Building the Requirements Model

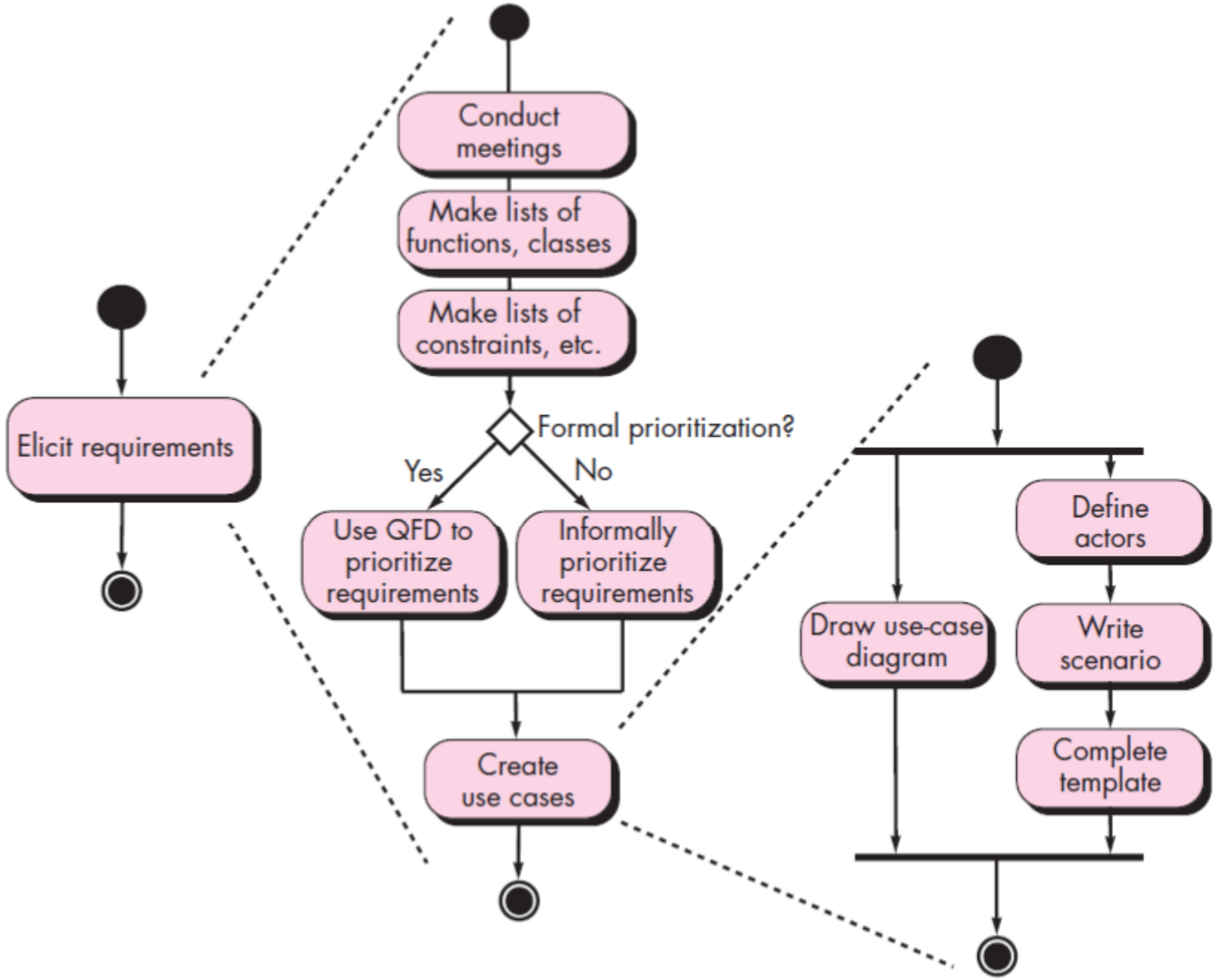
- The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.
- The model changes dynamically as you learn more about the system to be built, and other stakeholders understand more about what they really require.
- For that reason, the analysis model is a snapshot of requirements at any given time.
- You should expect it to change.
- As the requirements model evolves, certain elements will become relatively stable, providing a solid foundation for the design tasks that follow.

Elements of the Requirements Model:

- There are many different ways to look at the requirements for a computer-based system.
- Some software people argue that it's best to select one mode of representation (e.g., the use case) and apply it to the exclusion of all other modes.
- Other practitioners believe that it's worthwhile to use a number of different modes of representation to depict the requirements model.
- Different modes of representation force you to consider requirements from different viewpoints—an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity.

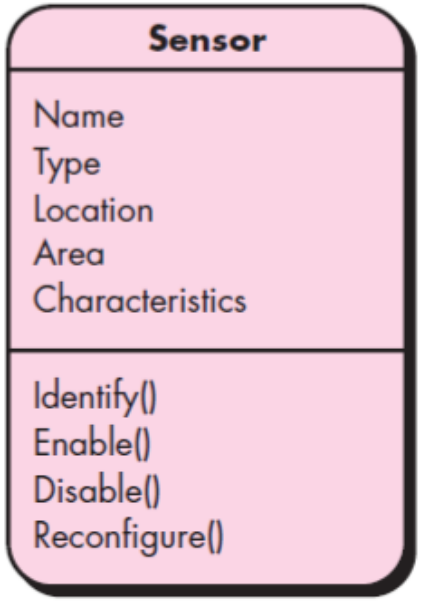
Scenario-based elements:

- The system is described from the user's point of view using a scenario-based approach.
- For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases.
- Scenario-based elements of the requirements model are often the first part of the model that is developed.
- As such, they serve as input for the creation of other modeling elements.
- Below Figure depicts a UML activity diagram for eliciting requirements and representing them using use cases.
- Three levels of elaboration are shown, culminating in a scenario-based representation.


Elicit requirements
Conduct meetings
Make lists of functions, classes
Make lists of constraints, etc.
Formal prioritization?
Yes
No
Use QFD to prioritize requirements
Informally prioritize requirements
Create use cases
Define actors
Draw use-case diagram
Write scenario
Complete template

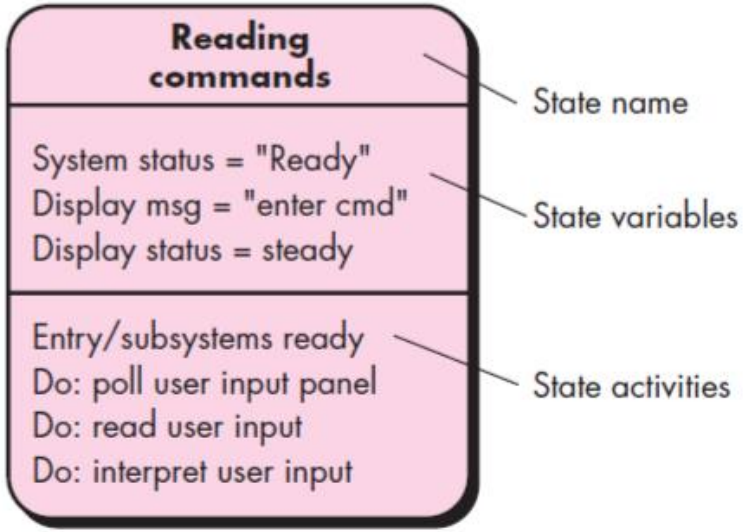
Class-based elements:

- Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system.
- These objects are categorized into classes—a collection of things that have similar attributes and common behaviors.
- For example, a UML class diagram can be used to depict a Sensor class for the SafeHome security function .
- Note that the diagram lists the attributes of sensors (e.g., name, type) and the operations (e.g., identify, enable) that can be applied to modify these attributes.
- In addition to class diagrams, other analysis modeling elements depict the manner in which classes collaborate with one another and the relationships and interactions between classes.



Behavioral elements:

- The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied.
- Therefore, the requirements model must provide modeling elements that depict behavior.
- The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state.
- A state is any externally observable mode of behavior.
- In addition, the state diagram indicates actions (e.g., process activation) taken as a consequence of a particular event.
- To illustrate the use of a state diagram, consider software embedded within the SafeHome control panel that is responsible for reading user input.



Flow-oriented elements.

- Information is transformed as it flows through a computer-based system.
- The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.
- Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage.
- The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.
- Output may light a single LED or produce a 200-page report.
- In effect, we can create a flow model for any computer-based system, regardless of size and complexity.

Analysis Patterns:

- Anyone who has done requirements engineering on more than a few software projects begins to notice that certain problems reoccur across all projects within a specific application domain.
- These analysis patterns suggest solutions (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications.
- Geyer-Schulz and Hahsler suggest two benefits that can be associated with the use of analysis patterns:

➢ First, analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models with examples as well as a description of advantages and limitations.

➢ Second, analysis patterns facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems.

- Analysis patterns are integrated into the analysis model by reference to the pattern name.
- They are also stored in a repository so that requirements engineers can use search facilities to find and apply them.

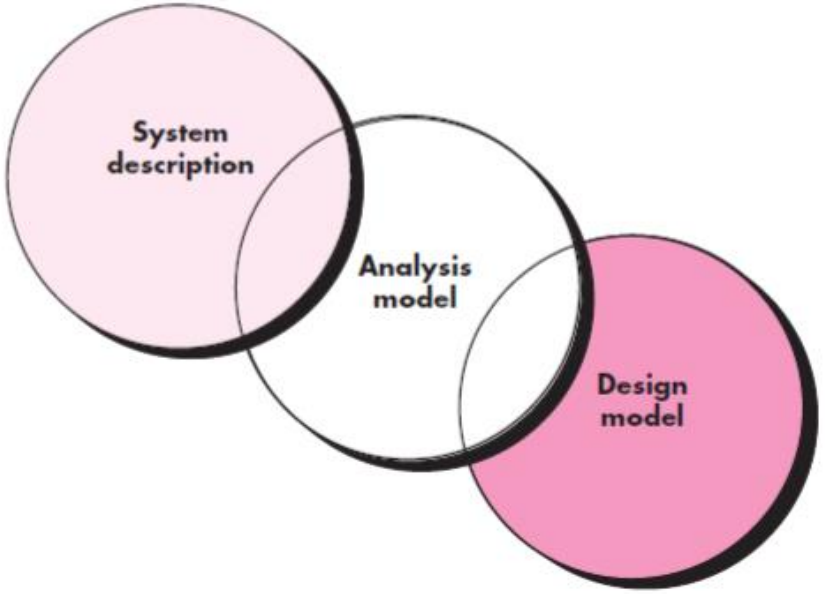
REQUIREMENTS ANALYSIS:

- Requirements analysis allows you (regardless of whether you're called a software engineer, an analyst, or a modeler) to elaborate on basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.
- The requirements modeling action results in one or more of the following types of models:

- Scenario-based models of requirements from the point of view of various system "actors".
- Data models that depict the information domain for the problem.
- Class-oriented models that represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements.
- Flow-oriented models that represent the functional elements of the system and how they transform data as it moves through the system.
- Behavioral models that depict how the software behaves as a consequence of external "events".

Overall Objectives and Philosophy:

- The requirements 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 system or business functionality as it is achieved by applying software, hardware, data, human, and other system elements and a software design that describes the software's application architecture, user interface, and component-level structure. This relationship is illustrated in Figure below.



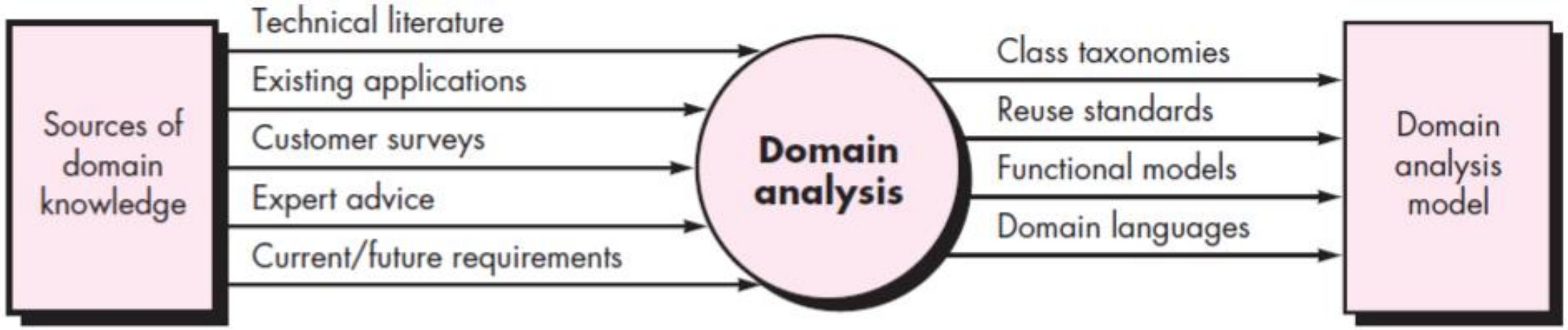
Analysis Rules of Thumb:

- Arlow and Neustadt [Arl02] suggest a number of worthwhile rules of thumb that should be followed when creating the analysis model:

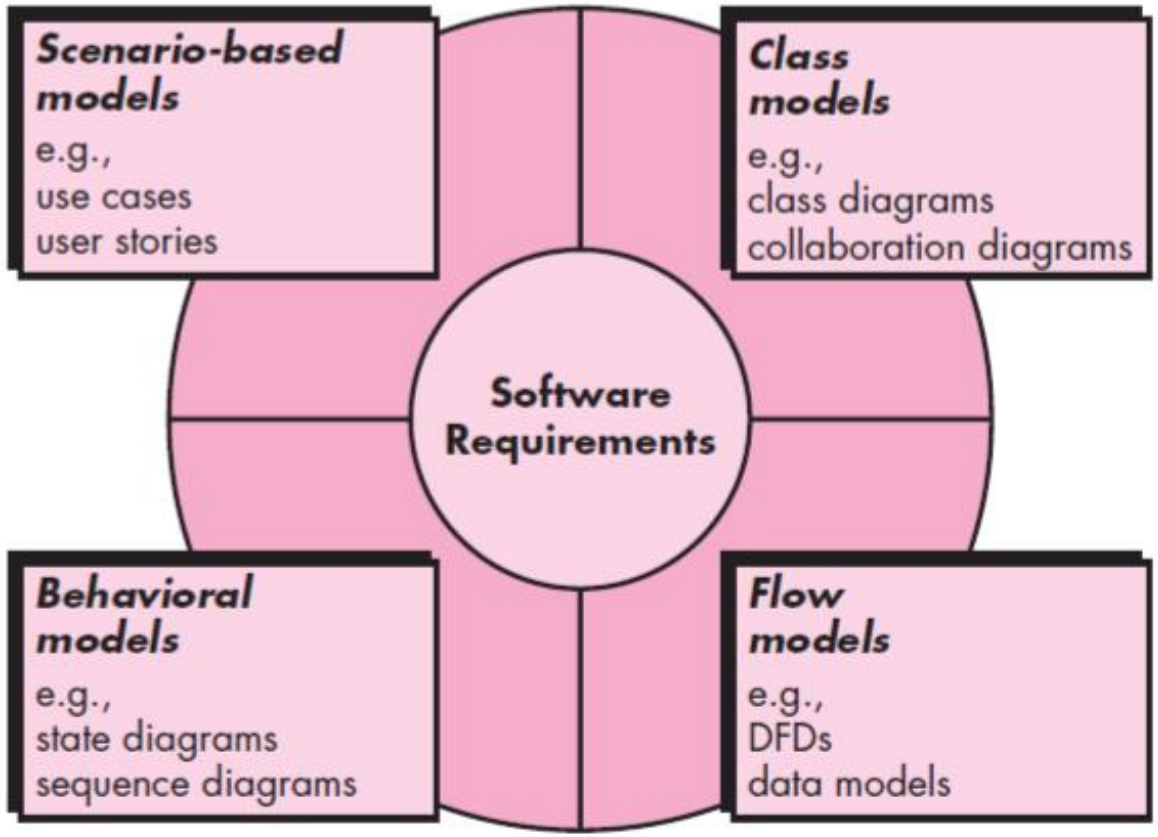
- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high. “Don’t get bogged down in details” that try to explain how the system will work.
- Each element of the requirements model should add to an overall understanding of software requirements and provide insight into the information domain, function, and behavior of the system.
- Delay consideration of infrastructure and other nonfunctional models until design. That is, 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. It is important to represent relationships between classes and functions. However, if the level of “interconnectedness” is extremely high, effort should be made to reduce it.
- Be certain that the requirements model provides value to all stakeholders. Each constituency has its own use for the model. For example, business stakeholders should use the model to validate requirements; designers should use the model as a basis for design; QA people should use the model to help plan acceptance tests.
- Keep the model as simple as it can be. Don’t create additional diagrams when they add no new information. Don’t use complex notational forms, when a simple list will do.

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.



Requirements Modeling Approaches:



SCENARIO-BASED MODELING:

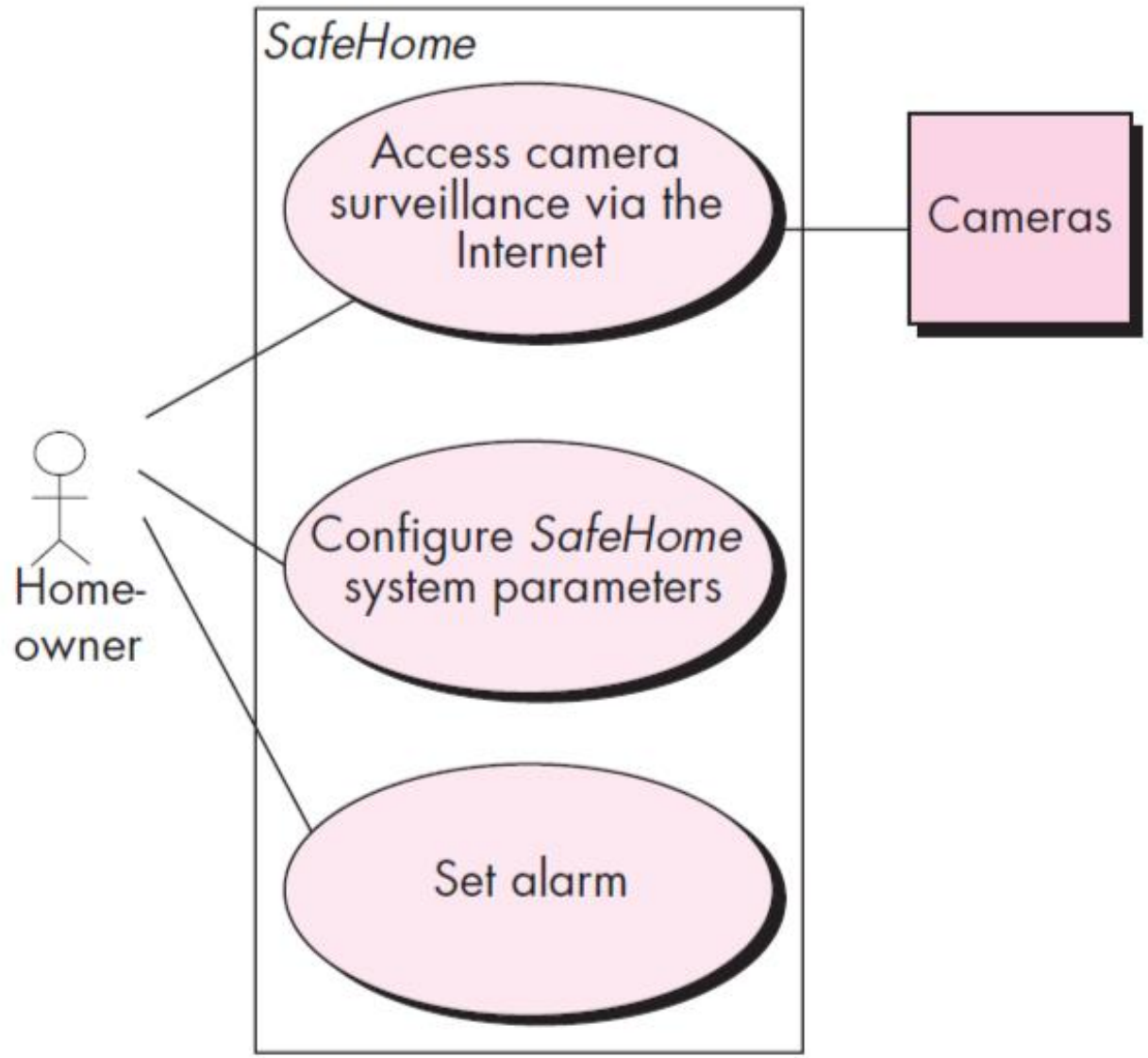
- Creating a Preliminary Use Case

- To begin developing a set of use cases, list the functions or activities performed by a specific actor.
- You can obtain these from a list of required system functions, through conversations with stakeholders, or by an evaluation of activity diagrams developed as part of requirements modeling.
- To illustrate, consider the function access camera surveillance via the Internet— display camera views (ACS-DCV). The stakeholder who takes on the role of the homeowner actor might write the following narrative that presents the interaction as an ordered sequence of user actions. Each action is represented as a declarative sentence.

- Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)
- Actor: homeowner

1. The homeowner logs onto the SafeHome Products website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the “surveillance” from the major function buttons.
6. The homeowner selects “pick a camera.”
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the “view” button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Preliminary use-case diagram for the *SafeHome* system



- Refining a Preliminary Use Case

- A description of alternative interactions is essential for a complete understanding of the function that is being described by a use case.
- Therefore, each step in the primary scenario is evaluated by asking the following questions:

- Can the actor take some other action at this point?
- Is it possible that the actor will encounter some error condition at this point? If so, what might it be?
- Is it possible that the actor will encounter some other behavior at this point (e.g., behavior that is invoked by some event outside the actor's control)? If so, what might it be?

- Cockburn recommends using a “brainstorming” session to derive a reasonably complete set of exceptions for each use case. In addition to the three generic questions suggested earlier in this section, the following issues should also be explored:

- Are there cases in which some “validation function” occurs during this use case? This implies that validation function is invoked and a potential error condition might occur.
- Are there cases in which a supporting function (or actor) will fail to respond appropriately? For example, a user action awaits a response but the function that is to respond times out.
- Can poor system performance result in unexpected or improper user actions? For example, a Web-based interface responds too slowly, resulting in a user making multiple selects on a processing button. These selects queue inappropriately and ultimately generate an error condition.

- Writing a Formal Use Case

- The informal use cases are sometimes sufficient for requirements modeling.
- However, when a use case involves a critical activity or describes a complex set of steps with a significant number of exceptions, a more formal approach may be desirable.

SAFEHOME

Use Case Template for Surveillance

Use case: Access camera surveillance via the Internet—display camera views (ACS-DCV)

Iteration: 2, last modification: January 14 by V. Raman.

Primary actor: Homeowner.

Goal in context: To view output of camera placed throughout the house from any remote location via the Internet.

Preconditions: System must be fully configured; appropriate user ID and passwords must be obtained.

Trigger: The homeowner decides to take a look inside the house while away.

Scenario:

1. The homeowner logs onto the *SafeHome Products* website.
2. The homeowner enters his or her user ID.
3. The homeowner enters two passwords (each at least eight characters in length).
4. The system displays all major function buttons.
5. The homeowner selects the "surveillance" from the major function buttons.
6. The homeowner selects "pick a camera."
7. The system displays the floor plan of the house.
8. The homeowner selects a camera icon from the floor plan.
9. The homeowner selects the "view" button.
10. The system displays a viewing window that is identified by the camera ID.
11. The system displays video output within the viewing window at one frame per second.

Exceptions:

1. ID or passwords are incorrect or not recognized—see use case **Validate ID and passwords.**
2. Surveillance function not configured for this system—system displays appropriate error message; see use case **Configure surveillance function.**
3. Homeowner selects "View thumbnail snapshots for all camera"—see use case **View thumbnail snapshots for all cameras.**
4. A floor plan is not available or has not been configured—display appropriate error message and see use case **Configure floor plan.**
5. An alarm condition is encountered—see use case **Alarm condition encountered.**

Priority: Moderate priority, to be implemented after basic functions.

When available: Third increment.

Frequency of use: Moderate frequency.

Channel to actor: Via PC-based browser and Internet connection.

Secondary actors: System administrator, cameras.

Channels to secondary actors:

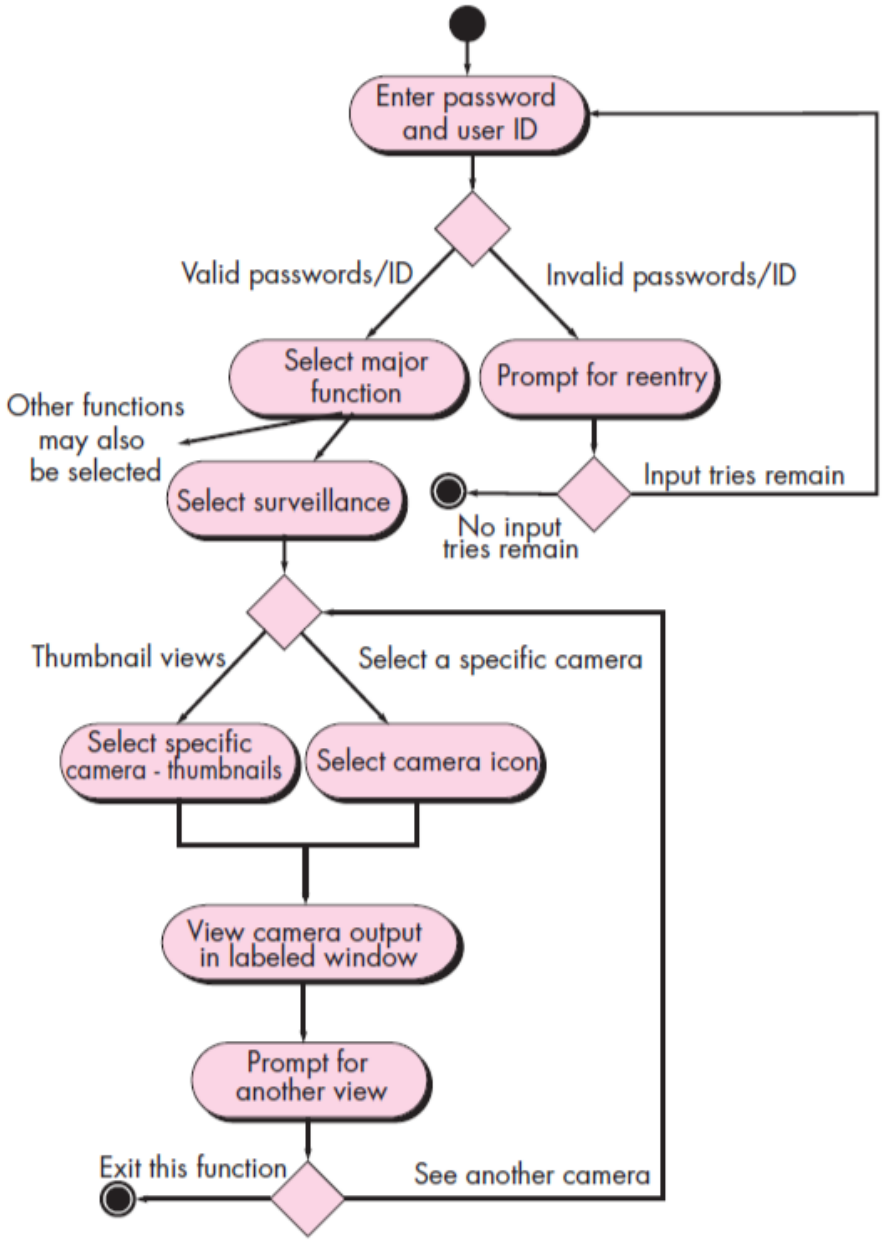
1. System administrator: PC-based system.
2. Cameras: wireless connectivity.

Open issues:

1. What mechanisms protect unauthorized use of this capability by employees of *SafeHome Products*?
2. Is security sufficient? Hacking into this feature would represent a major invasion of privacy.
3. Will system response via the Internet be acceptable given the bandwidth required for camera views?
4. Will we develop a capability to provide video at a higher frames-per-second rate when high-bandwidth connections are available?

UML MODELS THAT SUPPLEMENT THE USE CASE

- Developing an Activity Diagram



DATA MODELING CONCEPTS

- Data Objects

- A data object is a representation of composite information that must be understood by software.
- By composite information, I mean something that has a number of different properties or attributes.
- Therefore, width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object.
- A data object can be an external entity (e.g., anything that produces or consumes information), a thing (e.g., a report or a display), an occurrence (e.g., a telephone call) or event (e.g., an alarm), a role (e.g., salesperson), an organizational unit (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file).
- For example, a person or a car can be viewed as a data object in the sense that either can be defined in terms of a set of attributes.
- A data object encapsulates data only—there is no reference within a data object to operations that act on the data.

- Data Attributes

- Data attributes define the properties of a data object and take on 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—that is, the identifier attribute becomes a “key” when we want to find an instance of the data object.
- In some cases, 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.

Instance

Naming attributes

Ties one data object to another, in this case, owner

Identifier

Descriptive attributes

Referential attributes

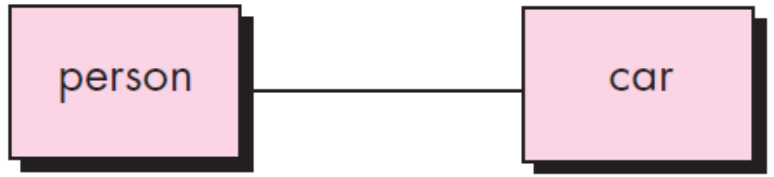
Make	Model	ID#	Body type	Color	Owner
Lexus	LS400	AB123. . .	Sedan	White	RSP
Chevy	Corvette	X456. . .	Sports	Red	CCD
BMW	750iL	XZ765. . .	Coupe	White	LJL
Ford	Taurus	Q12A45. . .	Sedan	Blue	BLF

- Relationships

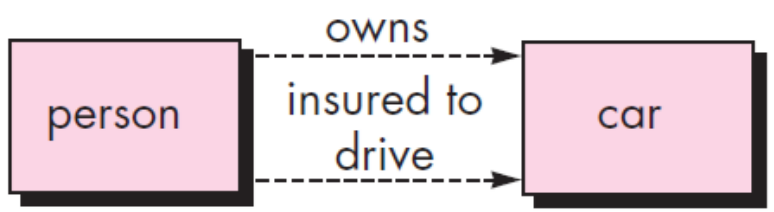
- Data objects are connected to one another in different ways.
- For example,

➢ A person owns a car.

➢ A person is insured to drive a car.



(a) A basic connection between data objects



(b) Relationships between data objects

CLASS-BASED MODELING:

- Identifying Analysis Classes

- We can begin to identify classes by examining the usage scenarios developed as part of the requirements model and performing a “grammatical parse” on the use cases developed for the system to be built.
- Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
- 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, 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.

Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

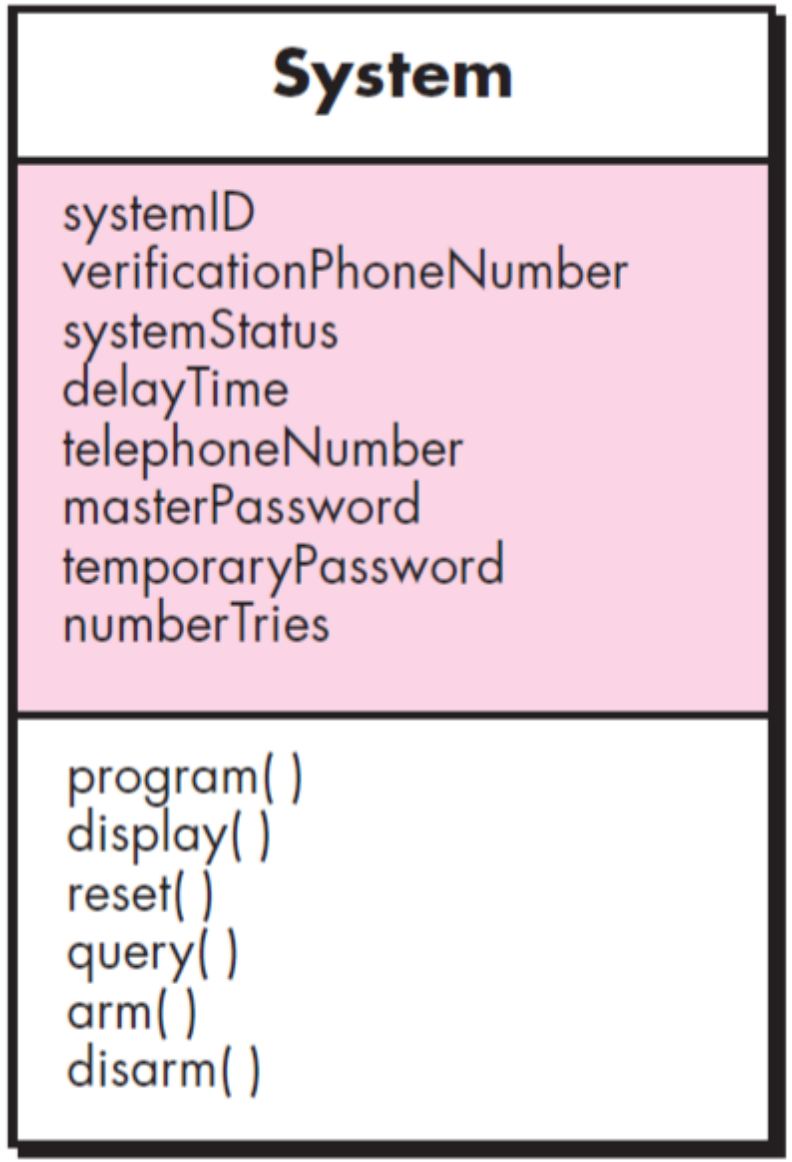
- Specifying Attributes

- Attributes describe a class that has been selected for inclusion in the requirements model.
- In essence, it is the attributes that define the class—that clarify what is meant by the class in the context of the problem space.

- Defining Operations

- Operations define the behavior of an object. Although many different types of operations exist, they can generally be divided into four broad categories: (1) operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting), (2) operations that perform a computation, (3) operations that inquire about the state of an object, and (4) operations that monitor an object for the occurrence of a controlling event.
- These functions are accomplished by operating on attributes and/or associations

Class diagram for the system class



Class diagram for FloorPlan (see sidebar discussion)