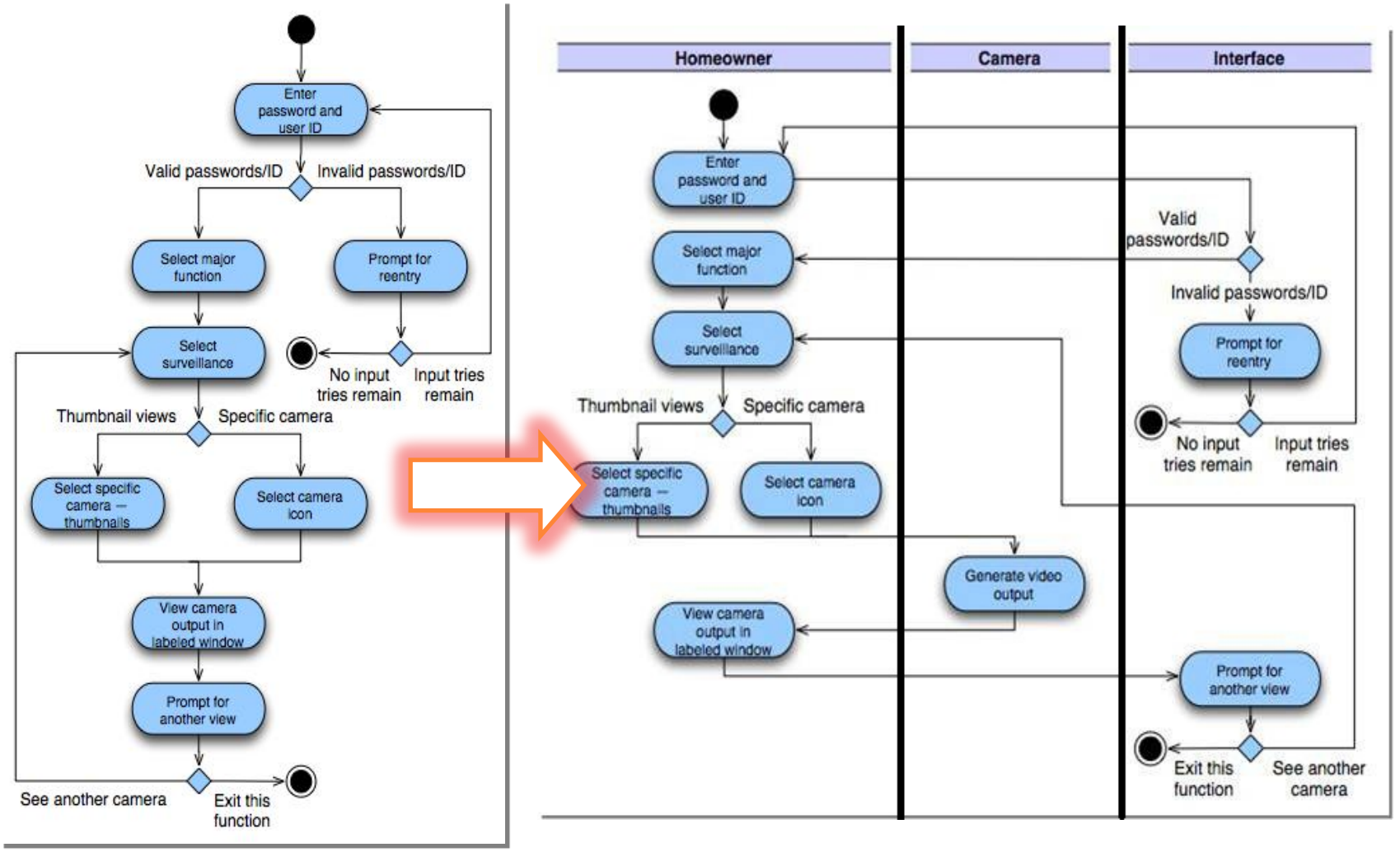


Actor or Analysis Class (has the responsibility for the action)

Swim Lane

Access Camera Surveillance via the Internet

Swim Lane Diagram



Discussion

- Is it proper to model a use case named
  - Input Password
  - Account Management (Check Balance, Modify Password, Check Login Record)
  - Place Order
  - Periodic Intrusion Detection

Primary and Secondary Actors

- A primary actor initiates a use case: the use case starts with an input from the primary actor to which the system has to respond
- Other actors, referred to as secondary actors, can participate in the use case

Actor

- Human actor uses various I/O devices to physically interact with the system.
- External system actor either initiates (as primary actor) or participates (as secondary actor) in the use case.
- Input device actor or an input/output device actor typically interacts with the system via a sensor
- Timer actor that periodically sends timer events to the system

Identify Use Cases

- Start by considering the actors and the interactions they have with the system
- Avoid functional decomposition: small use cases describe small individual functions rather than describe a sequence of events that provides a useful result to the actor
  - For example, in an ATM system, the query and transfer functions should be modeled as separate use cases

Class-Based Modeling

Class-based Modeling

- Identifying analysis classes
- Specifying attributes
- Defining operations
- Class-Responsibility-Collaborator (CRC) Modeling
- Modeling associations and Dependencies
- Analysis Packages

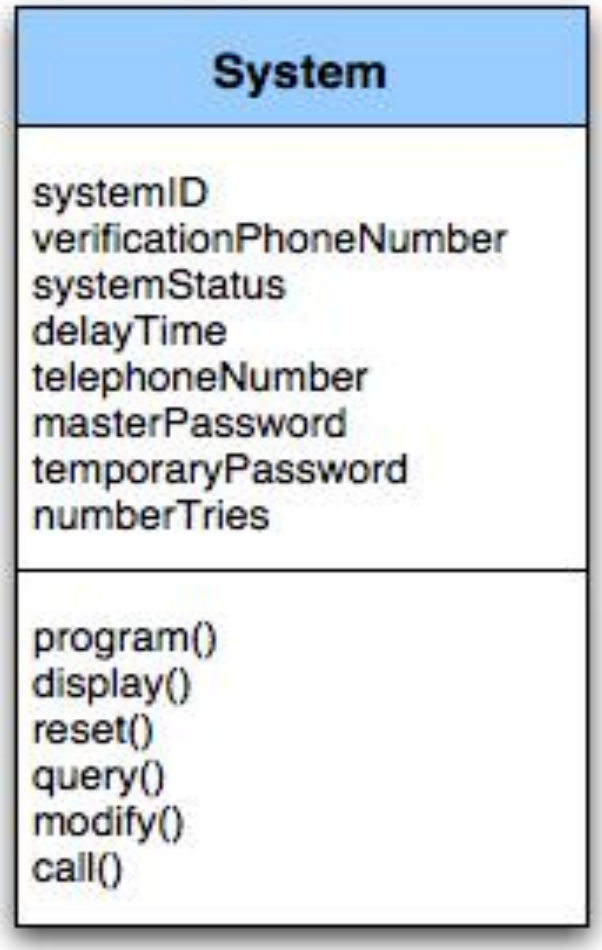
Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse" [Abb83]
  - Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
  - Synonyms should be noted.
  - If the class (noun) is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.
- But what should we look for once all of the nouns have been isolated?

Candidate Analysis Classes

- External entities (physical) that produce or consume information, e.g. *camera*
- Things (information entities) that are part of the information domain, e.g. *receipt, forms*
- Occurrences or events that occur within the context of system operation, e.g. *phone call*
- Roles played by people who interact with the system, e.g. *student*
- Organizational units , e.g. *department*
- Places that establish context, e.g. *classroom*
- Structures that define a class of objects

Class Selection Criteria



1. Retained information
2. Needed services
3. Multiple attributes
4. Common attributes
5. Common operations
6. Essential requirements

* Some decisions may violate some of them for specific purpose.

POTENTIAL CLASSES

- *Retained information.* The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
- *Needed services.* The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
- *Multiple attributes.* During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.
- *Common attributes.* A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
- *Common operations.* A set of operations can be defined for the potential class and these operations apply to all instances of the class.
- *Essential requirements.* External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

Find Candidate Classes by Grammatical Parse

The SafeHome security function enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through the Internet, a PC or a control panel…..

The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface. The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form…

- Nouns as candidate analysis classes
  - homeowner, security information, sensor, control panel…

IDENTIFYING CLASSES

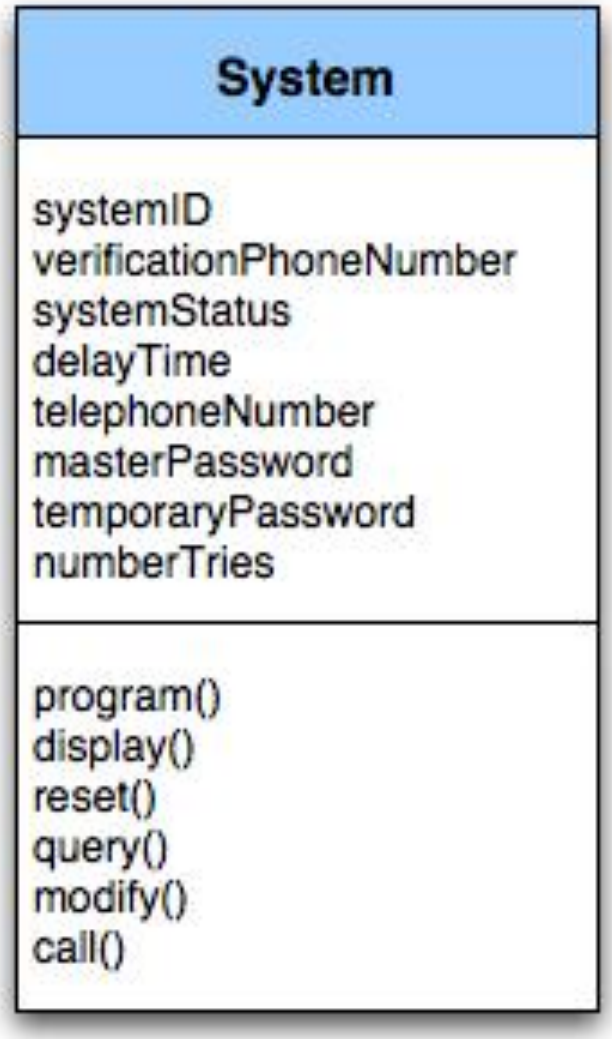
Potential class	Classification	Accept / Reject
homeowner	role; external entity	reject: 1, 2 fail
sensor	external entity	accept
control panel	external entity	accept
installation	occurrence	reject
system (security function)	thing	accept
number, type	not objects, attributes	reject: 3 fails
master password	thing	reject: 3 fails
telephone number	thing	reject: 3 fails
sensor event	occurrence	accept
audible alarm	external entity	accept: 1 fails, others apply
monitoring service	organizational unit; external entity	reject: 1, 2 fail

1. Retained information; 2. Needed services; 3. Multiple attributes
4. Common attributes; 5. Common operations; 6. Essential requirements

Class Types

- Entity classes
  - Business classes
  - Extracted directly from the statement of the problem
- Boundary classes
  - Interfaces (e.g. interactive screen or printed reports)
  - Designed to represent entity objects to users
- Controller classes
  - Manage a "unit of work" from start to finish
  - Design issues, not considered in analysis modeling

Defining Attributes

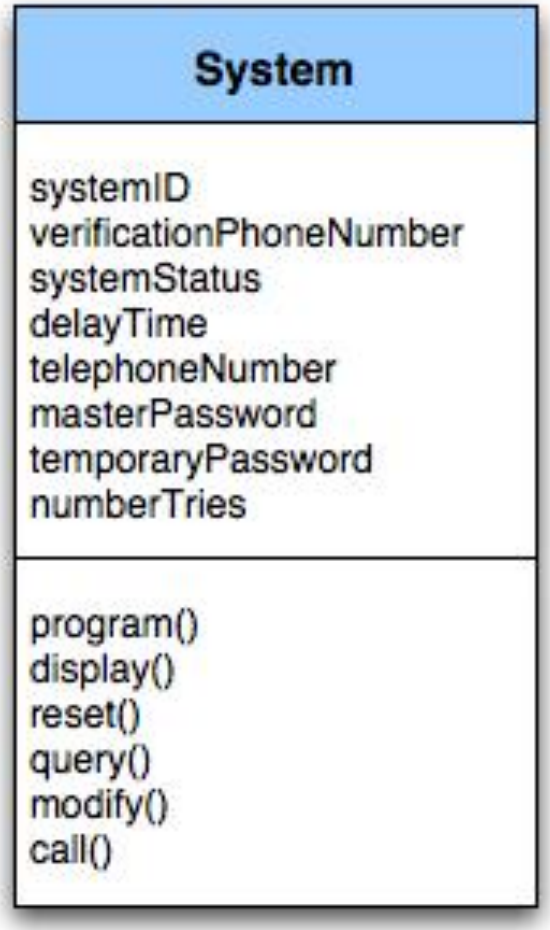


- To be meaningful, selected classes should have
  - **Attributes** to describe what is meant by the class
    - Can select "those" things in a use case that reasonably "belong" to the class
    - Answer the question: what data items (composite / elementary) fully define the class in the context of the problem at hand?
    - E.g. attributes of ***System*** class: identification information; alarm response information; activation / deactivation information

Defining Attributes (Cont.)

- *Attributes* describe a class that has been selected for inclusion in the analysis model
  - build two different classes for professional baseball players
    - **For Playing Statistics Software:** name, position, batting average, fielding percentage, years played, and games played might be relevant
    - **For Pension Fund Software:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

Defining Operations



- To be meaningful, selected potential classes should have
  - **Operations** to define possible behavior of an instance of the class
    - manipulate (add, delete, modify, reformat…) data in some way
    - perform a computation
    - inquire about the state of an object
    - monitor an object for the occurrence of a controlling event

An operation must have "knowledge" of the nature of the class's attributes and associations

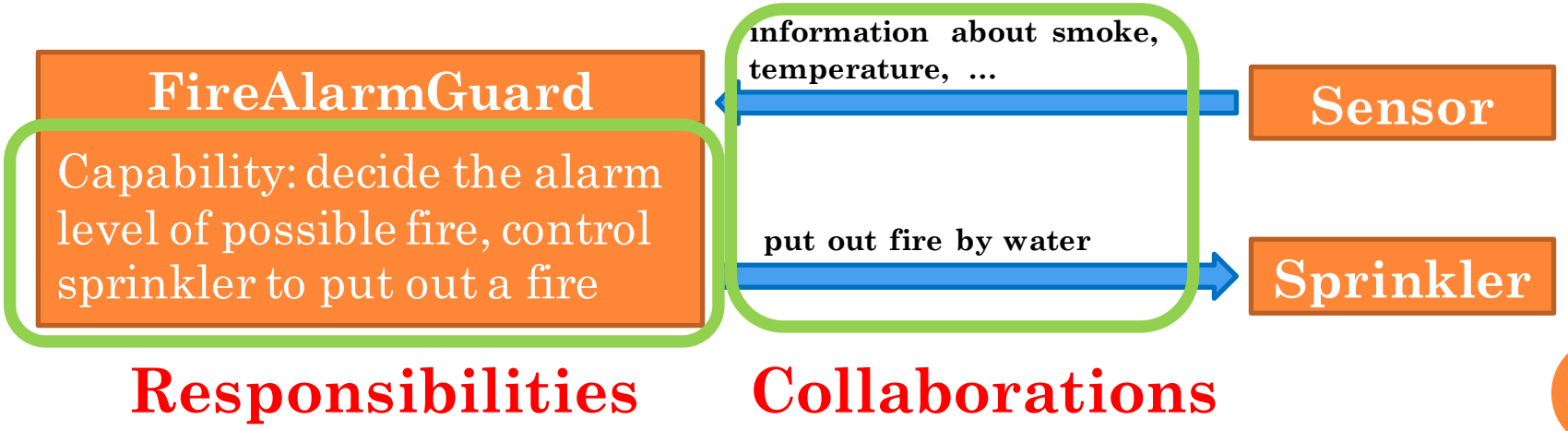
Discussion: Identify Class Attributes and Operations

ShoppingCart Class in a B2C E-Business System

- Attributes
  - customer, productItems, amount, ...
- Operations
  - manipulate data
    - add/remove/udpate productItems, ...
  - Computation
    - computeAmount, ...
  - inquire state
    - checkIfPaid, ...
  - monitor controlling event
    - monitorPayConfirm, ...

Class Responsibilities and Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - collaborate with other classes



CRC Models

- *Class-responsibility-collaborator (CRC) modeling* [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.
- Ambler [Amb95] describes CRC modeling in the following way:
  - A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the collaborators on the right.

CRC Modeling

Class: FloorPlan	
Description	
Responsibility	**Collaborator**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Incorporates walls, doors, windows	Wall
Shows position of video cameras	Camera

A CRC model index card for *FloorPlan* class

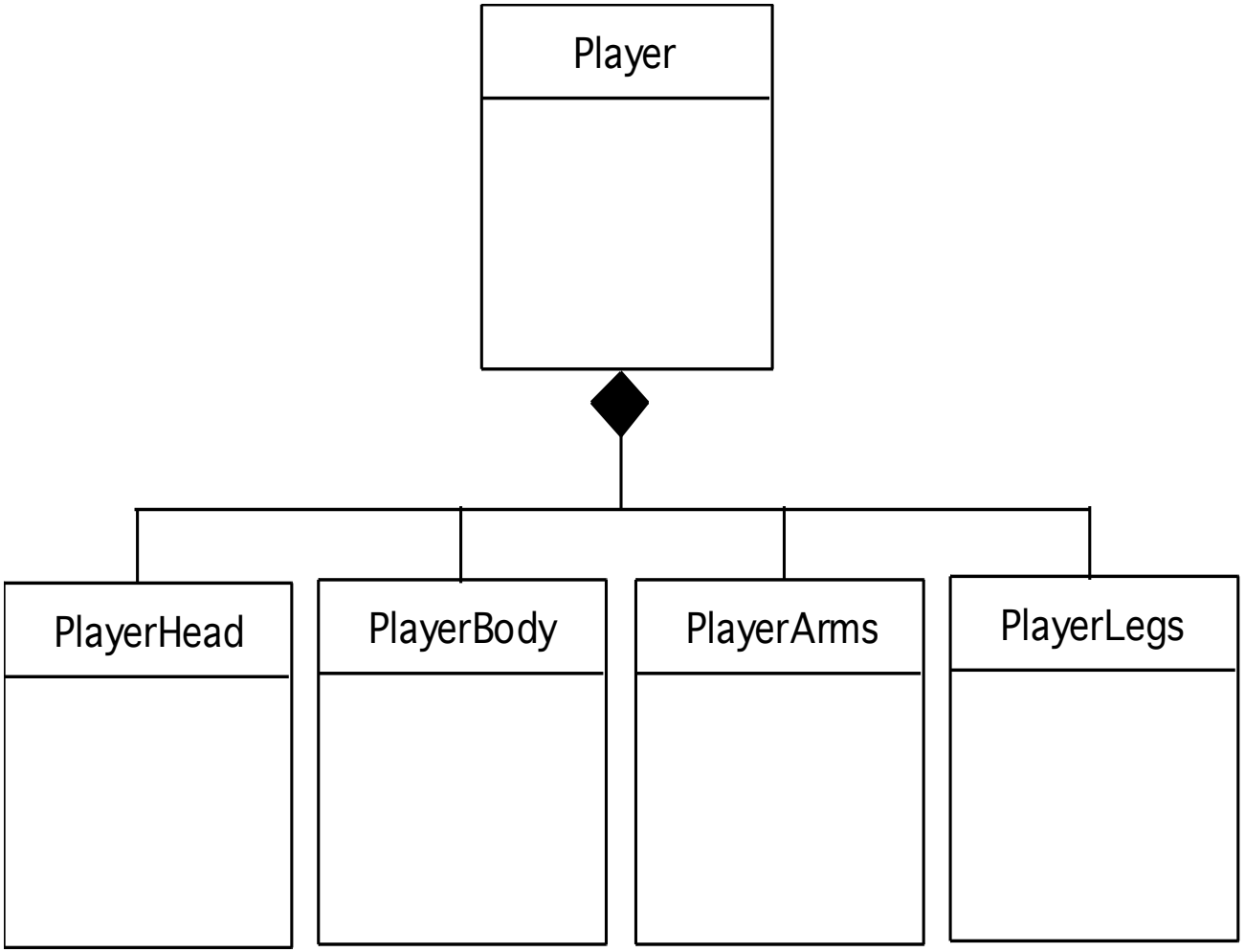
Class Responsibilities

- Distribute system intelligence across classes properly
- State each responsibility as generally as possible
- Put information and the behavior related to it in the same class
- Localize information about one thing rather than distributing it across multiple classes
- Share responsibilities among related classes, when appropriate

Class Collaborations

- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- Relationships between classes
  - is-part-of — used when classes are part of an aggregate class
  - has-knowledge-of — used when one class must acquire information from another class
  - depends-on — used in all other cases

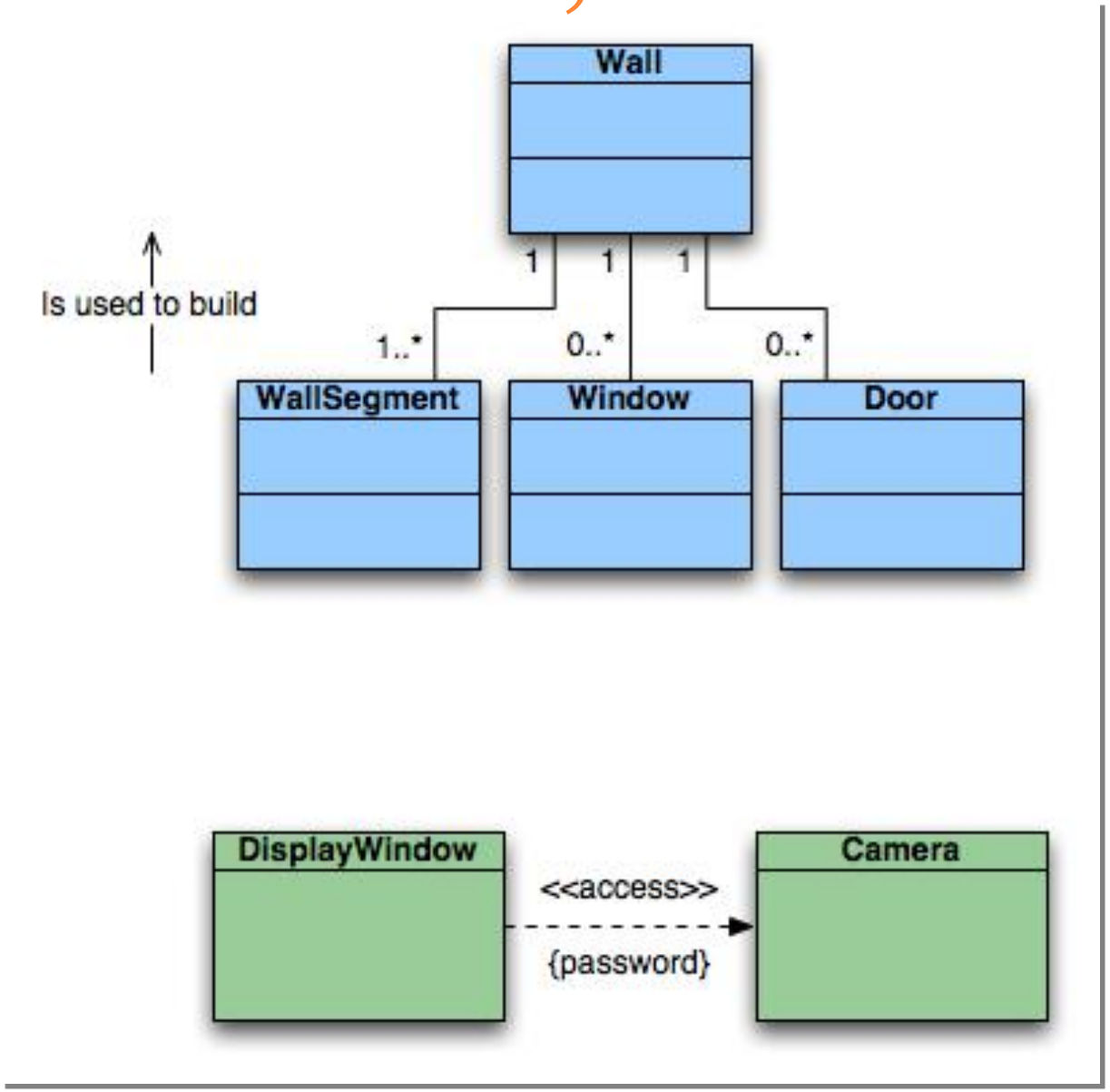
Composite Aggregate Class



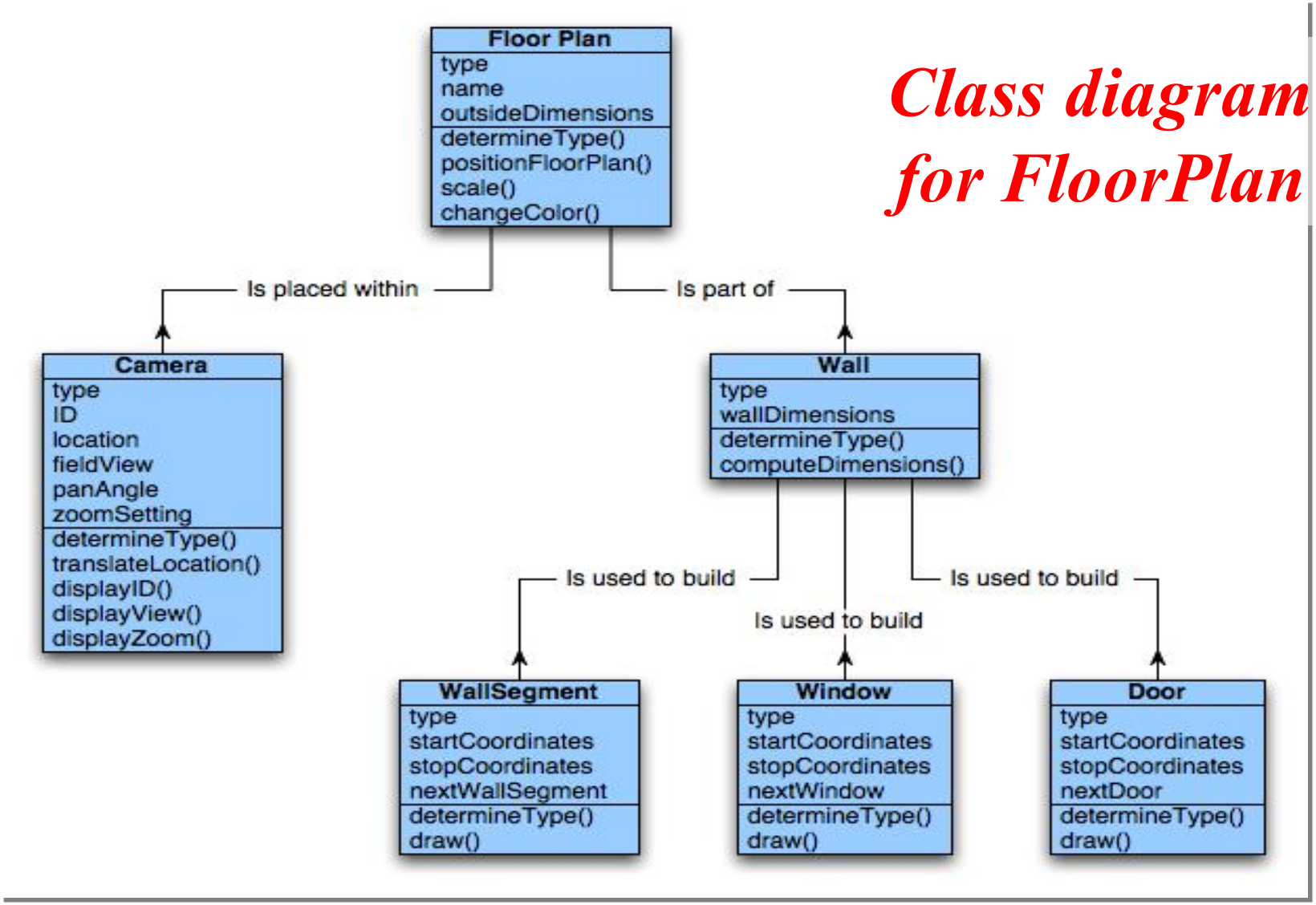
Associations and Dependencies

- 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

Class Diagrams – Multiplicity, Dependency



Class Diagram in Analysis Model

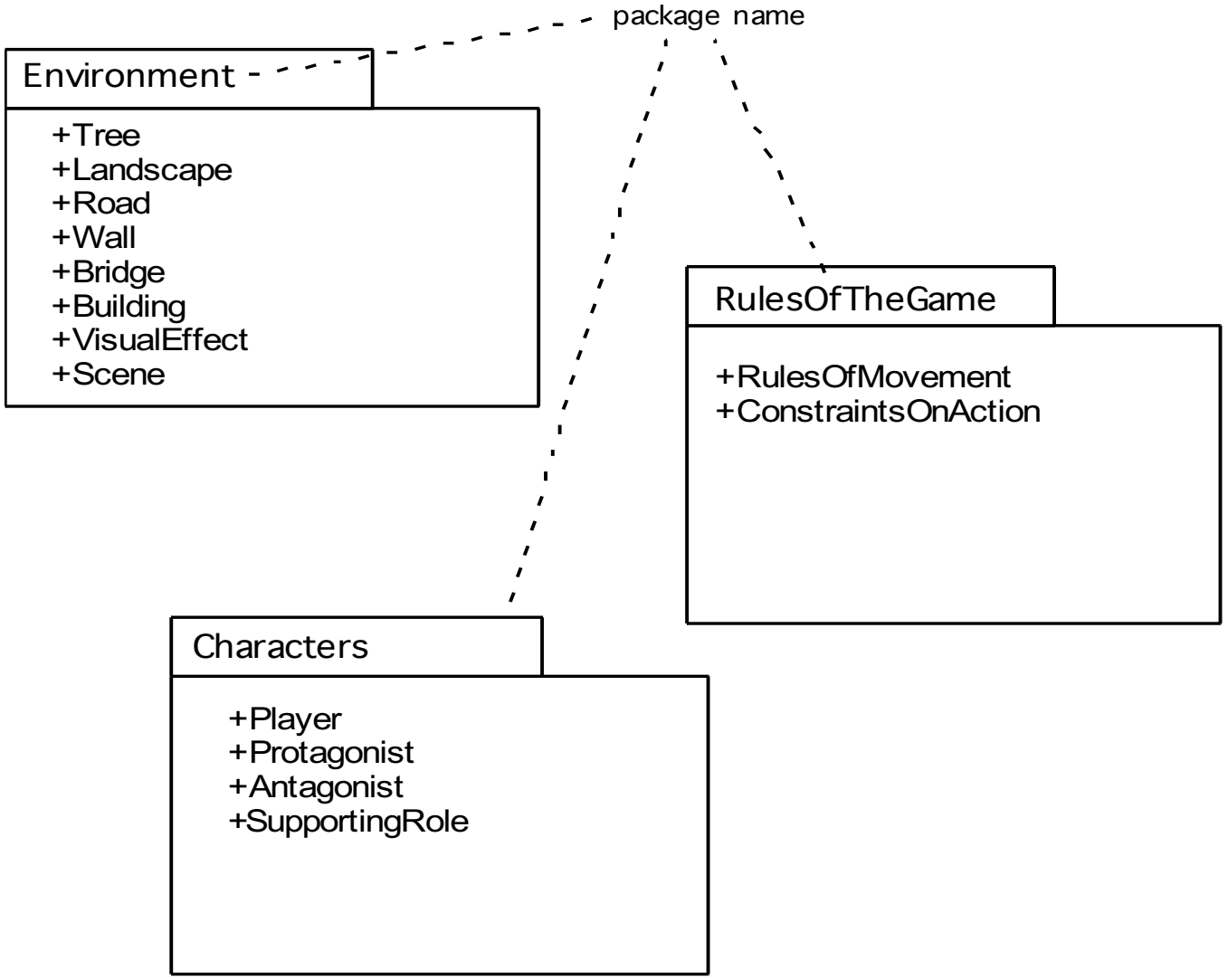


Class diagram for FloorPlan

Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a group
- The plus 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



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

CRC分析练习：POS系统

- **针对以下场景，按照CRC分析开发一组分析类及其职责和协作**

- 顾客推着购物车来到收银台前，递上会员卡
- 收银员扫描会员卡获取顾客信息后问好（如“xx先生，欢迎光临”）
- 收银员逐一扫描每件商品，系统获取每件商品的名称、规格、单价等信息后累加生成购物单
- 系统生成并显示购物单总金额，读取顾客会员信息，并根据超市当前优惠活动规则以及顾客信息（会员等级、累计消费额等）计算折扣，然后显示折扣后的付款金额
- 顾客递上信用卡，收银员刷卡后确认购物单已支付
- 系统记录此次购物信息（时间、收银台编号、收银员、顾客、金额、折扣、明细等），然后更新相应商品的库存信息

End of Chapter 6