UNIT III

ANALYSIS MODELLING

Requirements Analysis – Analysis Modeling approaches – data modeling concepts – Object oriented Analysis – Scenario based modeling – Flow oriented Modeling – Class based modeling – creating a behavior model.

Requirements Analysis

Requirement Analysis results in the specification of software's operational characteristics; indicates software's interface with other system element; and establishes constraints that software must need. Throughout analysis modeling the software engineer's primary focus is on what not how.

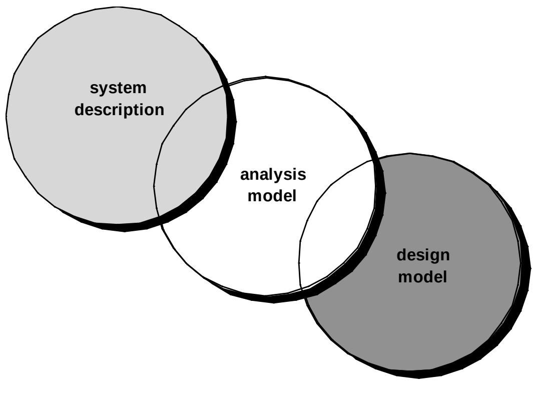
RA allows the software engineer (called an analyst or modeler in this role) to:

- 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.

Overall Objectives

The analysis model must achieve three primary objectives:

1. To describe what the customer requires
2. To establish a basis for the creation of a software design, and
3. To define a set of requirements that can be validated once the software is built.



The analysis model bridges the gap between a system-level description that describes overall functionality as it is achieved by applying software, hardware, data, human, and other system elements and a software design that describes the software application architecture.

Analysis Rules of Thumb

- 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.
    - For example, a database may be required, but the classes necessary to implement it, the functions required to access it, and the behavior that will be exhibited as it is used should be considered only after problem domain analysis has been completed.
- Minimize coupling throughout the system.
    - The level of interconnectedness between classes and functions should be reduced to a minimum.
- Be certain that the analysis model provides value to all stakeholders.
    - Each constituency has its own use for the model.
- Keep the model as simple as it can be.
    - Ex: Don't add additional diagrams when they provide no new information.
    - Only modeling elements that have values should be implemented.

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

- 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

Analysis Modeling Approaches

One view of analysis modeling, called structural analysis, considers data and the processes that transform the data as separate entities. Data objects are modeled in a way that defines attributes and relationships. Processes that manipulate data objects are in a manner that shows how they transform data as data objects flow through the system.

A second approach to analysis modeling called object-oriented analysis focuses on the definition of classes and the manner in which they collaborate with one another to affect customer requirements. UML and the Unified Process are predominantly Object Oriented.

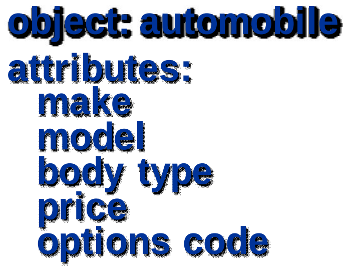
Data Modeling Concepts

Data Objects

A data object is a representation of almost any composite information that must be processed by software. By composite, we mean something that has a number of different properties and attributes.

- "Width" (a single value) would not be a valid data object, but dimensions (incorporating height, width and depth) could be defined as object.

A data object encapsulates data only – there is no reference within a data object to operations that act on the data. Therefore, the data can be represented as a table below.


object: automobile
attributes:
  make
  model
  body type
  price
  options code

Data Attributes

Data attributes define the properties of a data object and take one of three different characteristics. They can be used to:

1. Name an instance of the data object.
2. Describe the instance, or
3. Make reference to another instance in another table.

In addition, one or more of the attributes, must be defined as an identifier, i.e., the identifier attribute becomes a “key” when we want to find an instance of the data object. Values for the identifier(s) are unique, although this is not a requirement.

Referring to the data object car, a reasonable identifier might be the ID number.

Relationships

Indicates “connectedness”; a "fact" that must be "remembered" by the system and cannot or is not computed or derived mechanically

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

We can define a set of object/relationship pairs that define the relevant relationships. For example:

- A person owns a car.
- A person is insured to drive a car.

The relationship owns and insured to drive define the relevant connections between person and car.

Object-Oriented Analysis

The intent of Object Oriented Analysis (OOA) is to define all classes (and the relationships and behavior associated with them that are relevant to the problem to be solved.

To accomplish this, a number of tasks must occur:

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

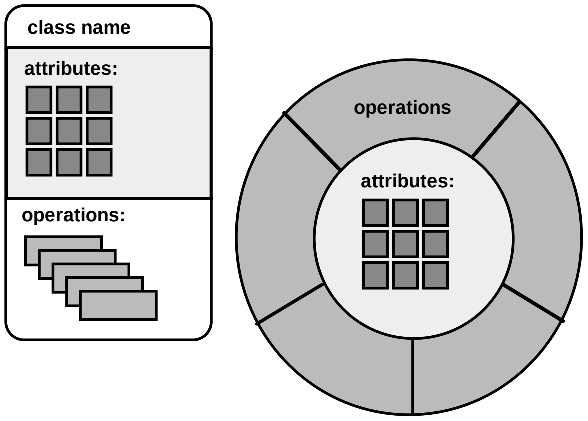
OOA builds a class-oriented model that relies on an understanding of OO concepts.

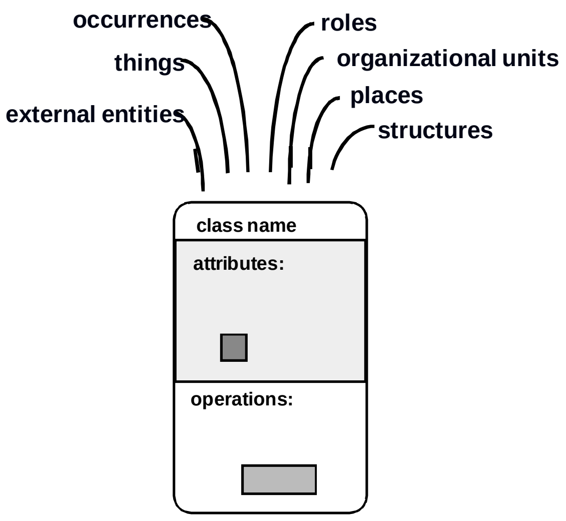
- Classes and objects
- Attributes and operations
- Encapsulation and instantiation
- Inheritance

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.

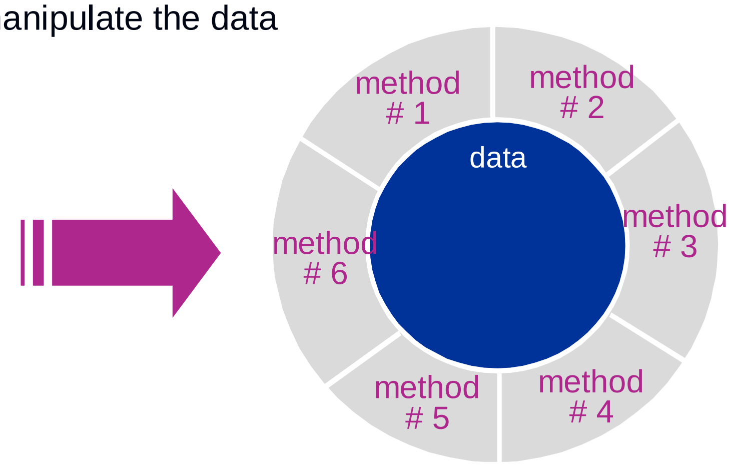
Building a class



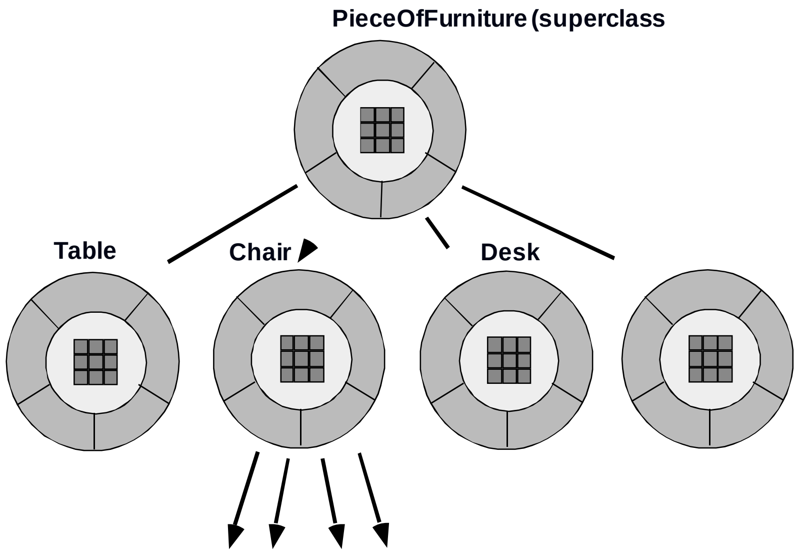


Encapsulating and Hiding

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.

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

The concept is relatively easy to understand- describe a specific usage scenario in straightforward language from the point of view of a defined actor.

Writing Use-Cases

(1) What should we write about?

Inception and elicitation provide us the information we need to begin writing use cases.

(2) How much should we write about it?

(3) How detailed should we make our description?

(4) How should we organize the description?

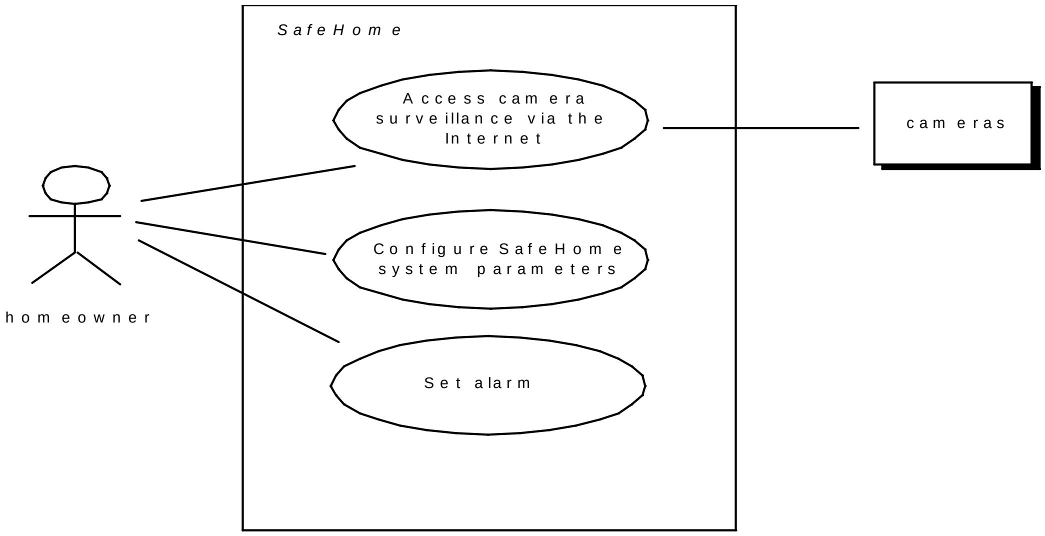
Use Cases:

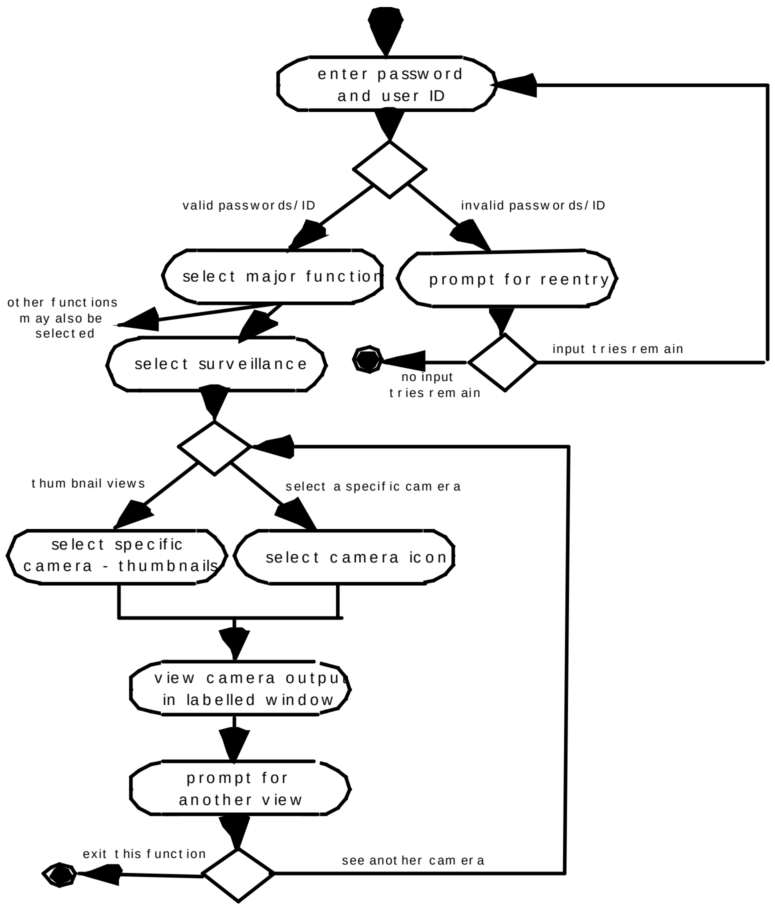
- A scenario that describes a “thread of usage” for a system.
- Actors represent roles people or devices play as the system functions.
- Users can play a number of different roles for a given scenario.

Quality Function Deployment and other R.E. mechanisms are used to identify stakeholders, define the scope of the problem, specify overall operational goals, outline all known functional requirements, and describe the object that will be manipulated by the system.

Developing an Activity Diagram

- 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?

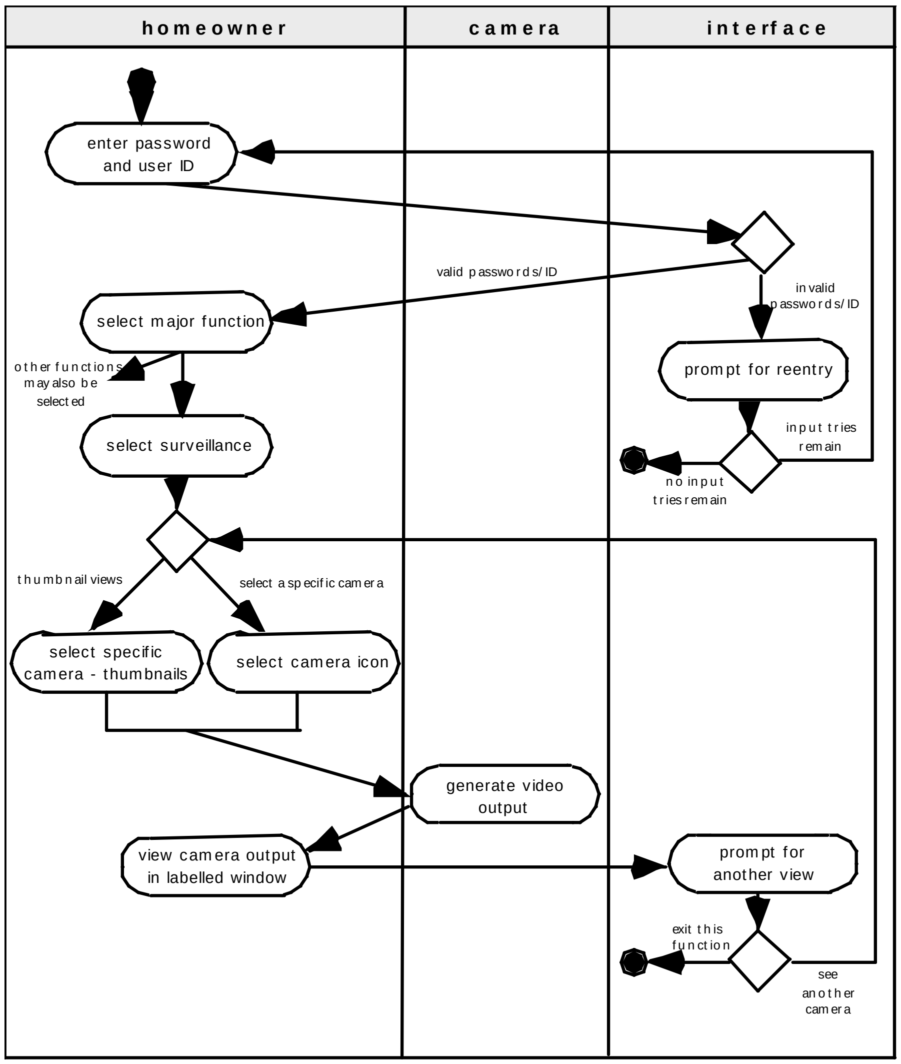

SafeHome
Access camera surveillance via the Internet
Configure SafeHome system parameters
Set alarm
homeowner
cameras


enter password and user ID
valid passwords/ID
invalid passwords/ID
select major function
prompt for reentry
other functions may also be selected
select surveillance
input tries remain
no input tries remain
thumbnail views
select a specific camera
select specific camera - thumbnails
select camera icon
view camera output in labelled window
prompt for another view
exit this function
see another camera

Swimlane Diagrams

The UML swimlane diagram is a useful variation of the activity diagram and allows the modeler to represent the flow of activities described by the user-case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.



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 to complement UML diagrams.

Considered by many to be an 'old school' approach, flow-oriented modeling continues to provide a view of the system that is unique.

The DFD takes an input-process-output insight into system requirements and flow.

Data objects are represented by labeled arrows and transformations are represented by circles (called bubbles).

Creating a Data Flow Model

The DFD diagram enables the software engineer to develop models of the information domain and functional domain at the same time.

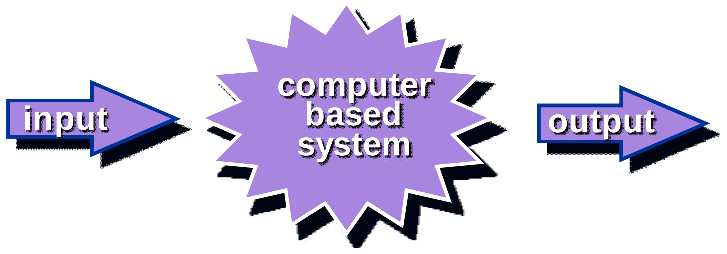
As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition of the system.

Guidelines:

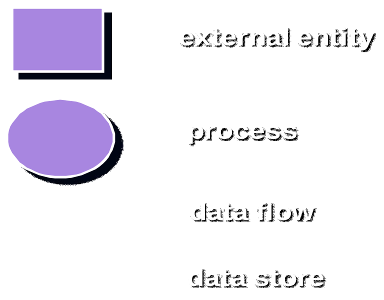
1. The level 0 data DFD should depict the software/system as a single bubble.
2. Primary I/O should be carefully noted.
3. Refinement should begin by isolating candidate processes, data objects, and data stores.
4. All arrows and bubbles should be labeled with meaningful names.
5. Information flow continuity must be maintained from level to level.
6. One bubble at one time should be refined.

Information continuity must be maintained at each level as DFD level is refined. This mean that input and output at one level must be the same as input and output at a refined level. Figures show how DFD works.

The flow Model



Flow Modeling Notations



External Entity

A producer or consumer of data

Example: computer-based system

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

Process

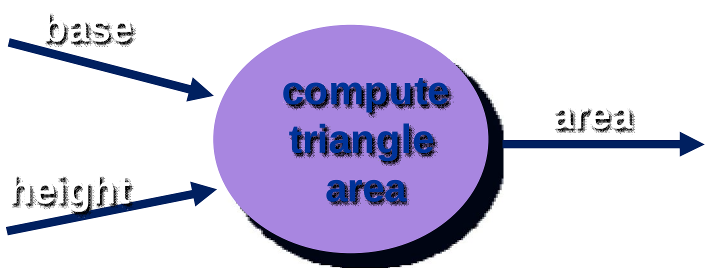
A data transformer (changes input to output)

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

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

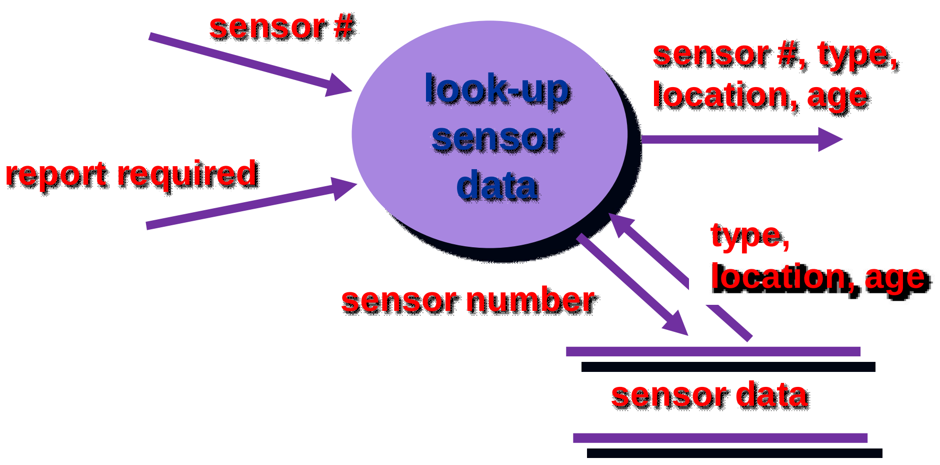
Data Flow

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



Data Stores

Data is often stored for later use.

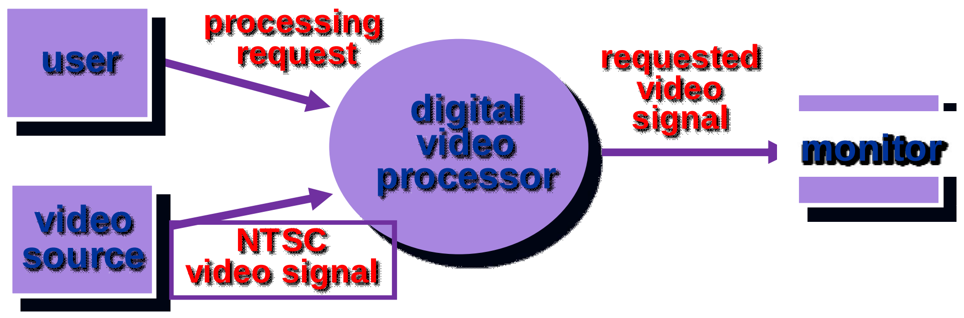


Data Flow Diagramming:

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

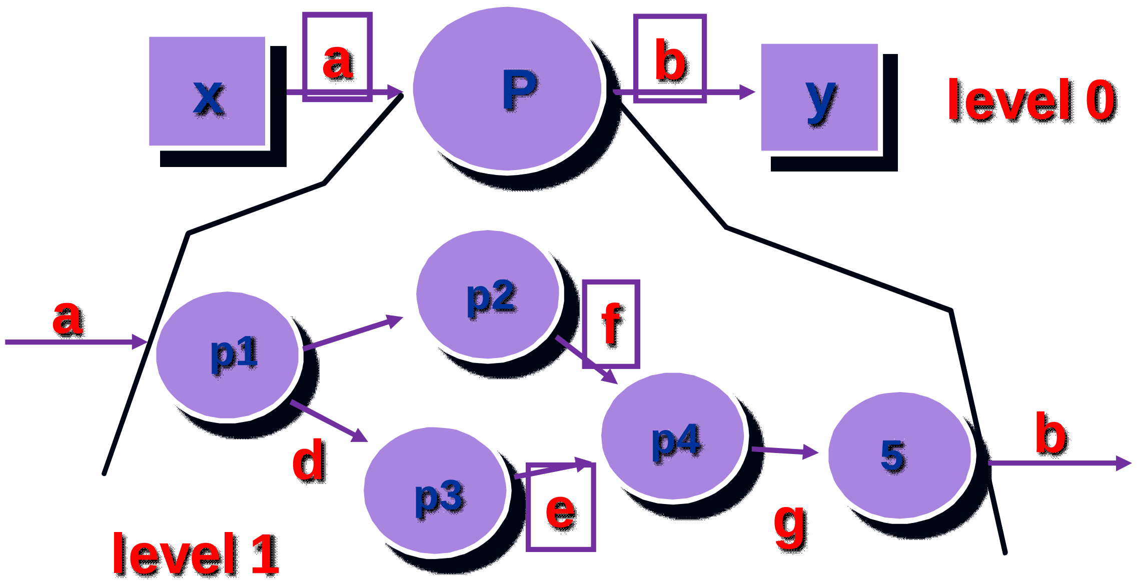
Level 0 DFD Example



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

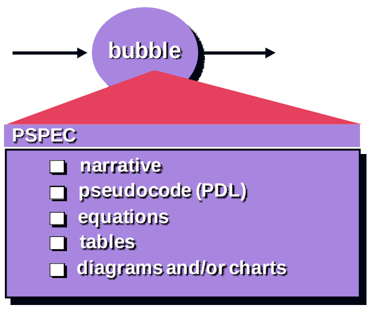


Flow Modeling Notes

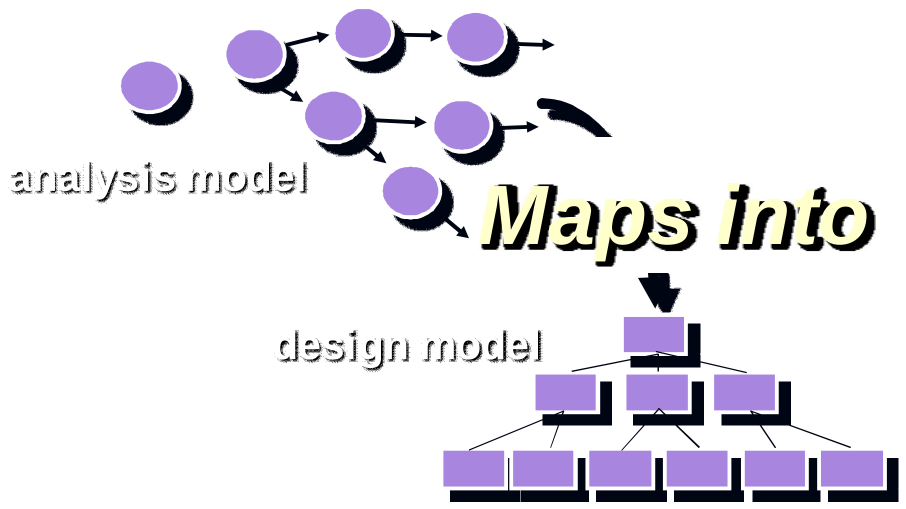
- Each bubble is refined until it does just one thing
- The expansion ratio decreases as the number of levels increase
- Most systems require between 3 and 7 levels for an adequate flow model
- A single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)

The Process Specification

The Process Specification (PSPEC) is used to describe all flow model processes that appear at the final level of refinement. It is a “mini” specification for each transform at the lowest refined of a DFD.



DFDs: A Look Ahead



Control Flow Diagrams

The diagram represents “events” and the processes that manage these events.

An “event” is a Boolean condition that can be ascertained by:

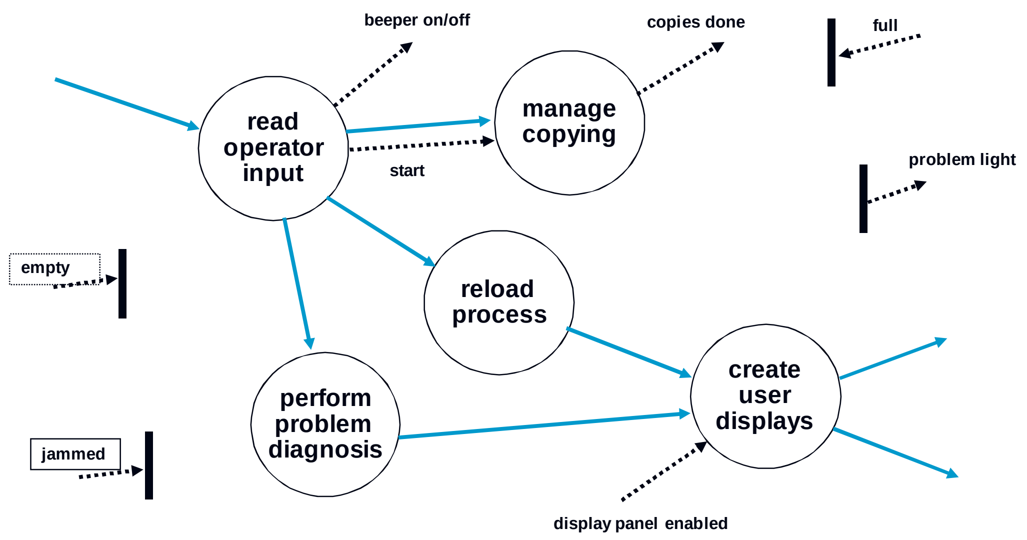
- Listing all sensors that are "read" by the software.
- Listing all interrupt conditions.
- Listing all "switches" that are actuated by an operator.
- Listing all data conditions.
- Recalling the noun/verb parse that was applied to the processing narrative, review all "control items" as possible CSPEC 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.

Control Flow Diagram



Class-Based Modeling

This section describes the process of developing an object-oriented analysis (OOA) model. The generic process described begins with guidelines for identifying potential analysis classes, suggestions for defining attributes and operations for those classes, and a discussion of the Class-Responsibility-Collaborator (CRC) model. The CRC card is used as the basis for developing a network of objects that comprise the object-relationship model.

Identifying Analysis Classes

- 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 manifest themselves in one of the following ways:

- 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, and 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.

Performing a “grammatical parse” on a processing narrative for a problem helps extracting the nouns. After identifying the nouns, a number of potential classes are proposed in a list. The list will be continued until all nouns in the processing narratives have been considered. Each entry is in the list is a potential object.

How do I determine whether a potential class should, in fact, become an analysis class?

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

1. Retained Information: The potential class will be useful during analysis only if information about it must be remembered so that the system can function.

2. Needed Services: The potential class must have a set of identifiable operations that can change the value of its attributes in some way.

3. Multiple attributes: During R.A., 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.

4. Common attributes: a set of attributes can be defined for the potential class, and these attributes apply to all instances of the class.

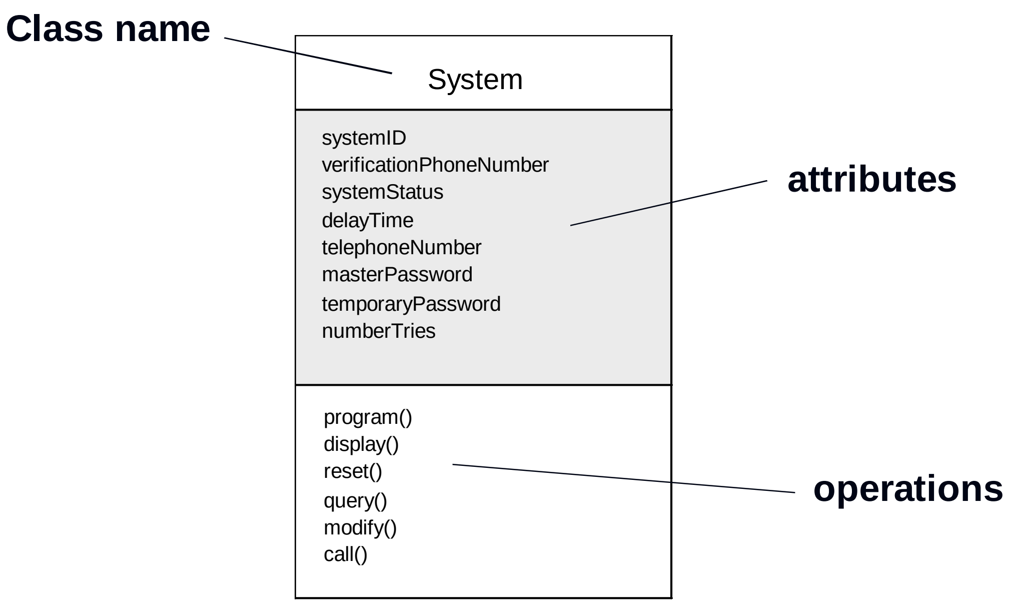
5. Common operations: a set of operations can be defined for the potential class, and these operations apply to all instances of the class.

6. 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 requirement model.

To be considered a legitimate class for inclusion in the requirements model, a potential class should satisfy all of these characteristics.

Specifying Attributes

Attributes are set of data objects that fully define the class within the context of the problem space.



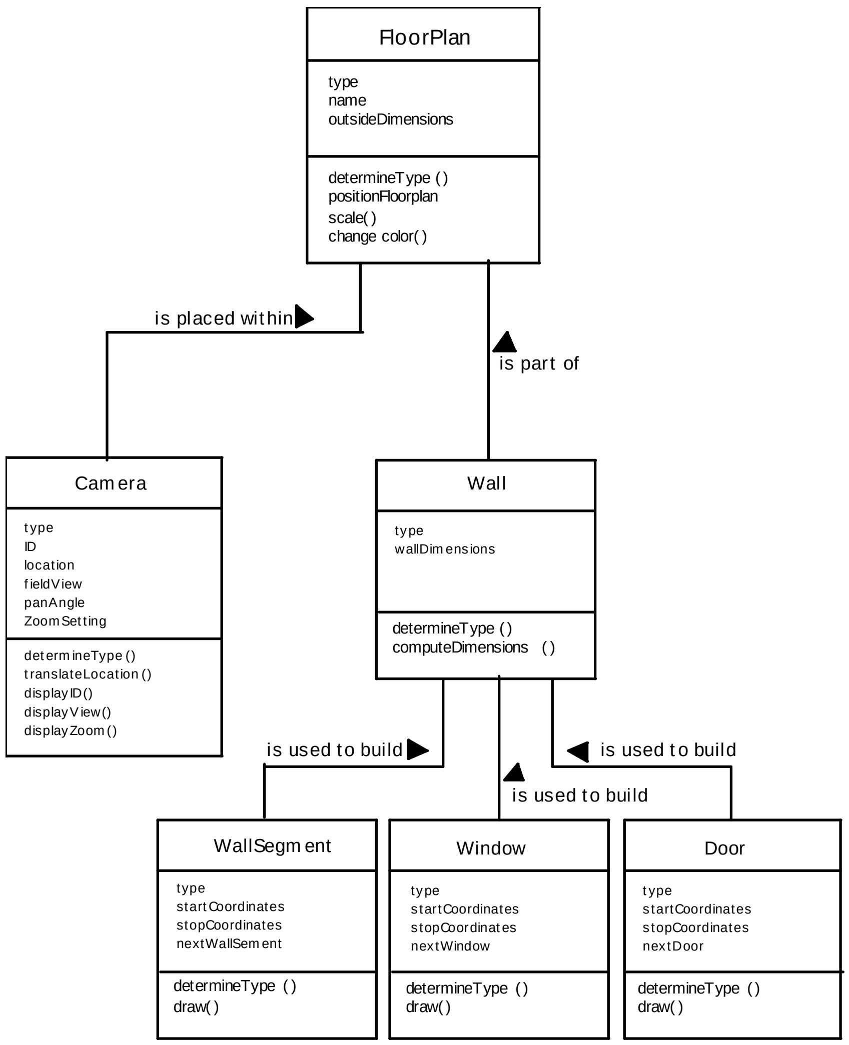
To develop a meaningful set of attributes for an analysis class, a software engineer can study a use-case and select those “things” that “reasonably” belong to the class. An important question that should be answered for each class: what data items fully define the class in the context of the problem at hand.

Defining Operations

Operations define the behavior of an object divided into 4 broad categories:

1. Operations that manipulate data (adding, deleting, selecting, reformatting.)
2. Operations that perform a computation.
3. Operations that inquire about the state of an object.
4. Operations that monitor an object for the occurrence of a controlling event.

Class Diagram



Class Responsibility Collaborator (CRC) Modeling

Class-Responsibility-Collaborator (CRC) Modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirement.

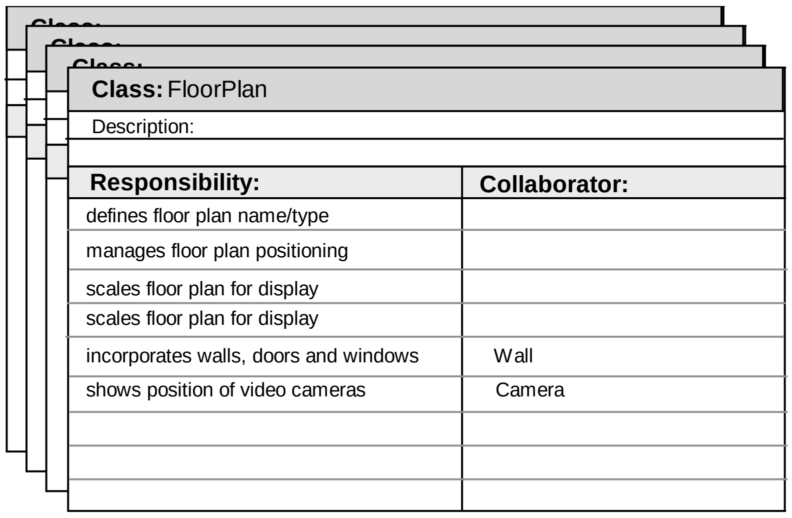
CRC modeling is described as follows:

"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."

Responsibilities are the attributes and operations that are relevant for the class. "Anything the class knows or does."

Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility.

In general, collaboration implies either a request for information or a request for some action.



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

Classes:

The taxonomy of class types can be extended by considering the following categories:

- 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.

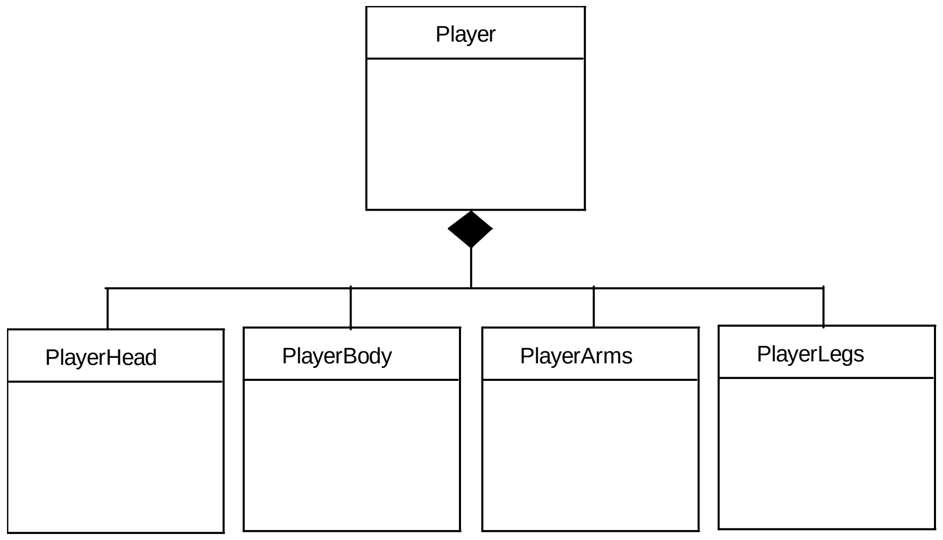
- Controller classes manage a “unit of work” [UML03] from start to finish. That is, controller classes can be designed to manage
    - the creation or update of entity objects;
    - the instantiation of boundary objects as they obtain information from entity objects;
    - complex communication between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

Responsibilities

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

Collaborations

- Classes fulfill their responsibilities in one of two ways:
    1. A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
    2. 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]:
    1. the is-part-of relationship
    2. the has-knowledge-of relationship
    3. the depends-upon relationship



Reviewing the CRC Model

1. 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).

2. All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.

3. 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.

4. 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.

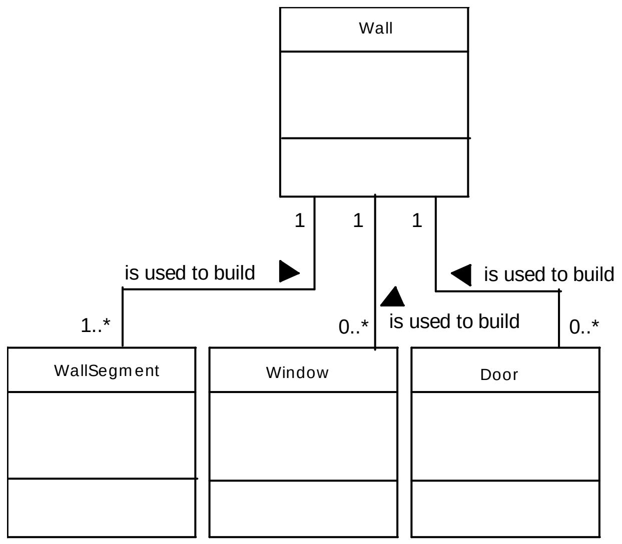
5. If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards. This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.

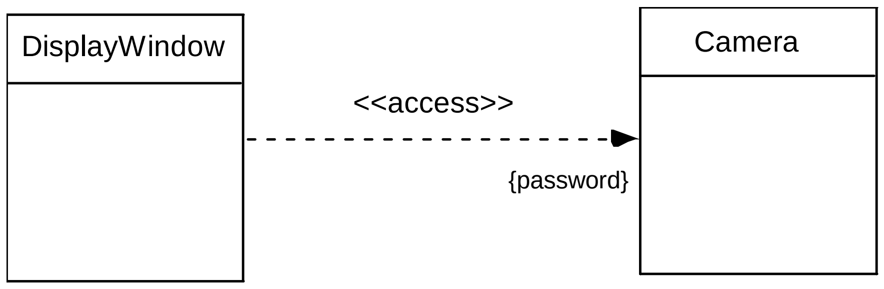
Associations and Dependencies

- 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



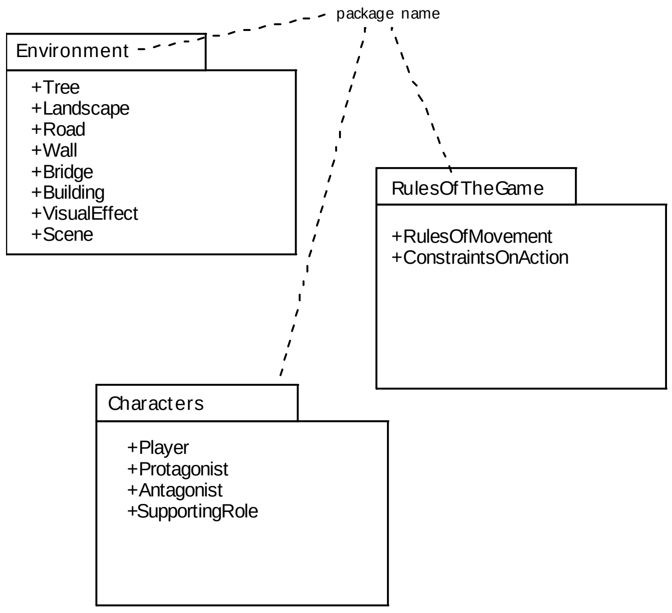


Analysis Packages

- 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.

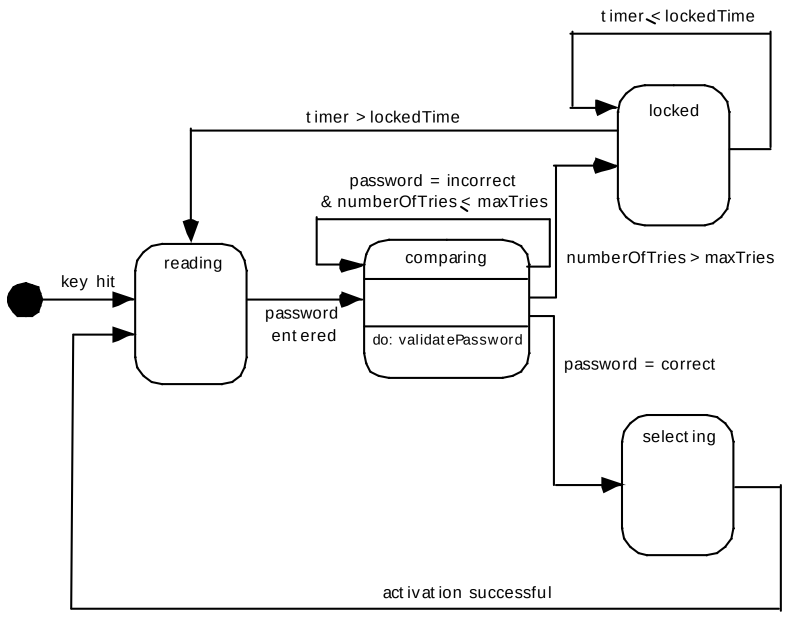


Creating a Behavioral Model

- 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 Representations
- In the context of behavioral modeling, two different characterizations of states must be considered:
    - the state of each class as the system performs its function and
    - the state of the system as 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 is simply the current status of all of an object's attributes.

- The active state of an object indicates the current status of the object as it undergoes a continuing transformation or processing.



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