Chapter 8
Analysis Modeling

本章要点

- 领域分析
- 数据建模
- OO概念
- 分析建模
  - Scenario-Based Modeling
  - Flow-Oriented Modeling
  - Class-Based Modeling
  - Behavioral Modeling

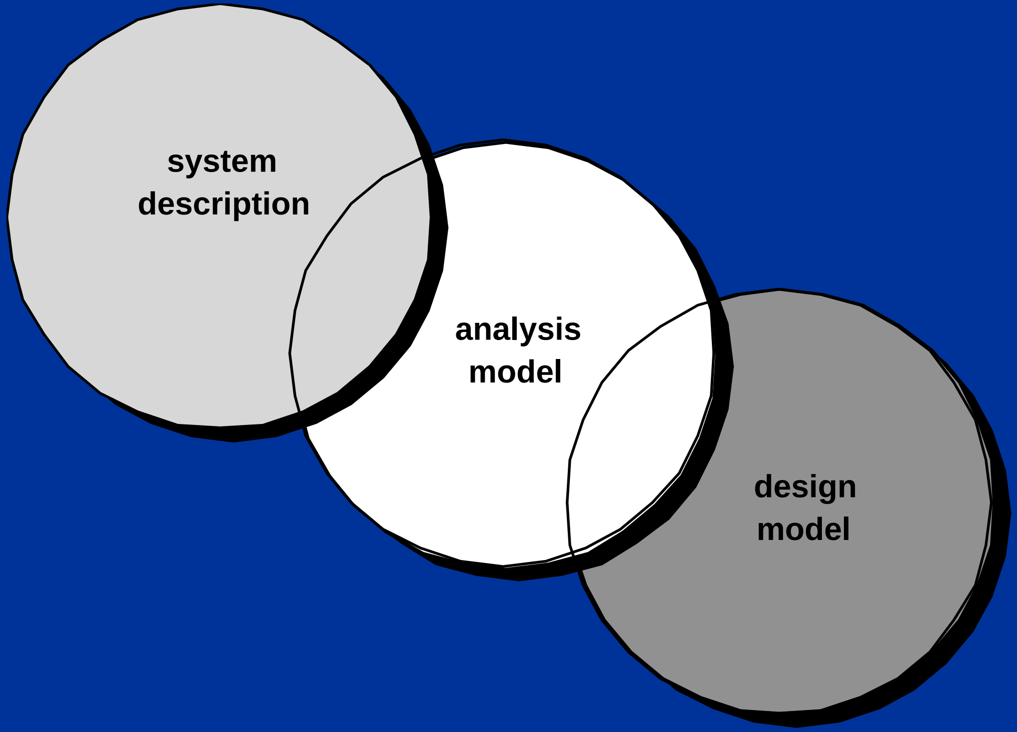
概述

- 使用很多不同格式的图表为信息、功能和行为需求建模。
- 基于场景的建模从用户的角度表现系统；
- 面向流的建模在说明数据对象如何通过处理函数进行转换方面提供了指示；
- 基于类的建模定义了对象、属性和关系；行为建模描述了系统状态、类和事件在这些类上的影响。
- 一旦创建了模型的雏型，就将不断改进，并分析评估清晰性、完整性和一致性。最终的分析模型将由所有的共同利益者确认。

8.1 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



Rules of Thumb[经验原则]

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

Domain Analysis

- 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.

8.2 Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer’ s level of abstraction
- indicates how data objects relate to one another

What is a Data Object?

Object—something that is described by a set of attributes (data items) and that will be manipulated within the software (system)

- each instance of an object (e.g., a book) can be identified uniquely (e.g., ISBN #)
- each plays a necessary role in the system i.e., the system could not function without access to instances of the object
- each is described by attributes that are themselves data items

Typical Objects

- *external entities* (printer, user, sensor)
- *things* (e.g, reports, displays, signals)
- *occurrences or events* (e.g., interrupt, alarm)
- *roles* (e.g., manager, engineer, salesperson)
- *organizational units* (e.g., division, team)
- *places* (e.g., manufacturing floor)
- *structures* (e.g., employee record)

Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

What is a Relationship?

relationship—indicates “connectedness”; a ″fact″ that must be ″remembered″ by the system

- several instances of a relationship can exist
- objects can be related in many different ways

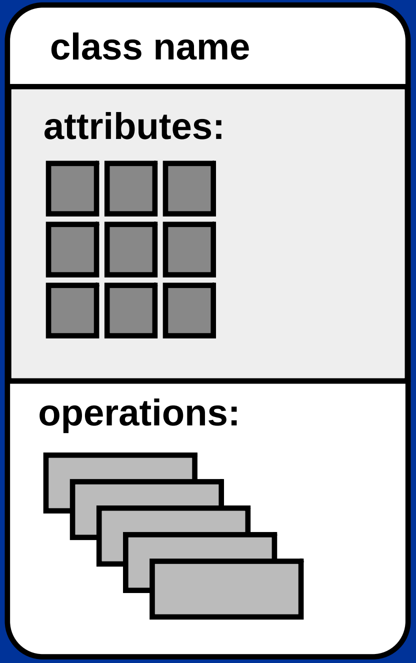
8.3 Object-Oriented Concepts

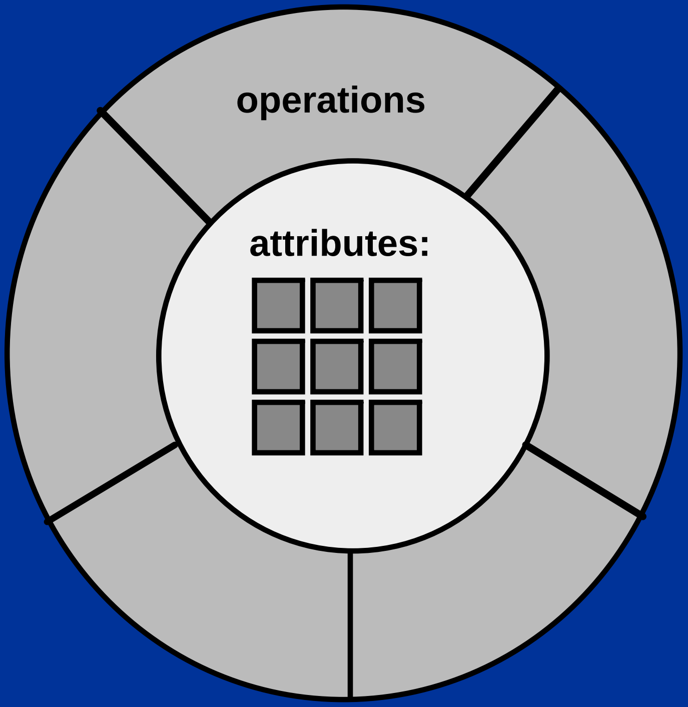
- Must be understood to apply class-based elements of the analysis model
- Key concepts:
  - Classes and objects
  - Attributes and operations
  - Encapsulation and instantiation
  - Inheritance

Classes

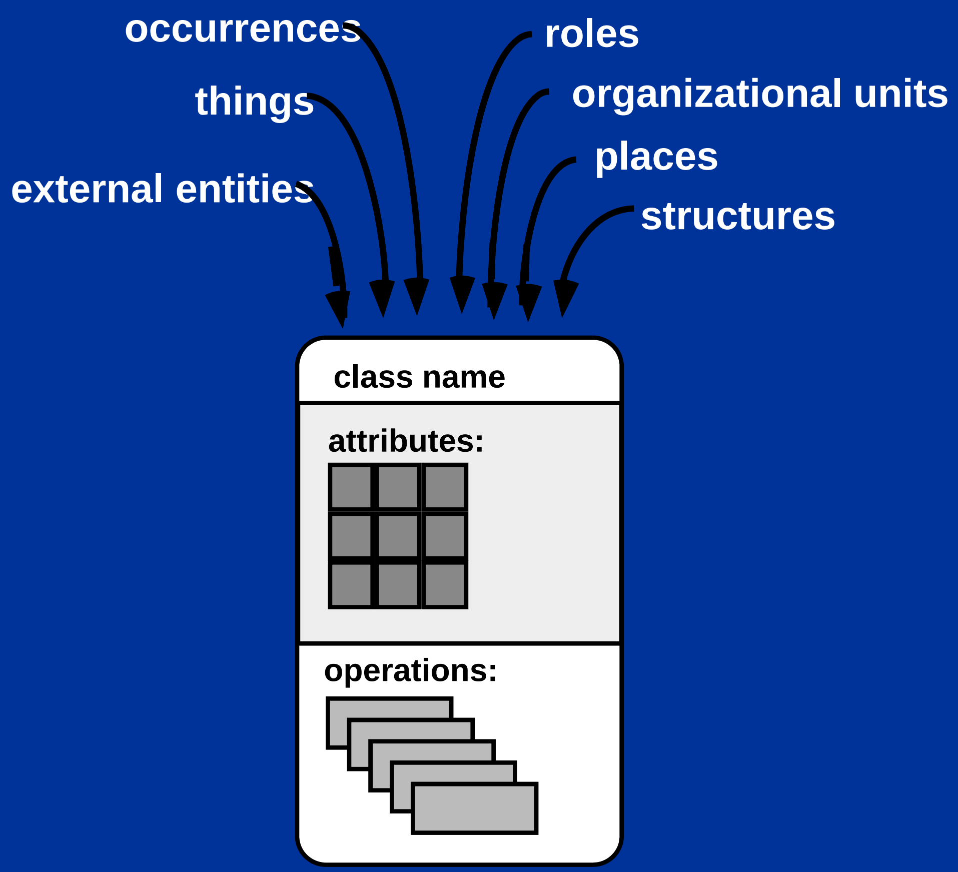
- object-oriented thinking begins with the definition of a class, often defined as:
  - template
  - generalized description
  - “blueprint” ... describing a collection of similar items
- a metaclass (also called a superclass) establishes a hierarchy of classes
- once a class of items is defined, a specific instance of the class can be identified

Building a Class



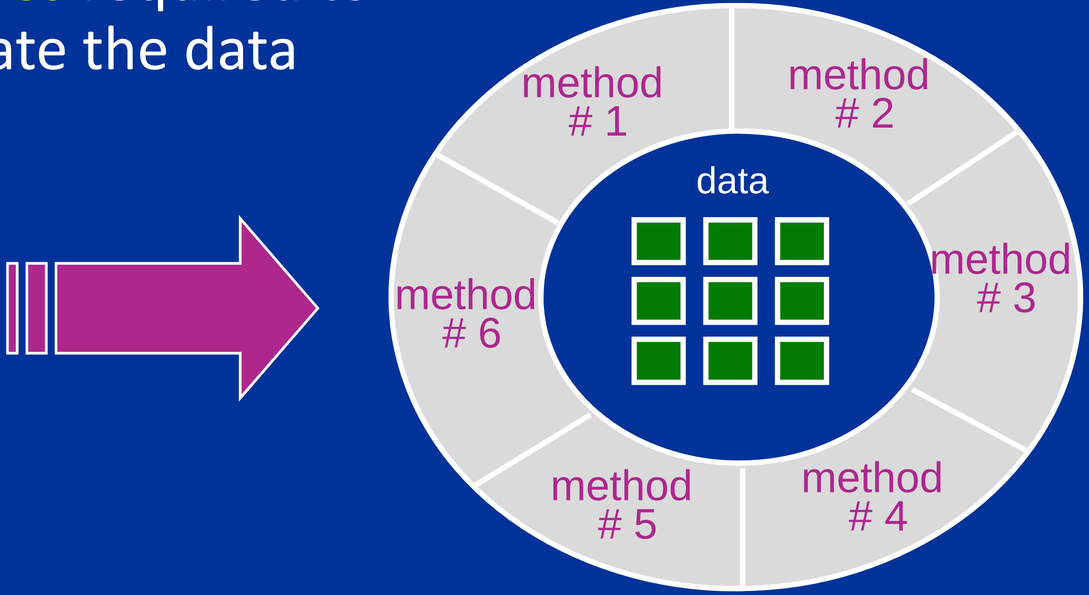


What is a Class?

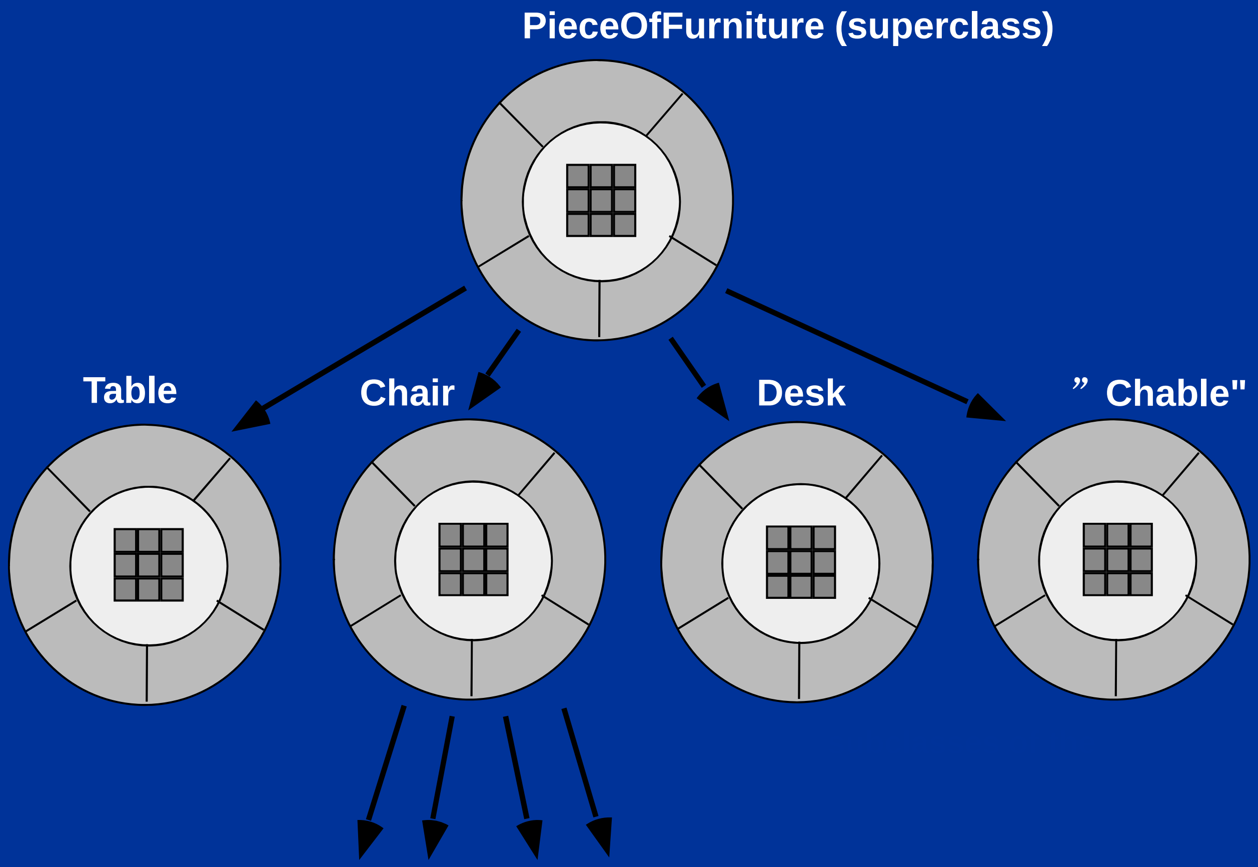


Encapsulation/Hiding

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



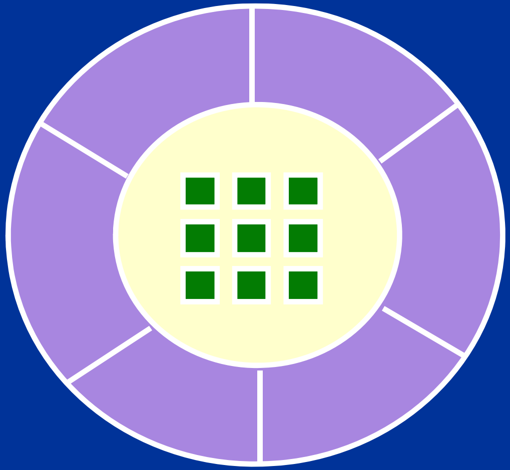
Class Hierarchy



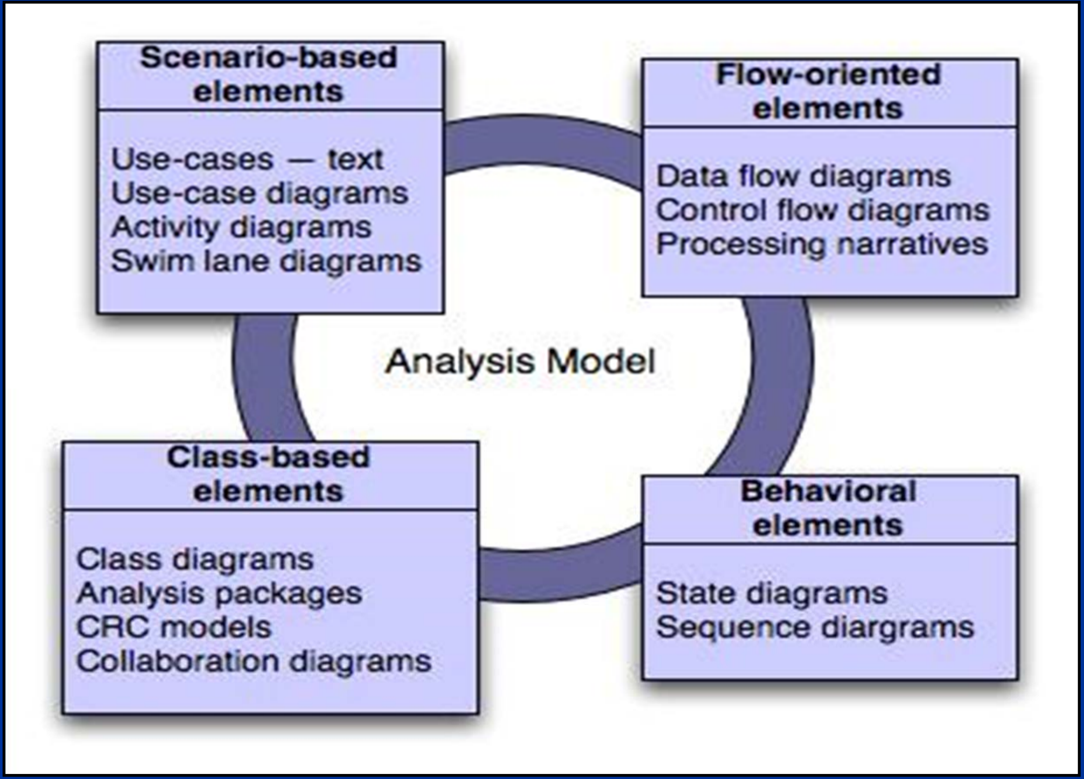
Methods (a. k. a. Operations, Services)

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.

A method is invoked via message passing.



8.4 Analysis Model



Elements of the analysis model

8.5 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?

Use Case UC1: Process Sale

Primary Actor: Cashier
Stakeholders and Interests:

- Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer short ages are deducted from his/her salary.
- Salesperson: Wants sales commissions updated.
- Customer: Wants purchase and fast service with minimal effort. Wants proof of pur chase to support returns.
- Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server compo nents (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
- Government Tax Agencies: Want to collect tax from every sale. May be multiple agen cies, such as national, state, and county.
- Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.
Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):
*a. At any time, System fails:
To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
   2a. System detects anomalies preventing recovery:
   1. System signals error to the Cashier, records the error, and enters a clean state.
   2. Cashier starts a new sale.

3a. Invalid identifier:
1. System signals error and rejects entry. 3b. There are multiple of same item category and tracking unique item identity not
important (e.g., 5 packages of veggie-burgers):
1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:
1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:
1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:
1. System records sale so that it is available for retrieval on any POS terminal. 4a.
The system generated item price is not wanted (e.g., Customer complained about something and is offered a lower price):
1. Cashier enters override price.
2. System presents new price.

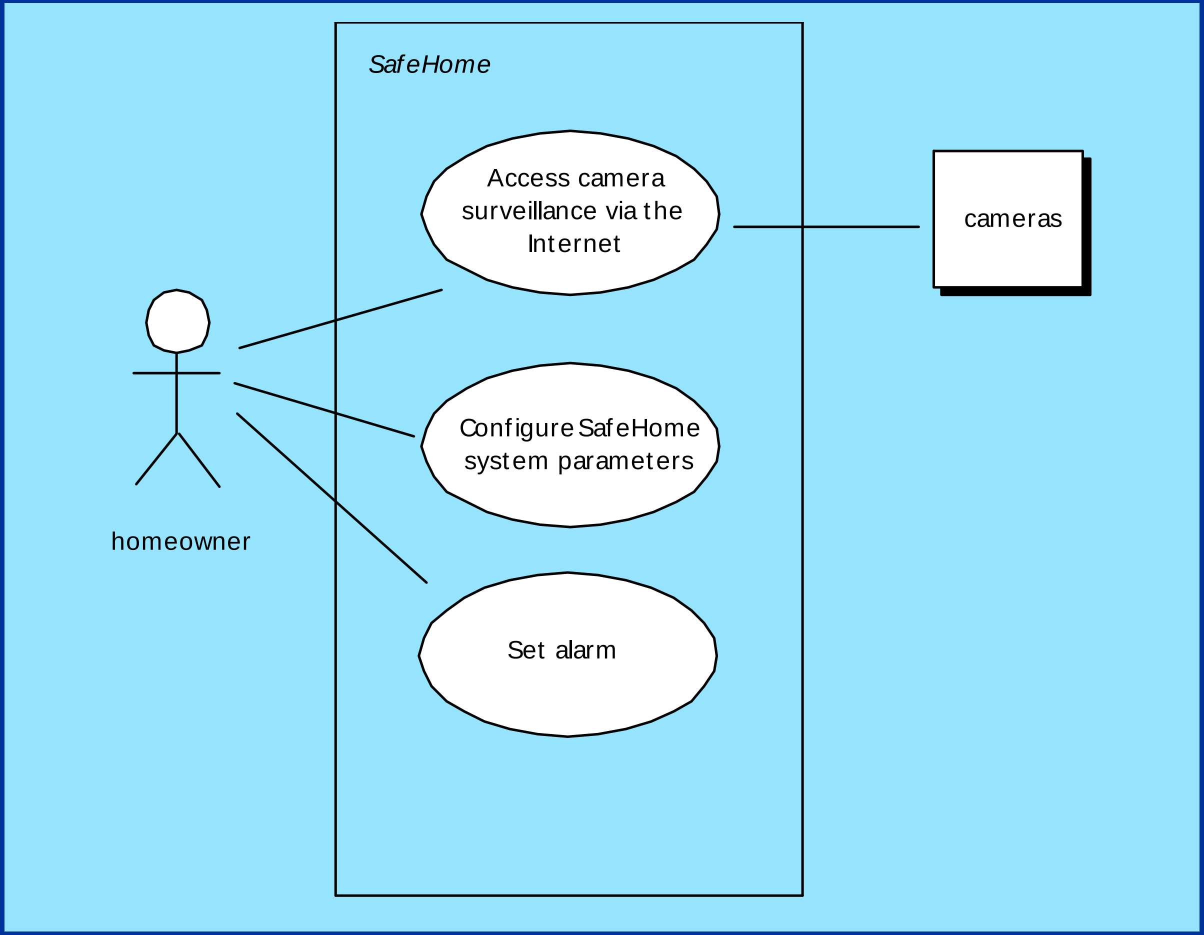
5a. System detects failure to communicate with external tax calculation system service:
1. System restarts the service on the POS node, and continues. 1a. System detects that the service does not restart.
   1. System signals error.
   2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):
1. Cashier signals discount request.
2. Cashier enters Customer identification.
3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:
1. Cashier signals credit request.
2. Cashier enters Customer identification.
3. Systems applies credit up to price=0, and reduces remaining credit.

6a. Customer says they intended to pay by cash but don't have enough cash:
1a. Customer uses an alternate payment method.
1b. Customer tells Cashier to cancel sale. Cashier cancels sale on System.

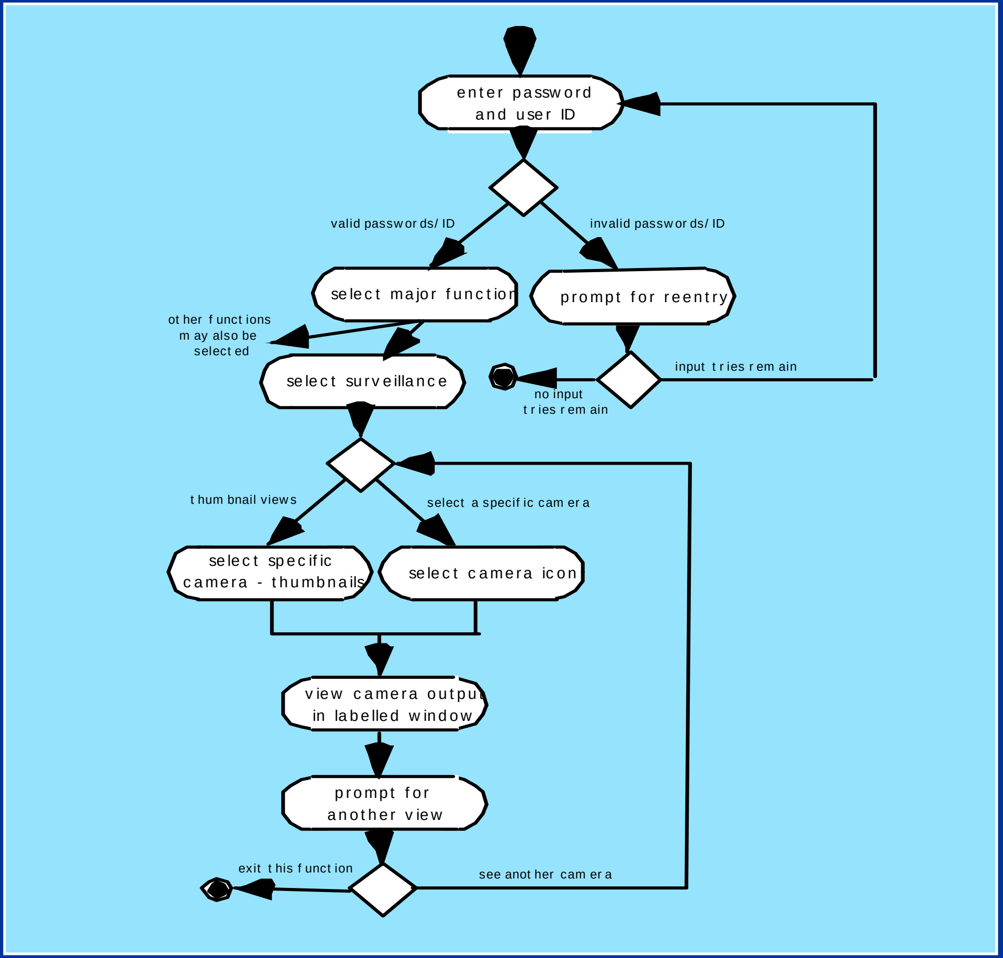
Use-Case Diagram



SafeHome
Access camera surveillance via the Internet
Configure SafeHome system parameters
Set alarm
homeowner
cameras

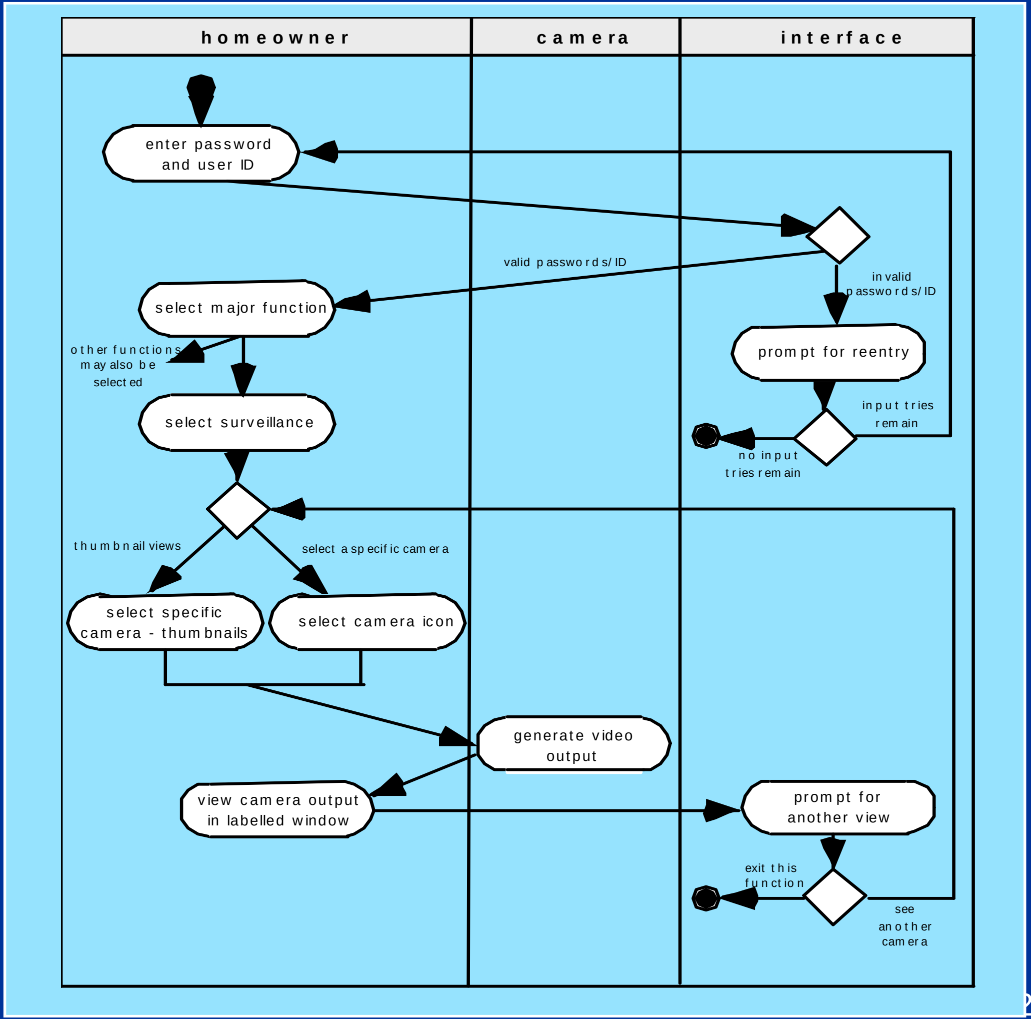
Activity Diagram

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



Swimlane Diagrams

Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle



8.6 Flow-Oriented Modeling

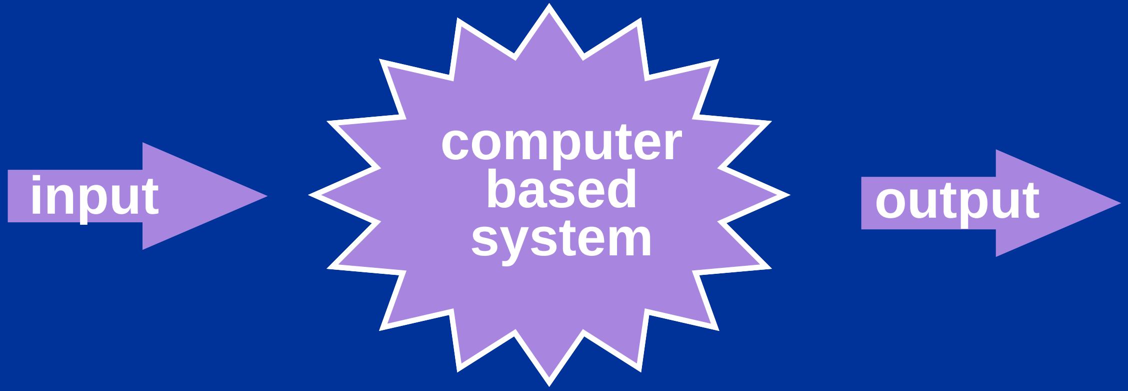
Represents how data objects are transformed as they move through the system

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

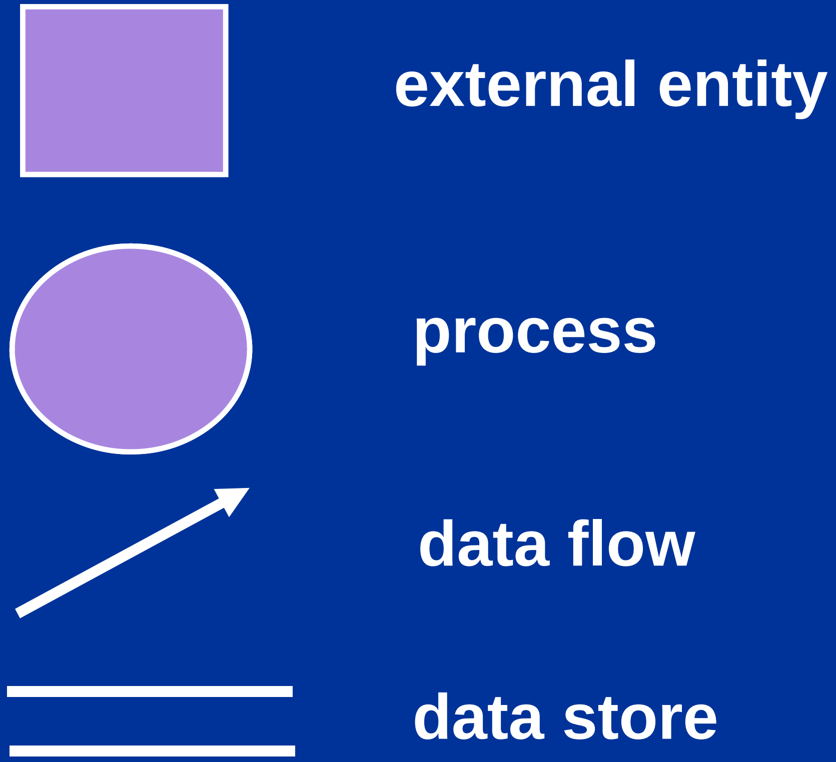
Considered by many to be an 'old school' approach, flow-oriented modeling continues to provide a view of the system that is unique—it should be used to supplement other analysis model elements

The Flow Model

Every computer-based system is an information transform ....



Flow Modeling Notation



External Entity

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

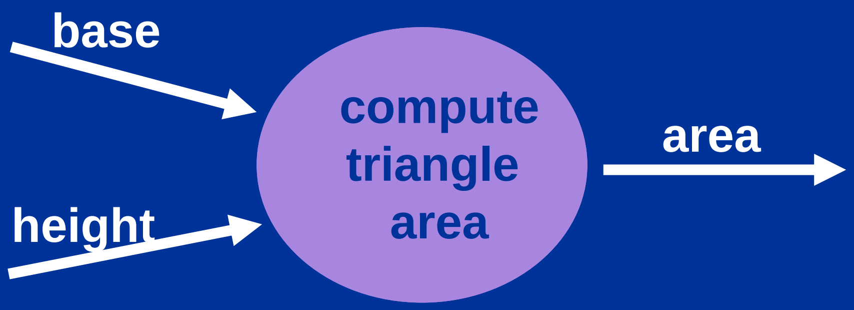
A data transformer (changes input to output)

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

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

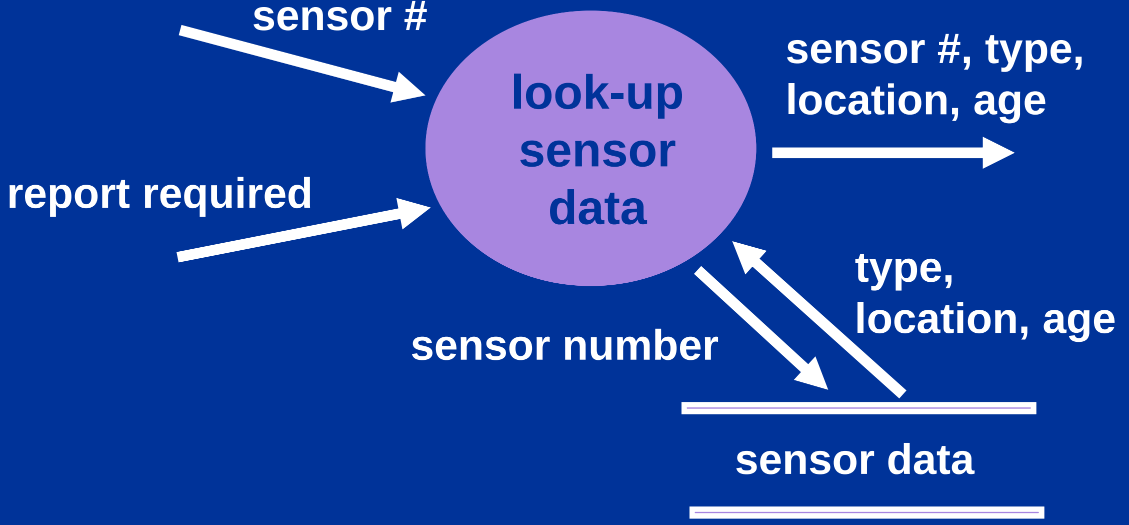
Data Flow

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



Data Stores

Data is often stored for later use.



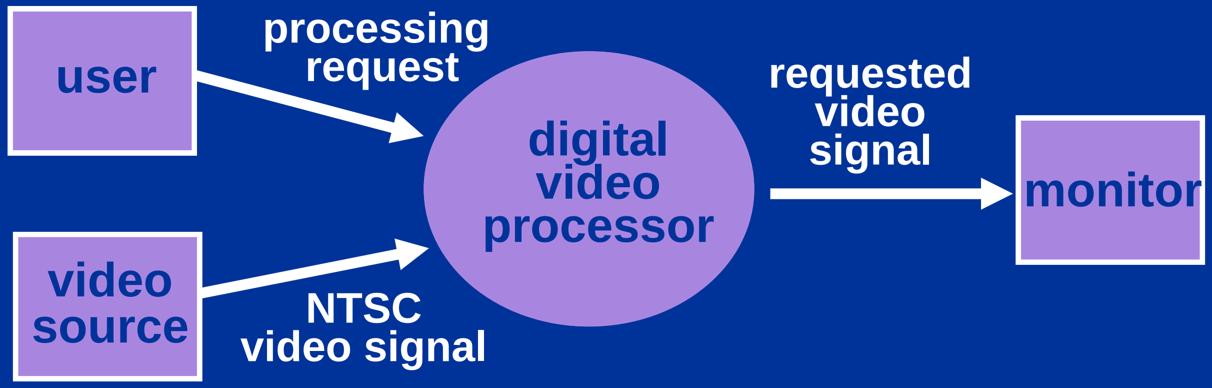
Data Flow Diagramming: Guidelines

- 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)
- always show external entities at level 0
- always label data flow arrows
- do not represent procedural logic

Constructing a DFD

- 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

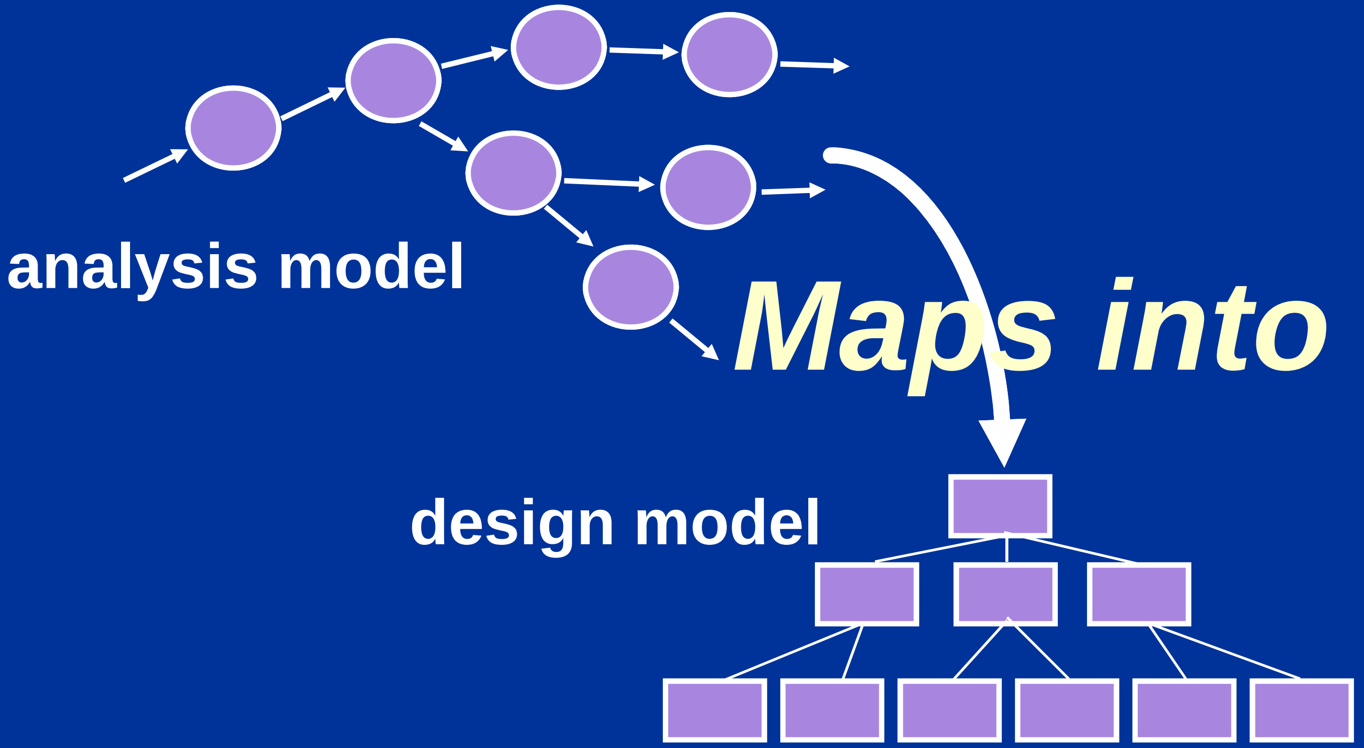
Level 0 DFD Example



Constructing a DFD…

- 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

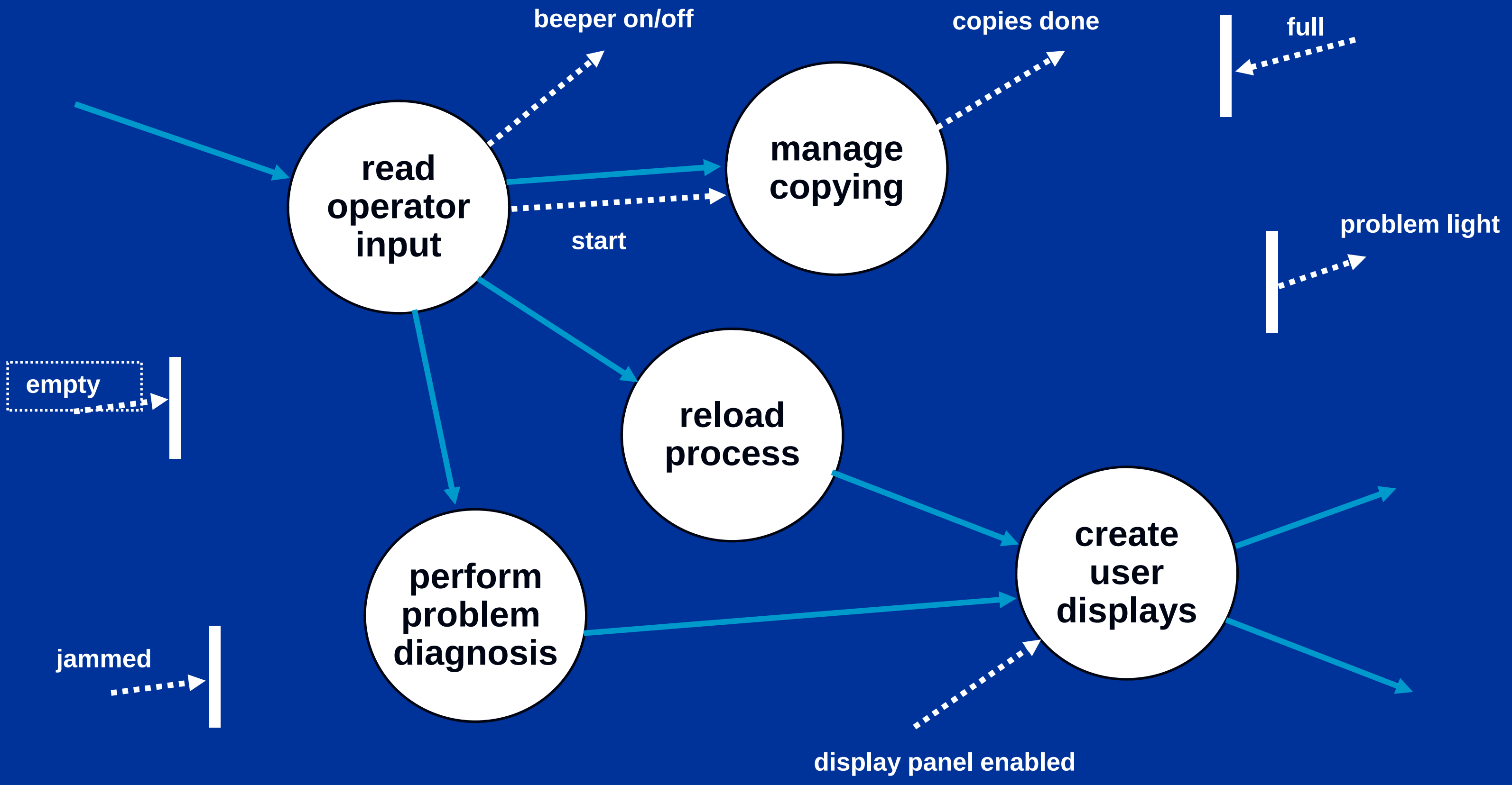
DFDs: A Look Ahead



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.

Control Flow Diagram



8.7 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

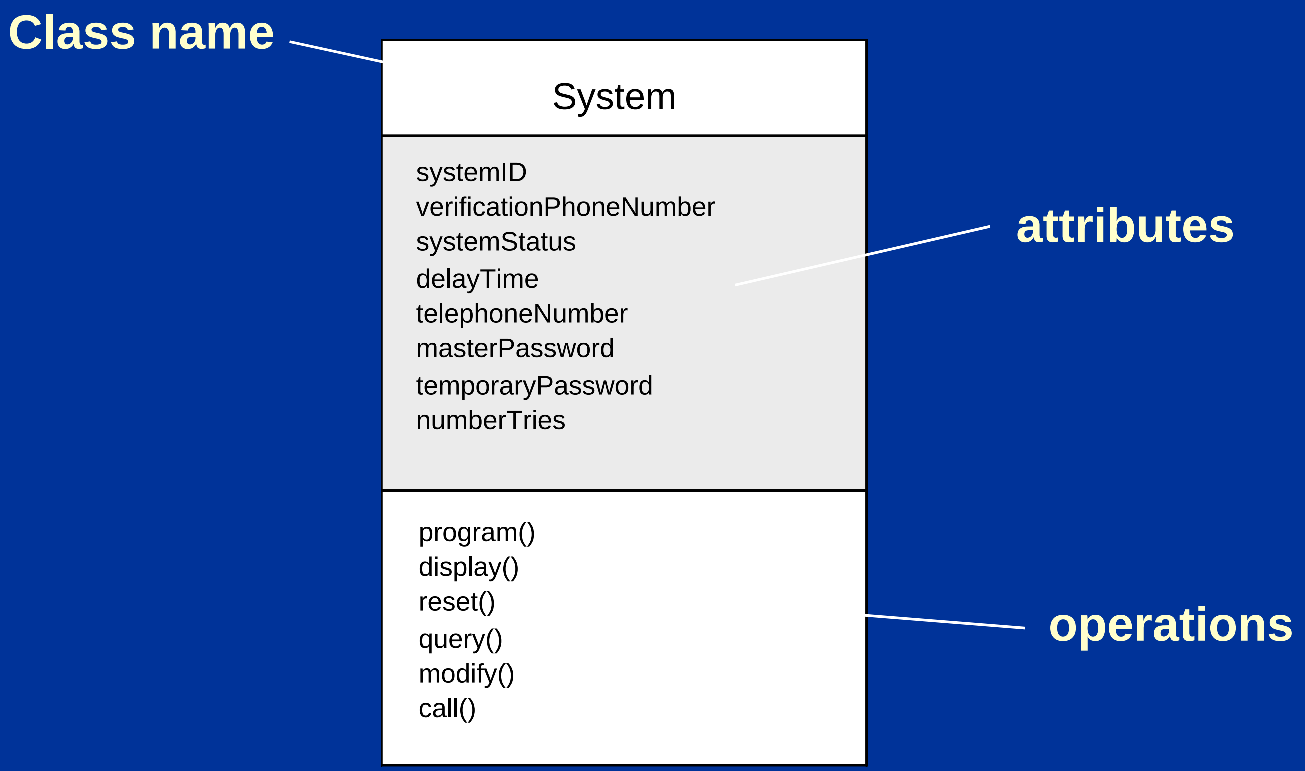
Analysis Classes

- *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.

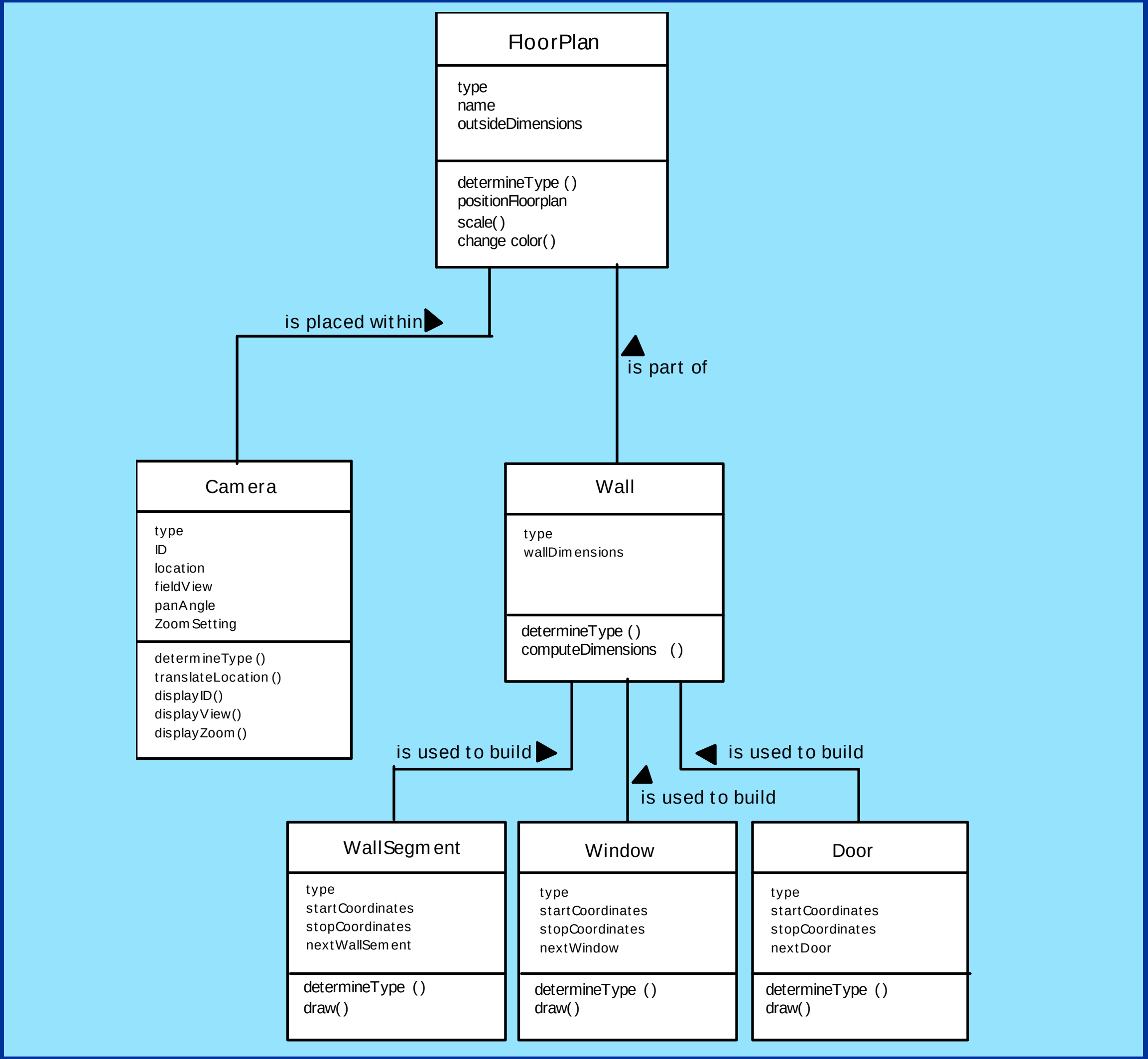
Selecting Classes—Criteria

- ✓ retained information
- ✓ needed services
- ✓ multiple attributes
- ✓ common attributes
- ✓ common operations
- ✓ essential requirements

UML Class


Class name
System
systemID
verificationPhoneNumber
systemStatus
delayTime
telephoneNumber
masterPassword
temporaryPassword
numberTries
program()
display()
reset()
query()
modify()
call()
attributes
operations

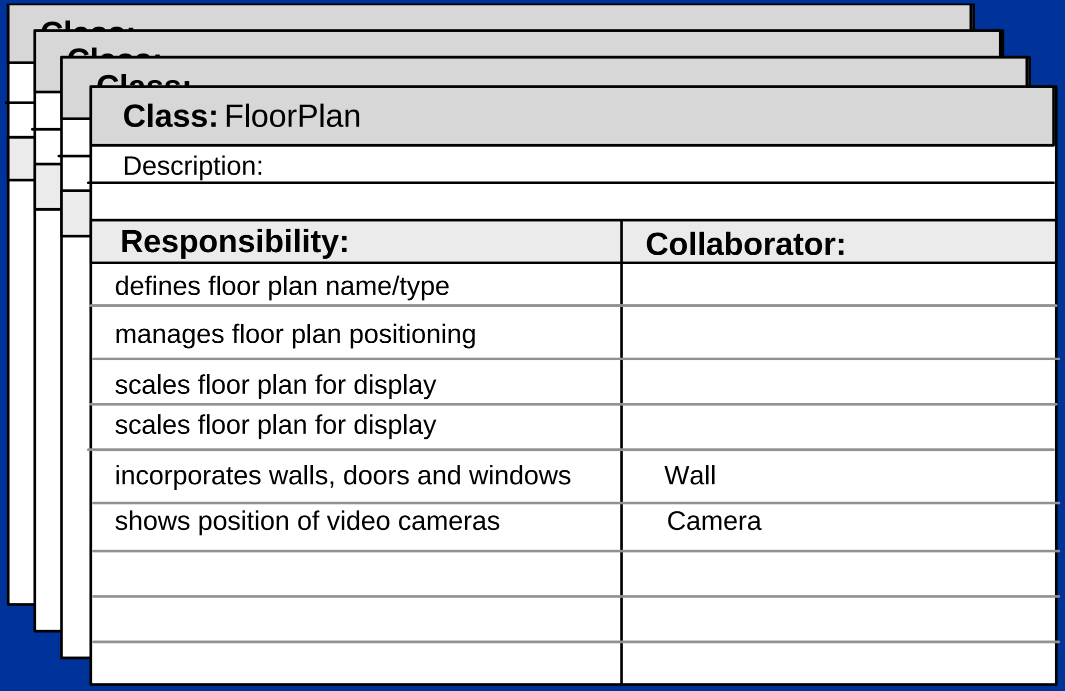
Class Diagram



CRC Modeling

- CRC: Class-Responsibility-Collaborator用于识别和组织与系统或产品需求相关的类.
- Analysis classes have “responsibilities”
  - *Responsibilities* are the attributes and operations encapsulated by the class[类所知道或能做的任何事]
- 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. [协作者是那些提供完成某个职责所需要信息的类]
  - In general, a collaboration implies either a request for information or a request for some action.

CRC Modeling

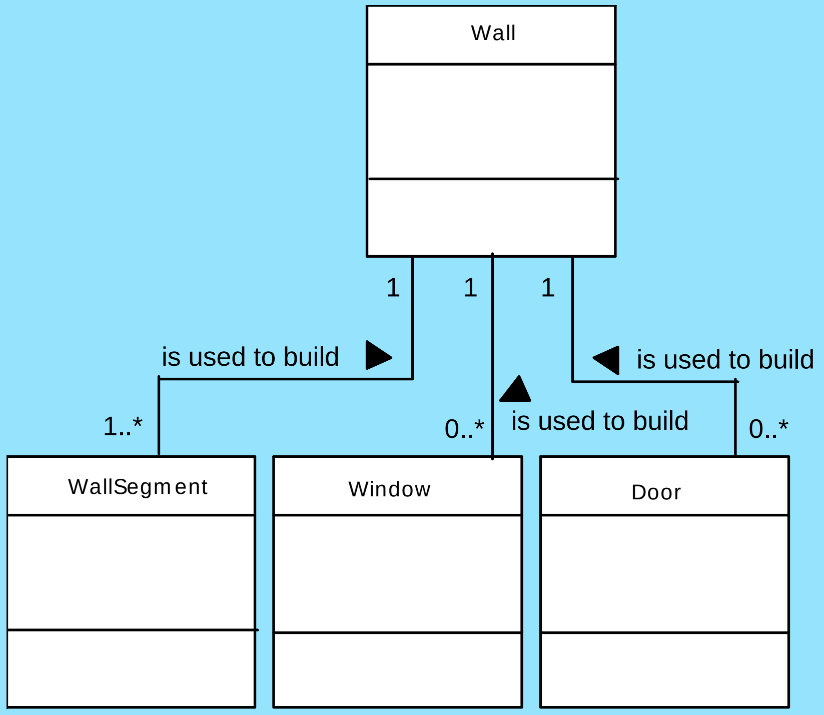


Class: FloorPlan	
Description:	
Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

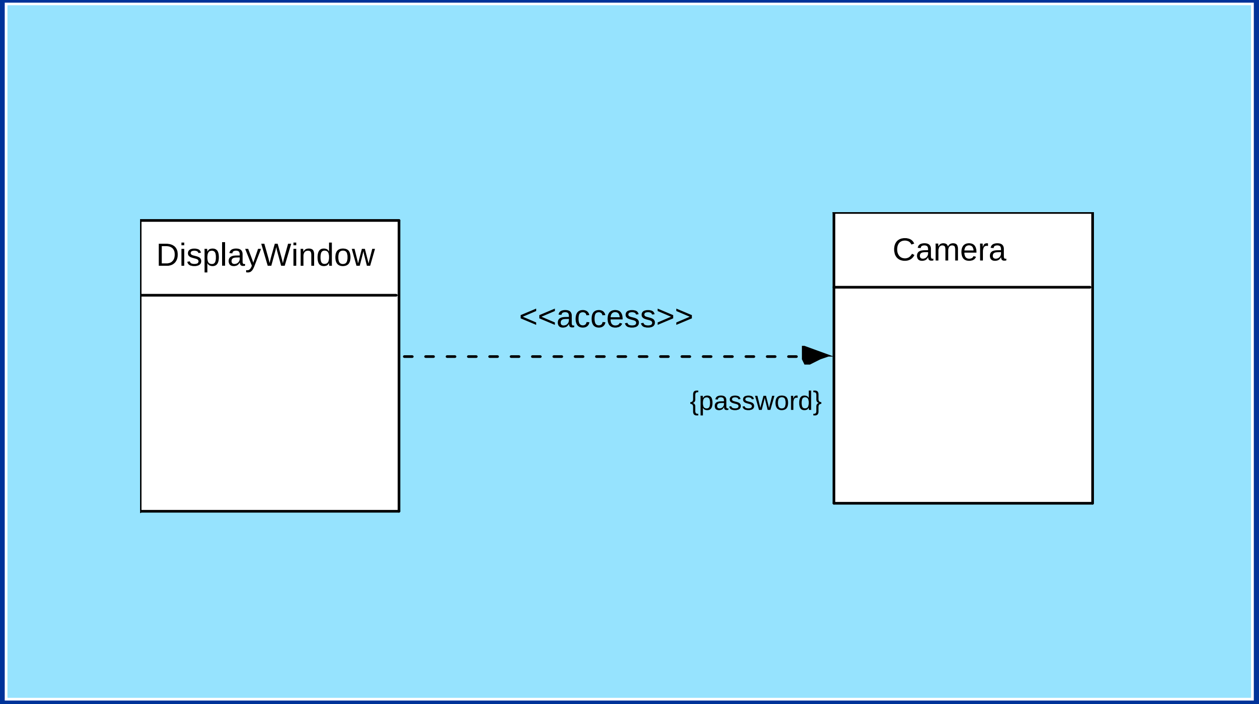
Associations and Dependencies

- 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*
  - 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



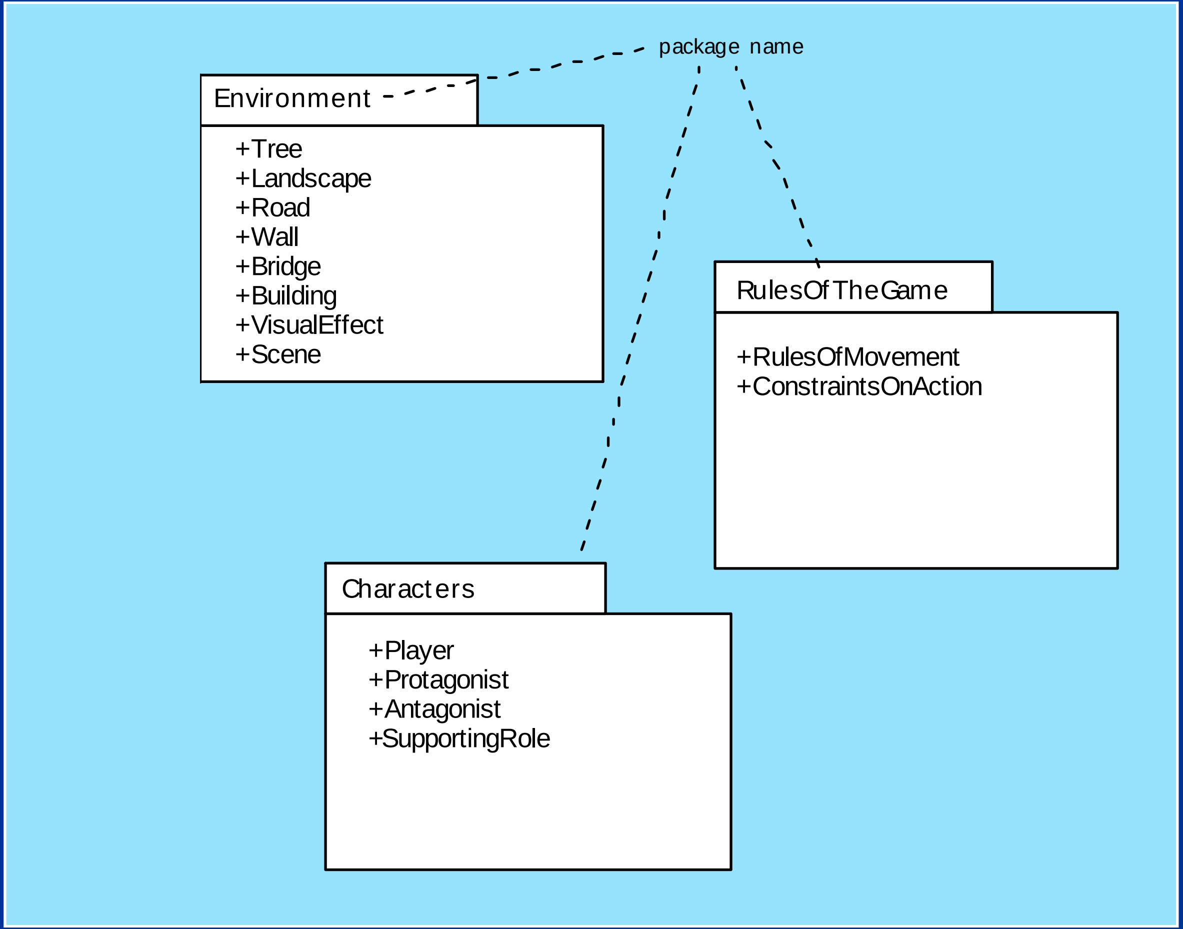
Dependencies



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 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



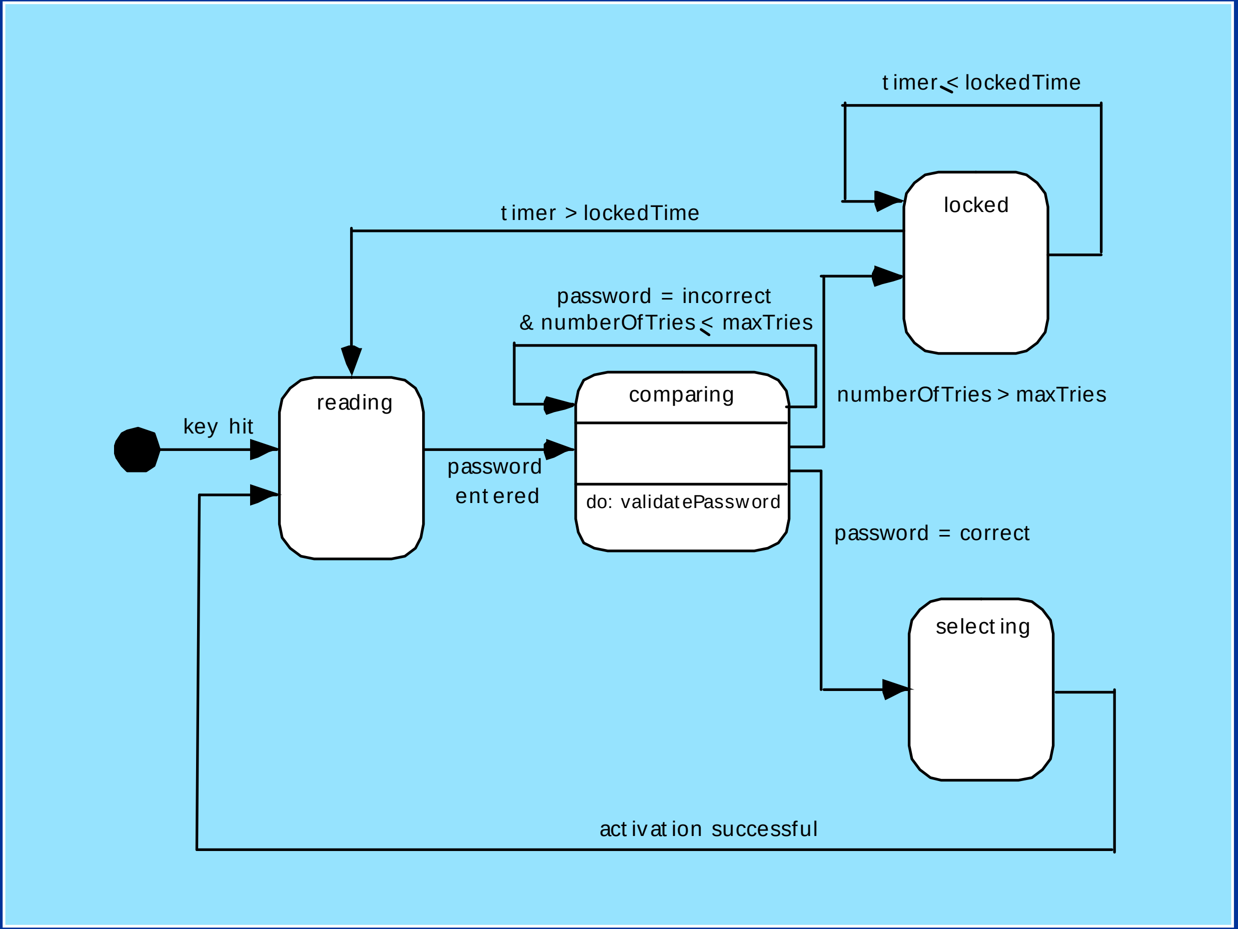
8.8 Behavioral Modeling

- 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 diagram for each use-case.
  - Build a state diagram for the system.
  - Review the behavioral model to verify accuracy and consistency.

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class when the system performs its function and
  - the state of the system when observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics [CHA93].
  - A passive state[钝态] is simply the current status of all of an object’s attributes.
  - The active state[活跃态] of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

State Diagram for the ControlPanel Class



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

Sequence Diagram

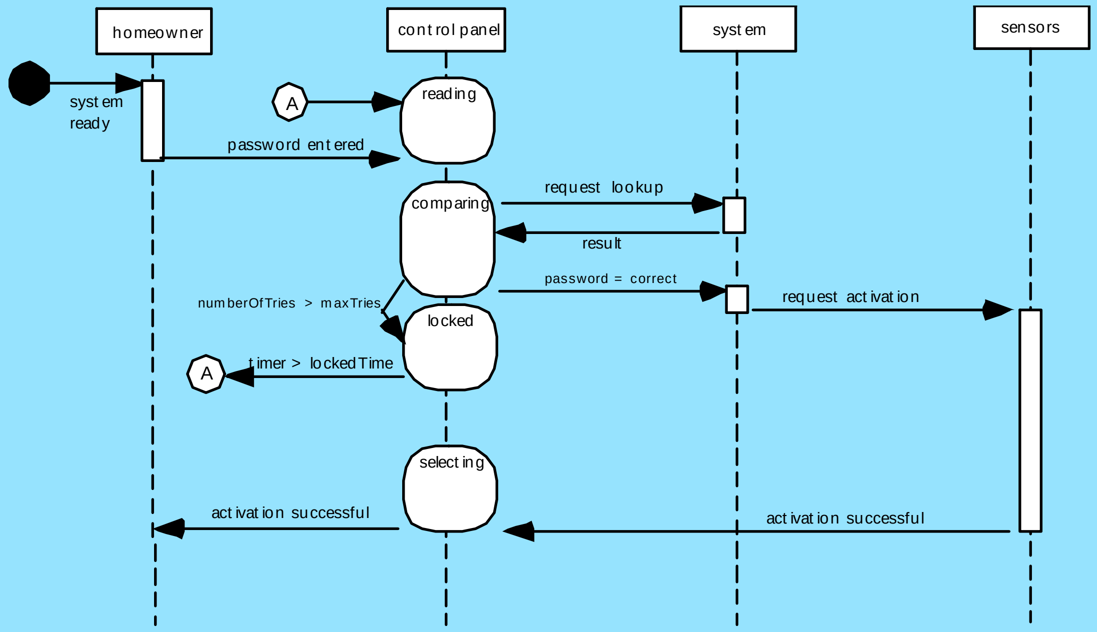


Figure 8.27 Sequence diagram (partial) for *SafeHome* security function

Writing the Software Specification

Everyone knew exactly what had to be done until someone wrote it down!