



A T M E
College of Engineering



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

MODULE-2



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

MODULE-2

12 hours

Understanding Requirements: Requirements Engineering, Establishing the ground work, Eliciting Requirements, Developing use cases, Building the requirements model, Negotiating Requirements, Validating Requirements.

Requirements Modeling Scenarios, Information and Analysis classes: Requirement Analysis, Scenario based modeling, UML models that supplement the Use Case, Data modeling Concepts, Class-Based Modeling.

Requirement Modeling Strategies : Flow oriented Modeling , Behavioral Modeling.

5.1 UNDERSTANDING REQUIREMENTS REQUIREMENTS ENGINEERING

- ❑ The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering
- ❑ From a software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity.
- ❑ Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.

Requirement encompasses seven distinct tasks:

1. Inception
 2. Elicitation
 3. Elaboration
 4. Negotiation
 5. Specification
 6. Validation
 7. Requirement Management.
- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.

1.Inception.

- How does a software project get started? Is there a single event that becomes the catalyst for a new computer-based system or product, or does the need evolve over time? There are no definitive answers to these questions.
- In some cases, a casual conversation is all that is needed to precipitate a major software engineering effort.
- But in general, most projects begin when a business need is identified or a potential new market or service is discovered.
- Stakeholders from the business community define a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis, and identify a working description of the project's scope.
- At project inception, you establish a ***basic understanding of the problem***, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

2.Elicitation

- It certainly seems simple enough—ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day-to-day basis. But it isn't simple—it's very hard. **Christel and Kang** identify a number of problems that are encountered as elicitation occurs
1. ***Problems of scope.*** The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
 2. ***Problems of understanding.*** The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be “obvious,” specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or untestable.
 3. ***Problems of volatility.*** The requirements change over time.

3.Elaboration

- ❖ The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- ❖ This task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information.
- ❖ Elaboration is driven by the creation and refinement of user scenarios that describe how the end user will interact with the system.
- ❖ Each user scenario is parsed to extract analysis classes—business domain entities that are visible to the end user.
- ❖ The attributes of each analysis class are defined, and the services that are required by each class are identified.
- ❖ The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

4.Negotiation

- ✓ It isn't unusual for customers and users to ask for more than can be achieved, given limited business resources.
- ✓ It's also relatively common for different customers or users to propose conflicting requirements, arguing that their version is “essential for our special needs.”
- ✓ You have to reconcile these conflicts through a process of negotiation.
- ✓ Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

5.Specification

- ❖ In the context of computer-based systems , the term specification means different things to different people.
- ❖ A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- ❖ Some suggest that a “standard template” should be developed and used for a specification, arguing that this leads to requirements that are presented in a consistent and therefore more understandable manner.
- ❖ However, it is sometimes necessary to remain flexible when a specification is to be developed. For large systems, a written document, **combining natural language descriptions and graphical models** may be the best approach.

6.Validation.

- The work products produced as a consequence of requirements engineering are assessed for quality during a validation step.
- Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.
- The primary requirements validation mechanism is the technical review .The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies conflicting requirements, or unrealistic requirements.

7.Requirements management.

- Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- Requirements management is a set of activities that help the project team **identify, control, and track** requirements and changes to requirements at any time as the project proceeds.
- Many of these activities are identical to the software configuration management (SCM).

5.2 ESTABLISHING THE GROUNDWORK

Steps required to establish the ground work

- 1 Identifying Stakeholders
- 2 Recognizing Multiple Viewpoints
- 3 Working toward Collaboration
- 4 Asking the First Questions

5.2.1 Identifying Stakeholders

Definition of a stakeholder

“anyone who benefits in a direct or indirect way from the system which is being developed.”

- ❑ Identified the usual suspects: business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers, and others.
- ❑ Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.

5.2.2 Recognizing Multiple Viewpoints

Because many different stakeholders exist, the requirements of the system will be explored from many different points of view.

For example

- **The marketing group** is interested in functions and features that will excite the potential market, making the new system easy to sell.
- **Business managers** are interested in a feature set that can be built within budget and that will be ready to meet defined market windows
- **End users** may want features that are familiar to them and that are easy to learn and use.
- **Software engineers** may be concerned with functions that are invisible to nontechnical stakeholders but that enable an infrastructure that supports more marketable functions and features.
- **Support engineers** may focus on the maintainability of the software.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Each of these constituencies will contribute information to the requirements engineering process.
- As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.
- You should categorize all stakeholder information in a way that will allow decision makers to choose an internally consistent set of requirements for the system.

5.2.3 Working toward Collaboration

- If five stakeholders are involved in a software project, you may have five different opinions about the proper set of requirements.
- Customers must collaborate among themselves and with software engineering practitioners if a successful system is to result. But how is this collaboration accomplished?
- The job of a requirements engineer is to identify areas of commonality and areas of conflict or inconsistency.
- Collaboration does not necessarily mean that requirements are defined by committee. In many cases, stakeholders collaborate by providing their view of requirements, but a strong “project champion” may make the final decision about which requirements make the cut.

5.2.4 Asking the First Questions

- Questions asked at the inception of the project should be “**context free**” .
- The first set of context-free questions **focuses on the customer and other stakeholders**, the overall project goals and benefits.
- For example, you might ask:
 1. *Who is behind the request for this work?*
 2. *Who will use the solution?*
 3. *What will be the economic benefit of a successful solution?*
 4. *Is there another source for the solution that you need?*

These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:
 1. *How would you characterize “good” output that would be generated by a successful solution?*
 2. *What problem(s) will this solution address?*
 3. *Can you show me (or describe) the business environment in which the solution will be used?*
 4. *Will special performance issues or constraints affect the way the solution is approached?*

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The final set of questions **focuses on the effectiveness of the communication activity itself.**

Gause and Weinberg call these “meta-questions” and propose the following list:

1. Are you the right person to answer these questions? Are your answers “official”?
2. Are my questions relevant to the problem that you have?
3. Am I asking too many questions?
4. Can anyone else provide additional information?
5. Should I be asking you anything else?

These questions will help to “break the ice” and initiate the communication that is essential to successful elicitation.

But a question-and-answer meeting format is not an approach that has been overwhelmingly successful.

In fact, the Q&A session should be used for the first encounter only and then replaced by a requirements elicitation format that combines elements of problem solving, negotiation, and specification.

5.3 ELICITING REQUIREMENTS

- Requirements elicitation also called **requirements gathering** combines elements of **problem solving, elaboration, negotiation, and specification.**
- In order to encourage a collaborative, team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements .

5.3.1 Collaborative Requirements Gathering

Many different approaches to collaborative requirements gathering have been proposed. Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

1. Meetings are conducted and attended by both software engineers and other stakeholders.
2. Rules for preparation and participation are established.
3. An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
4. A “facilitator” (can be a customer, a developer, or an outsider) controls the meeting.
5. A “definition mechanism” (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal.
- During inception basic questions and answers establish the scope of the problem and the overall perception of a solution. Out of these initial meetings, the developer and customers write a one- or two-page “**product request.**”
- A meeting place, time, and date are selected; a **facilitator** is chosen; and attendees from the software team and other stakeholder organizations are invited to participate. The product request is distributed to all attendees before the meeting date.

Example

- Consider a product request written by a marketing person involved in the SafeHome project.
- This person writes the following narrative about the home security function that is to be part of SafeHome.
- Our research indicates that the market for home management systems is growing at a rate of 40 percent per year.
- The first SafeHome function we bring to market should be the **home security function**. Most people are familiar with “**alarm systems**” so this would be an easy sell.
- The home security function would protect against and/or recognize a variety of **undesirable “situations”** such as **illegal entry, fire, flooding, carbon monoxide levels, and others**. It’ll use our **wireless sensors** to detect each situation.
- It can be programmed by the homeowner, and will automatically **telephone a monitoring agency** when a situation is detected.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- While reviewing the product request in the days before the meeting, each attendee is asked to make a list of objects that are part of the environment that surrounds the system, other objects that are to be produced by the system, and objects that are used by the system to perform its functions.
- Finally, lists of **constraints** (e.g., cost, size, business rules) and **performance criteria** (e.g., speed, accuracy) are also developed.
- The attendees are informed that the lists are not expected to be exhaustive but are expected to reflect each person's perception of the system.

- **Objects** described for SafeHome might include the *control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event, a display, a PC, telephone numbers, a telephone call, and so on.*
- The list of **services** might include *configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel, and reading the display.*
- In a similar fashion, each attendee will develop lists of constraints and performance criteria.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- The **lists of objects** can be pinned to the walls of the room using *large sheets of paper, stuck to the walls using adhesive-backed sheets, or written on a wall board.*
- The lists may have been posted on an electronic **bulletin board**, at an **internal website**, or posed in a **chat room** environment for review prior to the meeting.
- Each listed entry should be capable of being manipulated separately so that lists can be combined, entries can be modified, and additions can be made.
- At this stage, critique and debate are strictly prohibited.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- After individual lists are presented in one topic area, the group creates a combined list by eliminating redundant entries, adding any new ideas that come up during the discussion, but not deleting anything.
- After you create combined lists for all topic areas, discussion—coordinated by the facilitator—ensues.
- The combined list is shortened, lengthened, or reworded to properly reflect the product/system to be developed.
- The objective is to develop a consensus list of objects, services, constraints, and performance for the system to be built.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- In many cases, an object or service described on a list will require further explanation.
- To accomplish this, stakeholders develop mini-specifications for entries on the lists.
- Each mini-specification is an elaboration of an object or service.
- The mini-specs are presented to all stakeholders for discussion. Additions, deletions, and further elaboration are made.
- For example
 - The mini-specification for the safehome object **control panel** might be: the control panel is a wall-mounted unit that is approximately 9*5 inches in size. The control panel has wireless connectivity to sensors and a PC. User interaction occurs through a keypad containing 12 keys. A 3*3 inch LCD color display provides user feedback. Software provides interactive prompts, echo, and similar functions.

- In some cases, the development of mini-specs will uncover new objects, services, constraints, or performance requirements that will be added to the original lists.
- During all discussions, the team may raise an issue that cannot be resolved during the meeting.
- An issues list is maintained so that these ideas will be acted on later.

5.3.2 Quality Function Deployment

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- QFD “concentrates on maximizing customer satisfaction from the software engineering process”. To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.

QFD identifies three types of requirements:

1. Normal requirements.
2. Expected requirements
3. Exciting requirements.

1. Normal requirements

- The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied.
- Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.

2. Expected requirements

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them.
- Their absence will be a cause for significant dissatisfaction.
- Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation

3. Exciting requirements

- These features go beyond the customer's expectations and prove to be very satisfying when present.
- For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail) that delight every user of the product.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Although QFD concepts can be applied across the entire software process, specific QFD techniques are applicable to the requirements elicitation activity.
- QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity.
- These data are then translated into a table of requirements—called the **customer voice table**—that is reviewed with the customer and other stakeholders.
- A variety of diagrams, matrices, and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements.

5.3.3 Usage Scenarios

- As requirements are gathered, an overall vision of system functions and features begins to materialize.
- It is difficult to move into more technical software engineering activities until you understand how these functions and features will be used by different classes of end users.
- To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called **use cases**.
- provide a description of how the system will be used.

5.3.4 Elicitation Work Products

The work products include

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.
- A list of requirements and the domain constraints that apply to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements

5.4 DEVELOPING USE CASES

- The first step in writing a use case is to define the set of “actors” that will be involved in the story.
- Actors are the different people (or devices) that use the system or product within the context of the function and behavior that is to be described.
- Actors represent the roles that people (or devices) play as the system operates.
- Defined somewhat more formally, an actor is anything that communicates with the system or product and that is external to the system itself.
- Every actor has one or more goals when using the system

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- It is important to note that an actor and an end user are not necessarily the same thing.
- A typical user may play a number of different roles when using a system, whereas an actor represents a class of external entities that play just one role in the context of the use case.
- As an example, consider a machine operator (a user) who interacts with the control computer for a manufacturing cell that contains a number of robots and numerically controlled machines.
- After careful review of requirements, the software for the control computer requires four different modes (roles) for interaction:
 - programming mode,
 - test mode
 - monitoring mode
 - troubleshooting mode.
- Therefore, four actors can be defined: programmer, tester, monitor, and troubleshooter. In some cases, the machine operator can play all of these roles. In others, different people may play the role of each actor.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Because requirements elicitation is an evolutionary activity, not all actors are identified during the first iteration.
- It is possible to identify primary actors during the first iteration and secondary actors as more is learned about the system.
- Primary actors interact to achieve required system function and derive the intended benefit from the system.
- They work directly and frequently with the software. Secondary actors support the system so that primary actors can do their work.

Once actors have been identified, use cases can be developed. Jacobson suggests a number of questions that should be answered by a use case:

- 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

Ex : Safehome project

- Recalling basic SafeHome requirements
- we define four actors:
 - Homeowner (a user)
 - Setup manager (likely the same person as homeowner, but playing a different role)
 - Sensors (devices attached to the system)
 - The monitoring and response subsystem (the central station that monitors the safehome home security function).

For the purposes of this example, we consider only the homeowner actor. The homeowner actor interacts with the home security function in a number of different ways using either the alarm control panel or a PC.

- Enters a password to allow all other interactions.
- Inquires about the status of a security zone.
- Inquires about the status of a sensor.
- Presses the panic button in an emergency.
- Activates/deactivates the security system.

Considering the situation in which the homeowner uses the control panel, the basic use case for system activation follows:

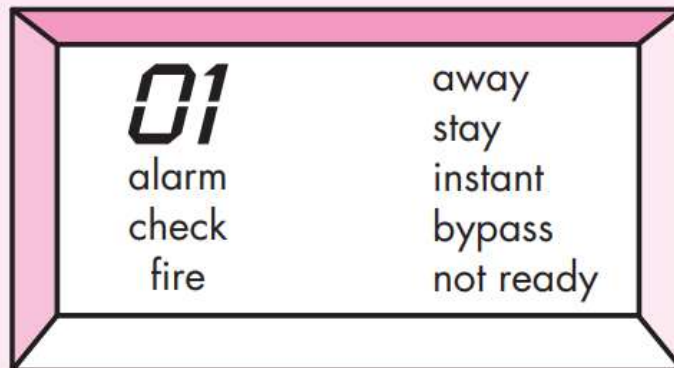
1. The homeowner observes the SafeHome control panel given in the Figure 5.1 to determine if the system is ready for input.

If the system is not ready, a not ready message is displayed on the LCD display, and the homeowner must physically close windows or doors so that the not ready message disappears. [A not ready message implies that a sensor is open; i.e., that a door or window is open].



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

SAFEHOME



armed



power



off	away	stay
1	2	3
max	test	bypass
4	5	6
instant	code	chime
7	8	9
ready		
*	0	#
panic		

2. The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect, the control panel will beep once and reset itself for additional input. If the password is correct, the control panel awaits further action.
3. The homeowner selects and keys in stay or away to activate the system. Stay activates only perimeter sensors (inside motion detecting sensors are deactivated). Away activates all sensors.
4. When activation occurs, a red alarm light can be observed by the homeowner.

Cockburn suggests the following template for detailed descriptions of use cases:

- ***Use case:*** Initiate Monitoring
- ***Primary actor:*** Homeowner.
- ***Goal in context:*** To set the system to monitor sensors when the homeowner leaves the house or remains inside.
- ***Preconditions:*** System has been programmed for a password and to recognize various sensors.
- ***Trigger:*** The homeowner decides to “set” the system, i.e., to turn on the alarm functions

Scenario:

1. Homeowner: observes control panel
2. Homeowner: enters password
3. Homeowner: selects “stay” or “away”
4. Homeowner: observes read alarm light to indicate that SafeHome has been armed

Exceptions:

- 1. Control panel is not ready:** homeowner checks all sensors to determine which are open; closes them.
- 2. Password is incorrect** (control panel beeps once): homeowner reenters correct password.
- 3. Password not recognized:** monitoring and response subsystem must be contacted to reprogram password.
- 4. Stay is selected:** control panel beeps twice and a stay light is lit; perimeter sensors are activated.
- 5. Away is selected:** control panel beeps three times and an away light is lit; all sensors are activated.

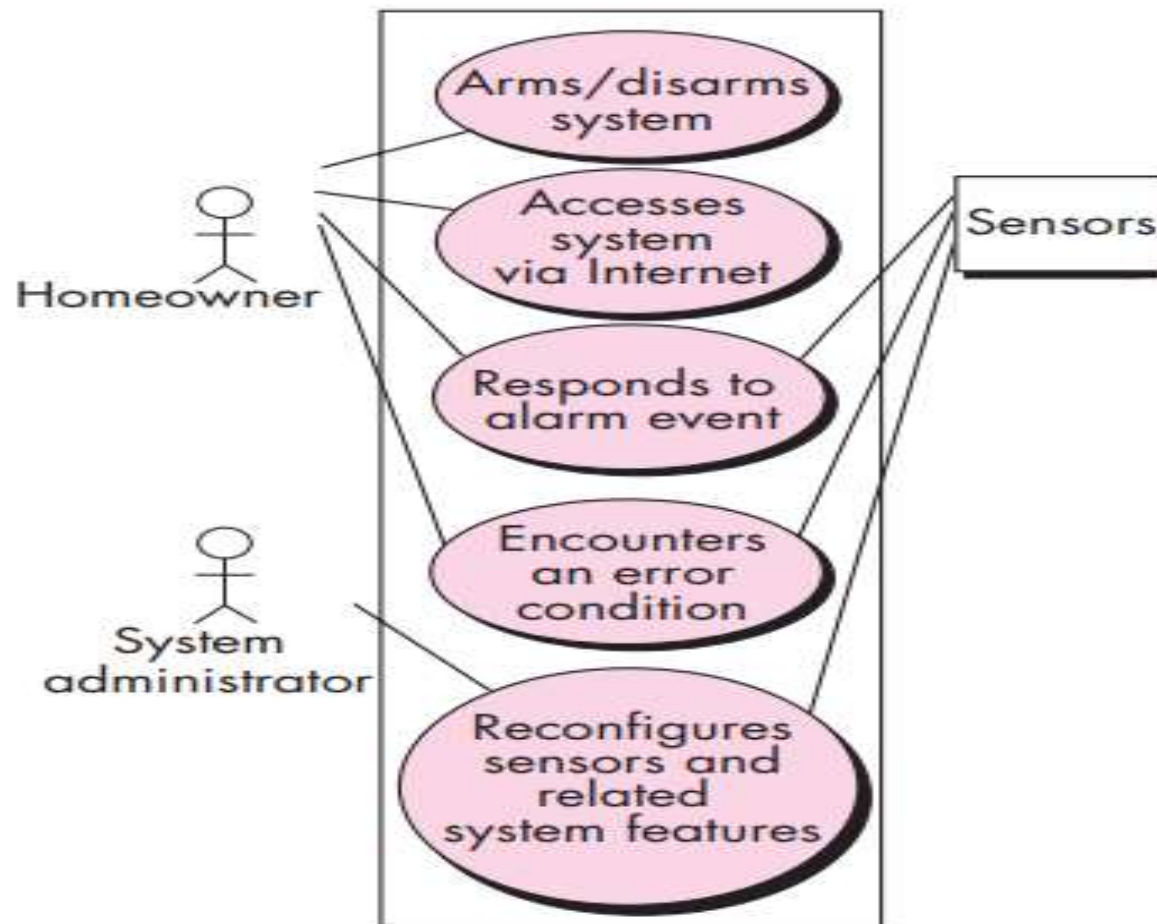
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Priority: Essential, must be implemented
- When available: First increment
- Frequency of use: Many times per day
- Channel to actor: Via control panel interface
- Secondary actors: Support technician, sensors
- Channels to secondary actors:
 - Support technician: phone line
 - Sensors: hardwired and radio frequency interfaces

Open issues:

1. Should there be a way to activate the system without the use of a password or with an abbreviated password?
2. Should the control panel display additional text messages?
3. How much time does the homeowner have to enter the password from the time the first key is pressed?
4. Is there a way to deactivate the system before it actually activates?

UML use case diagram for SafeHome home security function



5.5 BUILDING THE REQUIREMENTS MODEL

- As the requirements model evolves, certain elements will become relatively stable, providing a solid foundation for the design tasks that follow.
- However, other elements of the model may be more volatile, indicating that stakeholders do not yet fully understand requirements for the system.

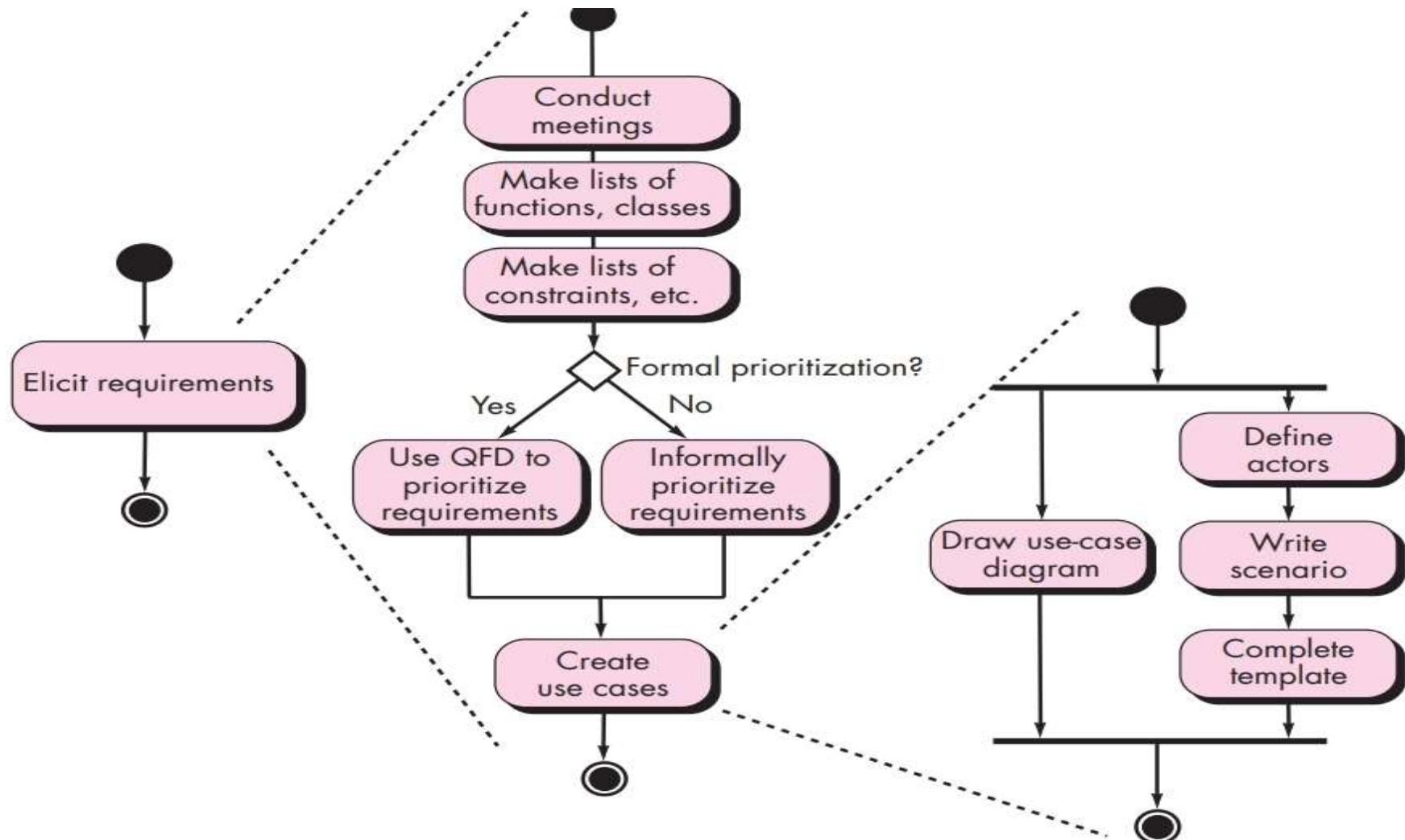
5.5.1 Elements of the Requirements Model

- ❑ 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.

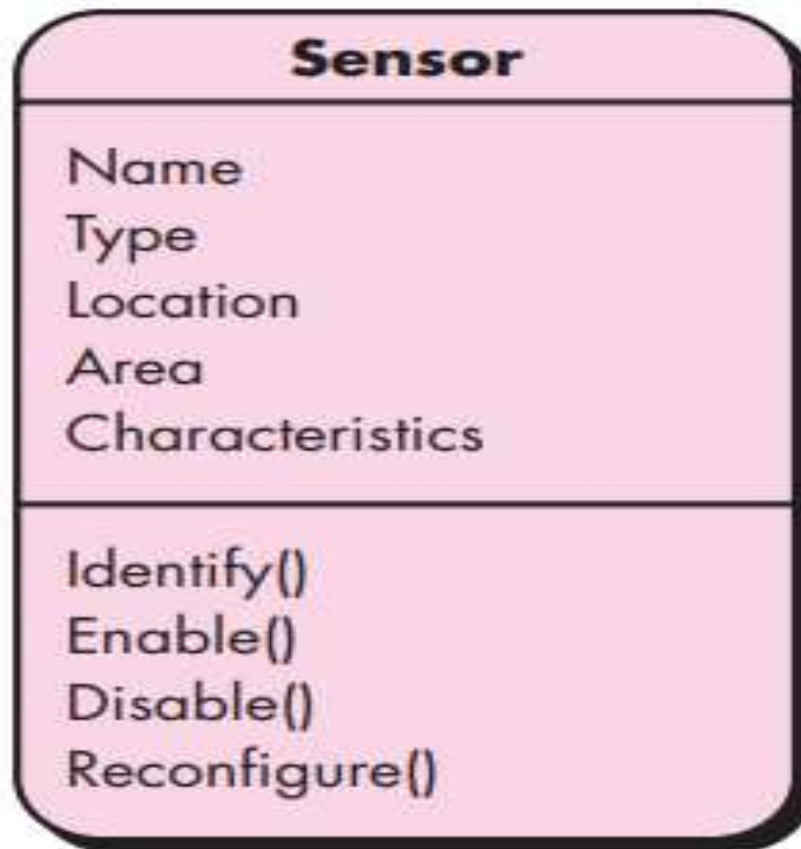
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML



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.

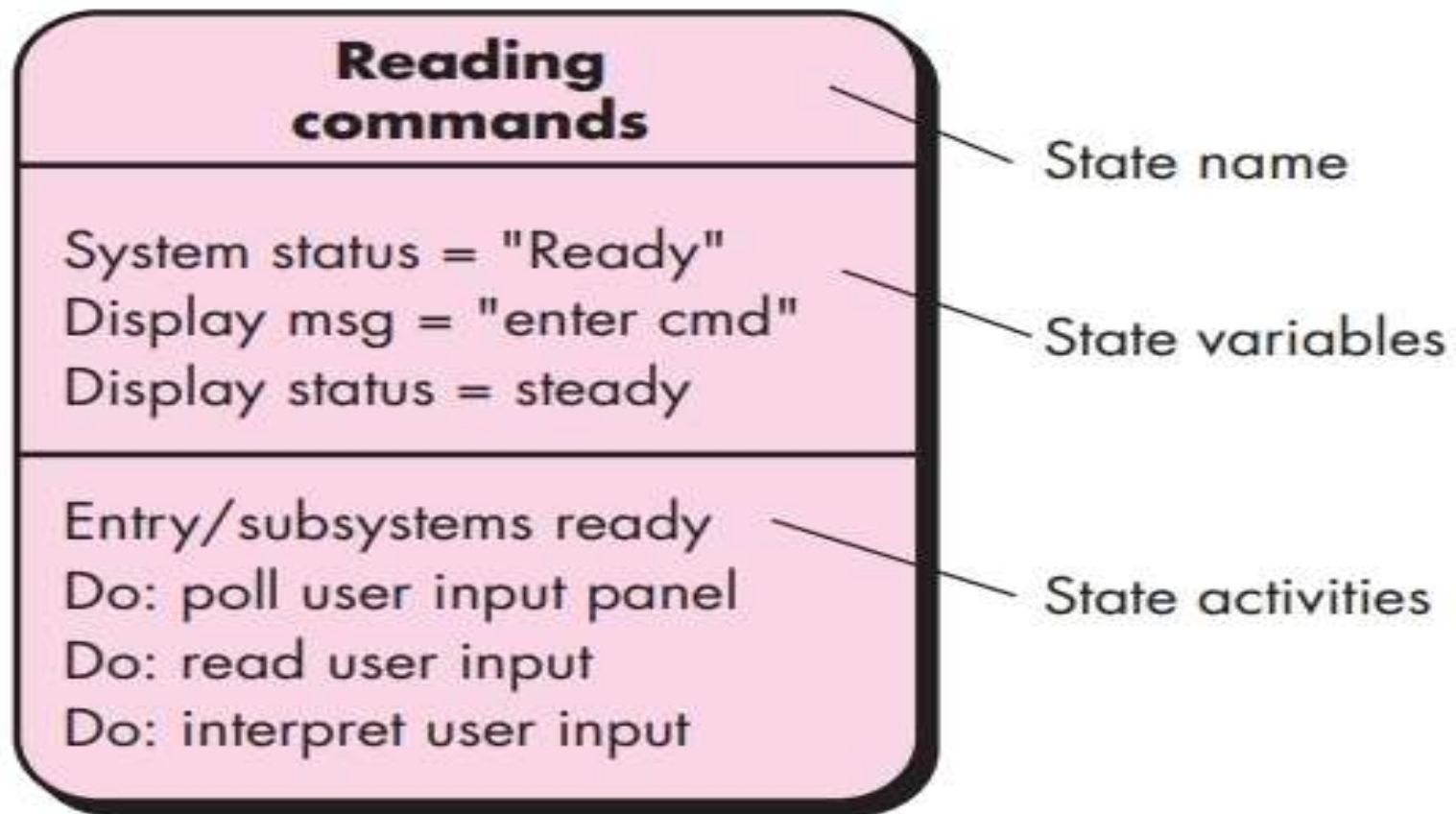
Class diagram for sensor



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.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML



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.

5.5.2 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.
- 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.

5.6 NEGOTIATING REQUIREMENTS

Boehm defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined

1. Identification of the system or subsystem's key stakeholders.
2. Determination of the stakeholders' "win conditions."
3. Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team)

5.7 VALIDATING REQUIREMENTS

- **A review of the requirements model addresses the following questions**
- 1. Is each requirement consistent with the overall objectives for the system/product?
- 2. Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- 3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- 4. Is each requirement bounded and unambiguous?
- 5. Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

6. Do any requirements conflict with other requirements?
7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?
9. Does the requirements model properly reflect the information, function, and behavior of the system to be built?
10. Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?
11. Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements?

MODULE-2

- CHAPTER-2
- Requirements Modeling Scenarios, Information and Analysis classes:

6.1 REQUIREMENTS ANALYSIS

- Requirements analysis allows us 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

1. **Scenario-based models** of requirements from the point of view of various system “actors”
2. **Data models** that depict the information domain for the problem
3. **Class-oriented models** that represent object-oriented classes and the manner in which classes collaborate to achieve system requirements
4. **Flow-oriented models** that represent the functional elements of the system and how they transform data as it moves through the system
5. **Behavioral models** that depict how the software behaves as a consequence of external “events.”



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- These models provide a software designer with information that can be translated to architectural, interface, and component-level designs.
- Finally, the requirements model provides the developer and the customer with the means to assess quality once software is built.

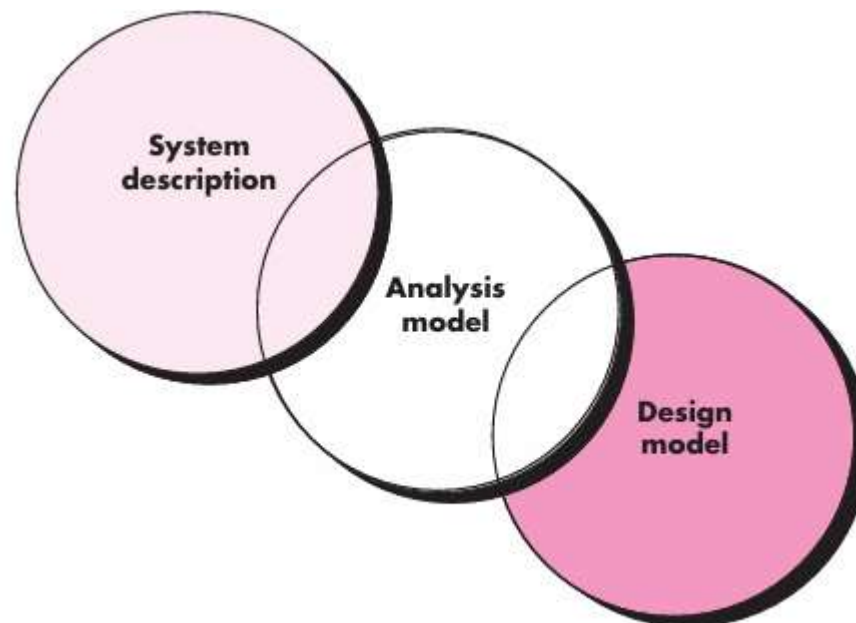
6.1.1 Overall Objectives and Philosophy

- Throughout requirements modeling, our primary focus is on **what**, not how. What **user interaction** occurs in a particular circumstance, what **objects** does the system manipulate, what **functions** must the system perform, what **behaviors** does the system exhibit, what interfaces are defined, and what **constraints** apply?

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
- (3) to define a set of requirements that can be validated once the software is built.

The requirements model as a bridge between the system description and the design model



6.1.2 Analysis Rules of Thumb

- Arlow and Neustadt suggest a number of worthwhile rules of thumb that should be followed when creating the analysis model
1. 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.
 2. 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.
 3. 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.

Cont.....

4. 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.
5. Be certain that the requirements model provides value to all stakeholders. Each constituency has its own use for the model. For example, business stake holders 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.
6. 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.

Definition

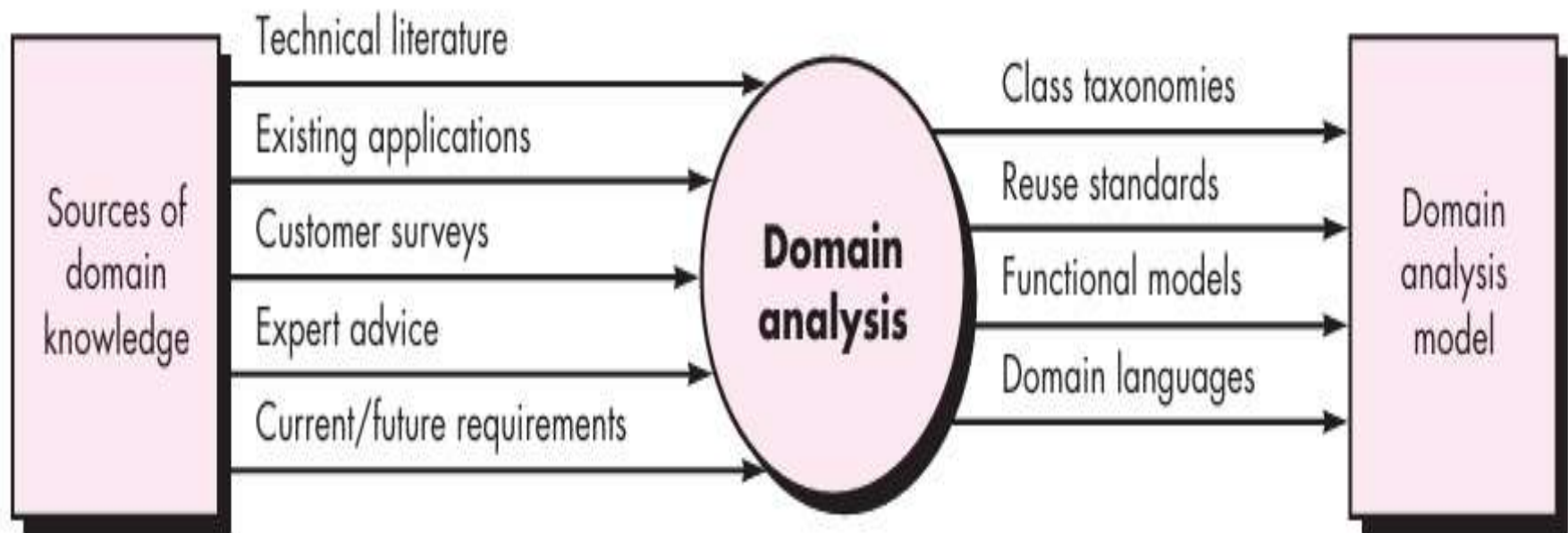
- Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain. . . .the identification, analysis, and specification of common, reusable capabilities within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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- domain analysis may be viewed as an umbrella activity for the software process.
- The role of the domain analyst is to discover and define analysis patterns, analysis classes, and related information that may be used by many people working on similar but not necessarily the same applications.

6.1.3 Domain Analysis

FIGURE 6.2 Input and output for domain analysis

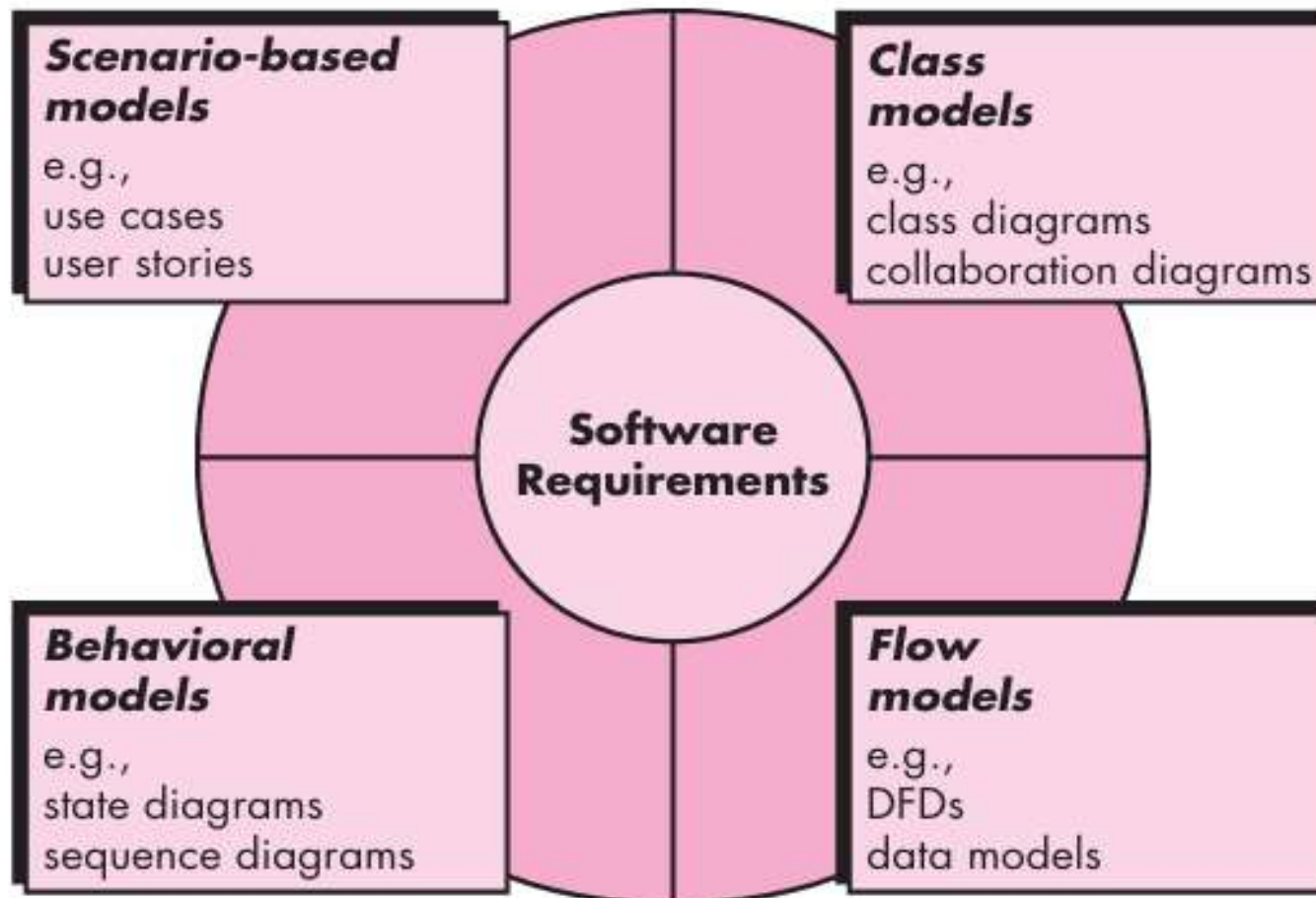


illustrates key inputs and outputs for the domain analysis process. Sources of domain knowledge are surveyed in an attempt to identify objects that can be reused across the domain

6.1.4 Requirements Modeling Approaches

- One view of requirements modeling, called **structured analysis**, considers data and the processes that transform the data as separate entities.
- Data objects are modeled in a way that defines their attributes and relationships. Processes that manipulate data objects are modeled 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 effect customer requirements.

Elements of the analysis model



Each element of the requirements model presents the problem from a different point of view.

1. **Scenario-based elements** depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.
2. **Class-based elements** model the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships (some hierarchical) between the objects, and the collaborations that occur between the classes that are defined.
3. **Behavioral elements** depict how external events change the state of the system or the classes that reside within it.
4. **flow-oriented elements** represent the system as an information transform, depicting how data objects are transformed as they flow through various system functions.

6.2 SCENARIO-BASED MODELING

6.2.1 Creating a Preliminary Use Case

- Alistair Cockburn characterizes a use case as a “contract for behavior”, the “contract” defines the way in which an actor uses a computer-based system to accomplish some goal.
- A use case captures the interactions that occur between producers and consumers of information and the system itself
- These are the questions that must be answered if use cases are to provide value as a requirements modeling tool.
 - (1) What to write about
 - (2) How much to write about it
 - (3) How detailed to make your description
 - (4) How to organize the description?

What to write about?

- The first two requirements engineering task
 - 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. 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.

The SafeHomehome surveillance function (subsystem) discussed in the sidebar identifies the following functions (an abbreviated list) that are performed by the homeowner actor

1. Select camera to view.
2. Request thumbnails from all cameras.
3. Display camera views in a PC window.
4. Control pan and zoom for a specific camera.
5. Selectively record camera output.
6. Replay camera output.
7. Access camera surveillance via the Internet

6.2.2 Refining a Preliminary Use Case

Each step in the primary scenario is evaluated by asking the following questions

1. Can the actor take some other action at this point?
2. Is it possible that the actor will encounter some error condition at this point? If so, what might it be?
3. 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

Answers to these questions result in the creation of a set of secondary scenarios that are part of the original use case but represent alternative behavior.



- **Can the actor take some other action at this point?**
The answer is “yes.” Referring to the free-flowing narrative, the actor may choose to view thumbnail snapshots of all cameras simultaneously. Hence, one secondary scenario might be “View thumbnail snapshots for all cameras.”

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Is it possible that the actor will encounter some error condition at this point?
- Any number of error conditions can occur as a computer-based system operates.
- In this context, we consider only error conditions that are likely as a direct result of the action described in step 6 or step 7. Again the answer to the question is “yes.”
- A floor plan with camera icons may have never been configured. Hence, selecting “pick a camera” results in an error condition: “No floor plan configured for this house.” This error condition becomes a secondary scenario.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Is it possible that the actor will encounter some other behavior at this point?
- Again the answer to the question is “yes.”
- As steps 6 and 7 occur, the system may encounter an alarm condition.
- This would result in the system displaying a special alarm notification (type, location, system action) and providing the actor with a number of options relevant to the nature of the alarm.
- Because this secondary scenario can occur at any time for virtually all interactions, it will not become part of the ACS-DCV use case. Rather, a separate use case—Alarm condition encountered—would be developed and referenced from other use cases as required

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

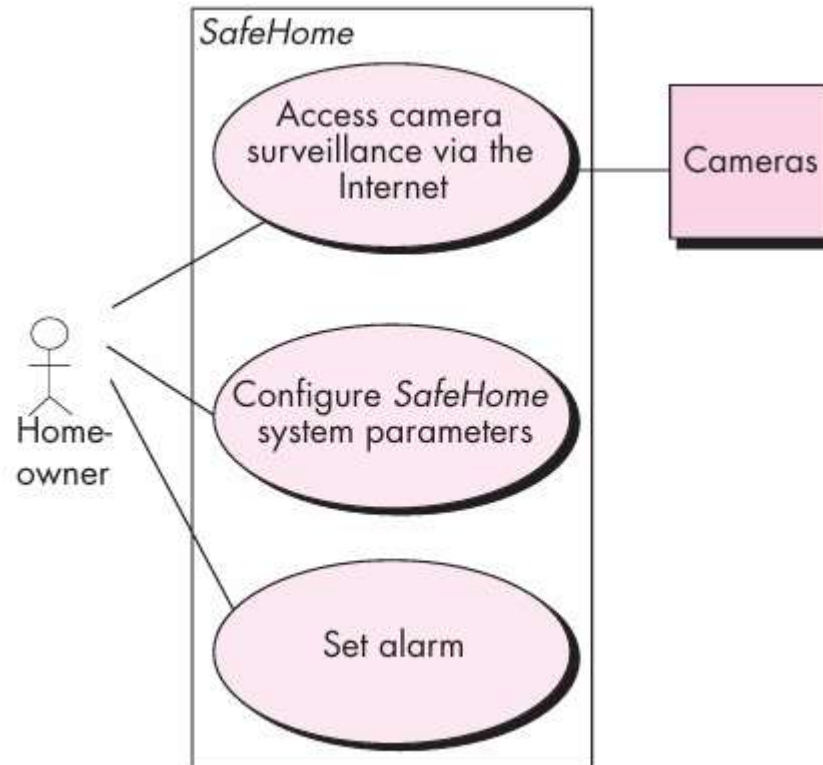
In addition to the three generic questions suggested earlier in this section, the following issues should also be explored:

1. 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.
2. 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.
3. 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

6.2.3 Writing a Formal Use Case

- The informal use cases presented 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.
- The goal in context identifies the overall scope of the use case. The precondition describes what is known to be true before the use case is initiated. The trigger identifies the event or condition that “gets the use case started” .
- The scenario lists the specifications that are required by the actor and the appropriate system responses. Exceptions identify the situations uncovered as the preliminary use case is refined. Additional headings may or may not be included and are reasonably self-explanatory

Preliminary use-case diagram for the SafeHome system





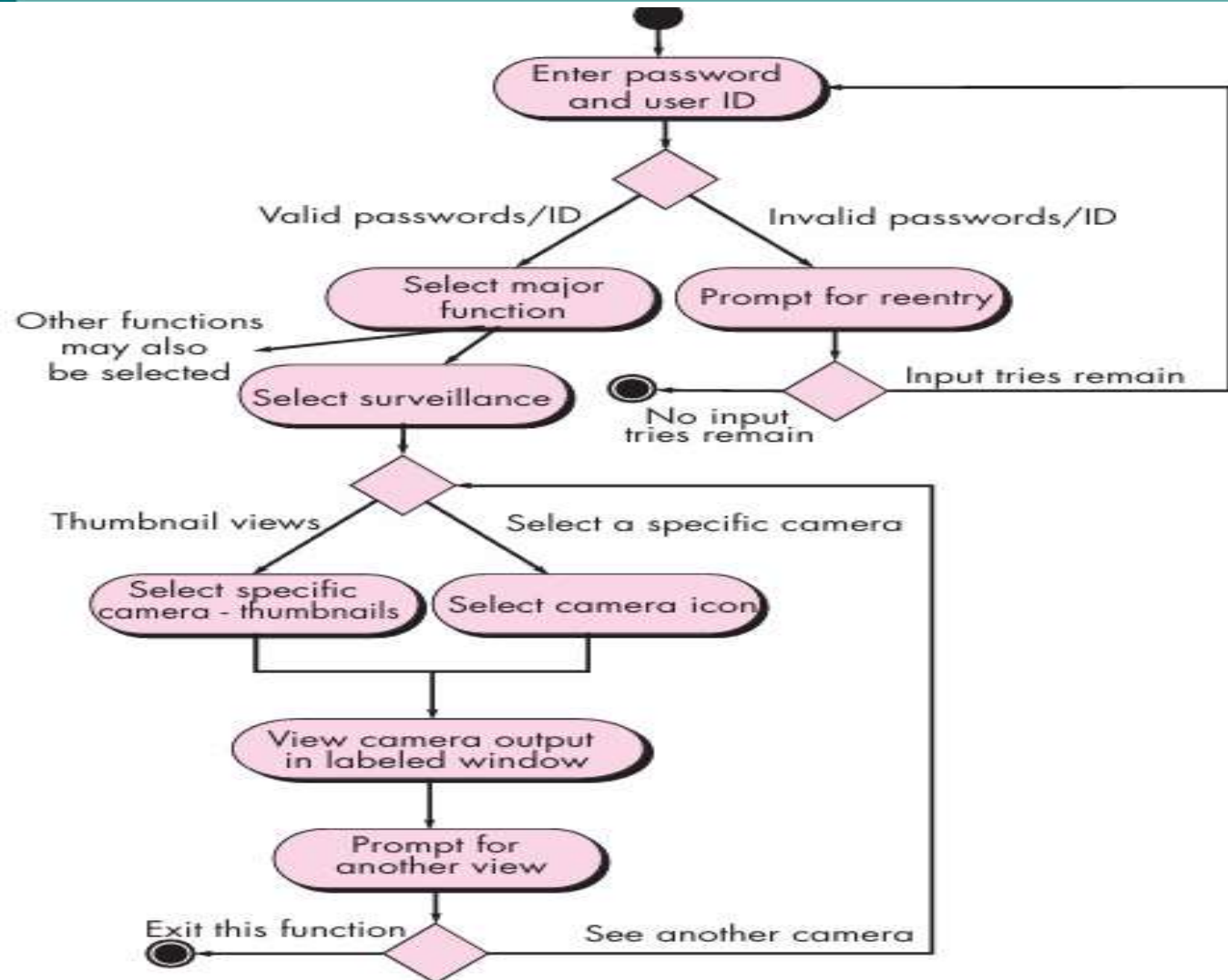
6.3 UML MODELS THAT SUPPLEMENT THE USE CASE

6.3.1 Developing an Activity Diagram

- ❖ The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario.
- ❖ Similar to the flowchart, an **activity diagram** uses **rounded rectangles** to imply a specific system function, **arrows** to represent **flow** through the system, **decision diamonds** to depict a **branching decision** , and **solid horizontal lines** to indicate that **parallel activities** are occurring.
- ❖ The activity diagram adds additional detail not directly mentioned by the use case.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

**Activity
diagram for
Access camera
surveillance via
the Internet—
display camera
views function**

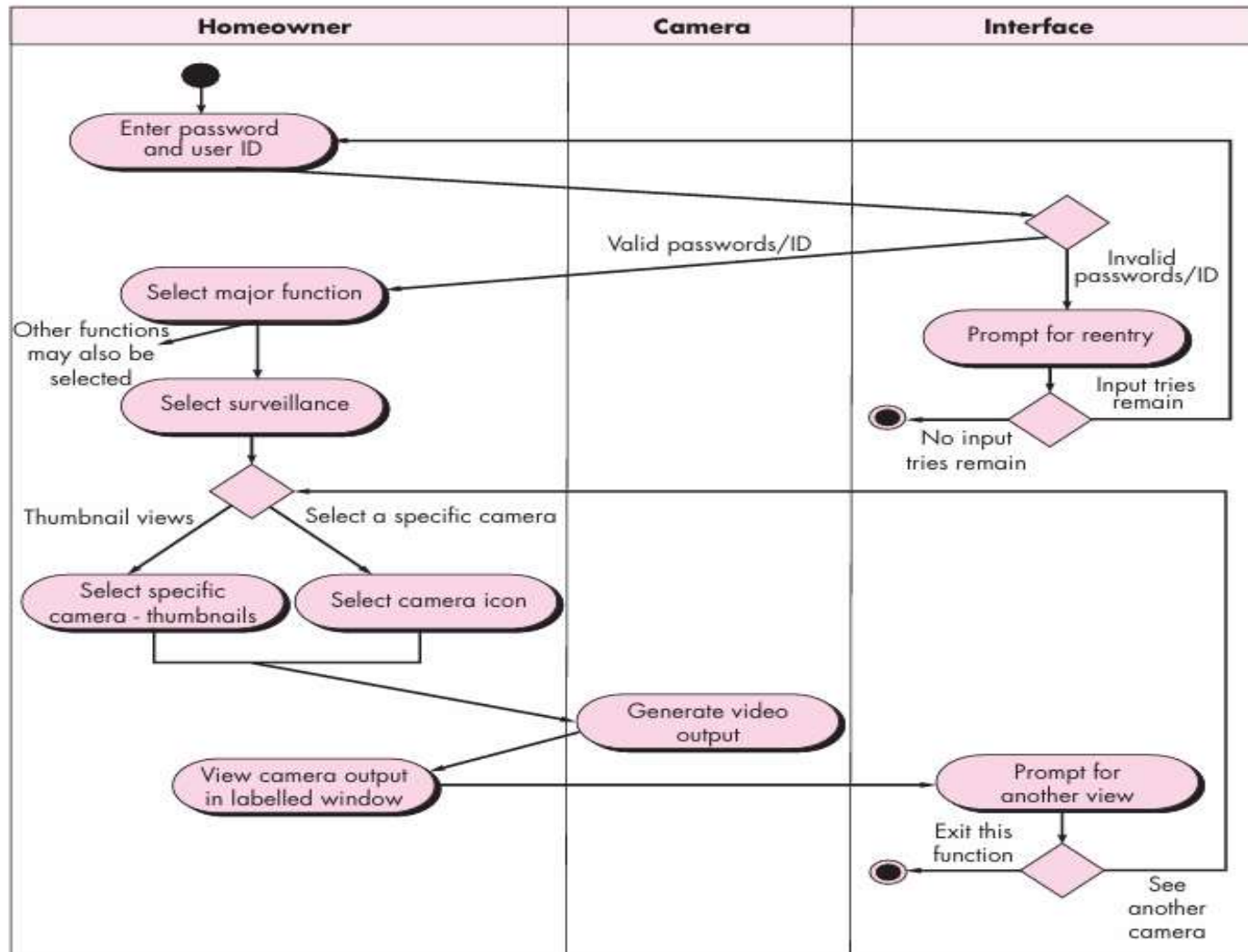


6.3.2 Swimlane Diagrams

- The UML swimlane diagram is a useful variation of the activity diagram and allows you to represent the flow of activities described by the use 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.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Swimlane diagram for Access camera surveillance via the Internet—display camera views function



6.4 DATA MODELING CONCEPTS

- If software requirements include the need to create, extend, or interface with a data base or if complex data structures must be constructed and manipulated, the software team may choose to create a data model as part of overall requirements modeling.
- A software engineer or analyst defines all data objects that are processed within the system, the relationships between the data objects, and other information that is pertinent to the relationships.
- The entity-relationship diagram (ERD) addresses these issues and represents all data objects that are entered, stored, transformed, and produced within an application.

6.4.1 Data Objects

- A data object is a representation of composite information that must be understood by software.
- By composite information, 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.

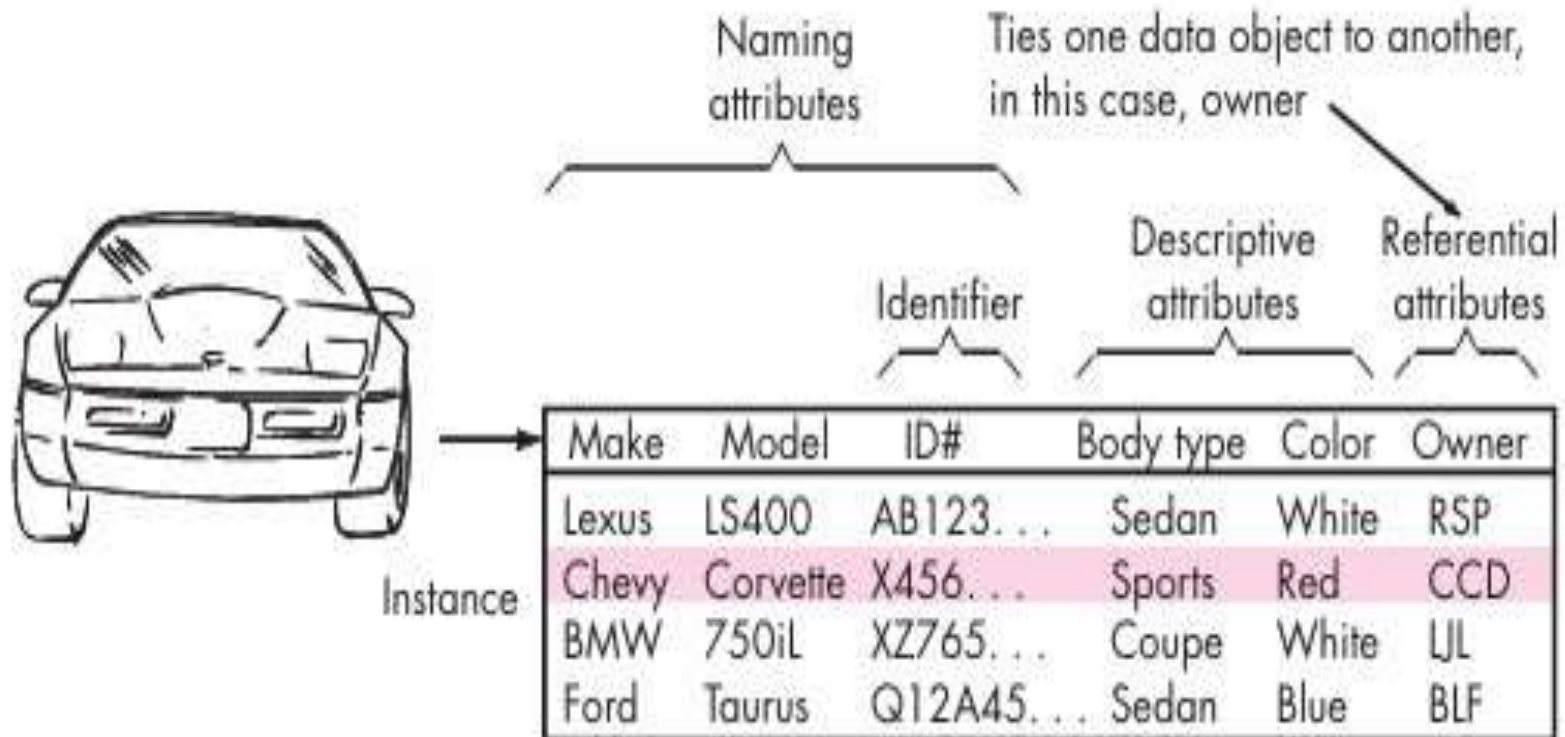
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

A data object can be an

1. External entity (e.G., Anything that produces or consumes information),
2. A thing (e.G., A report or a display),
3. An occurrence (e.G., A telephone call)
4. event (e.G., An alarm),
5. A role (e.G., Salesperson),
6. An organizational unit (e.G., Accounting department),
7. A place (e.G., A warehouse),
8. 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. The description of the data object incorporates the data object and all of its attributes. A data object encapsulates data only—there is no reference within a data object to operations that act on the

Tabular representation of data objects



6.4.3 Relationships

- ❑ Data objects are connected to one another in different ways.
- ❑ Consider the two data objects, person and car. These objects can be represented using the simple notation

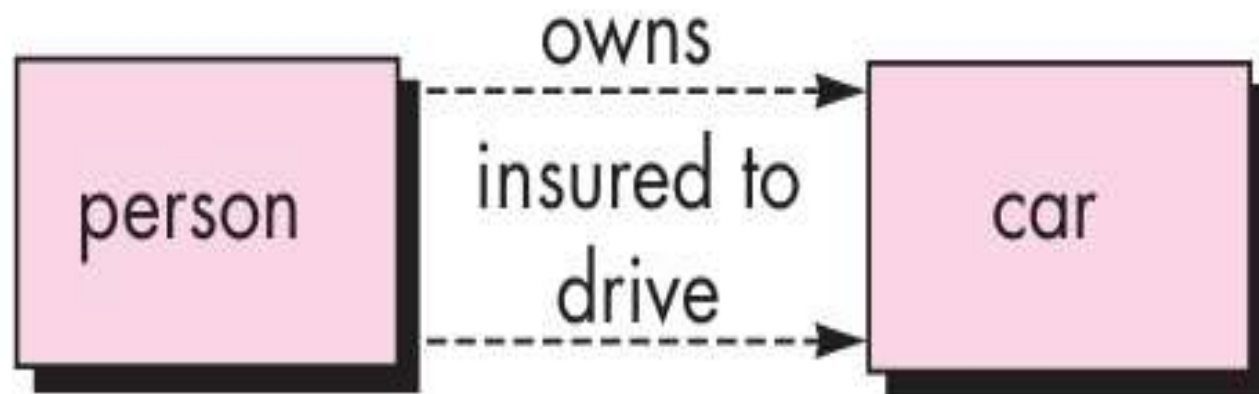


(a) A basic connection between data objects

- ❑ A connection is established between person and car because the two objects are related.
- ❑ You can establish 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.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Figure provide important information about the directionality of the relationship and often reduce ambiguity or misinterpretations



(b) Relationships between data objects

6.5 CLASS-BASED MODELING

- Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.
- The elements of a class-based model include classes and objects, attributes, operations, class responsibility-collaborator (CRC) models, collaboration diagrams, and packages.

Informal guidelines that will assist in their identification and representation

- 1 Identifying Analysis Classes
- 2 Specifying Attributes
- 3 Defining Operations
- 4 Class-Responsibility-Collaborator (CRC) Modeling
- 5 Associations and Dependencies
- 6 Analysis Packages

6.5.1 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.
- 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.

Analysis classes manifest themselves in one of the following ways:

1. **External entities** (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
2. **Things** (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
3. **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.
4. **Roles** (e.g., manager, engineer, salesperson) played by people who interact with the system.
5. **Organizational units** (e.g., division, group, team) that are relevant to an application.
6. **Places** (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
7. **Structures** (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects.

It is also important to note what classes or objects are not. In general, a class should never have an “imperative procedural name”.

- Example

InvertImageor even ImageInversion

- To illustrate how analysis classes might be defined during the early stages of modeling, consider a grammatical parse (nouns are underlined, verbs italicized) for a processing narrative for the SafeHome security function.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- 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**.
- During **installation**, the SafeHome PC is used to program and configure the **system**. Each sensor is assigned a **number and type**, a master password is programmed for arming and disarming the system, and **telephone number(s)** are input for dialing when **a sensor event** occurs.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- When a sensor event is recognized, the software invokes an **audible alarm** attached to the system. After a **delay time** that is specified by the homeowner during system configuration activities, the software dials a telephone number of a **monitoring service**, provides **information** about the **location**, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until **telephone connection** is obtained.
- 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 . . .

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Extracting the nouns, we can propose a number of potential classes

Potential Class

homeowner

sensor

control panel

installation

system (alias security system)

number, type

master password

telephone number

sensor event

audible alarm

monitoring service

General Classification

role or external entity

external entity

external entity

occurrence

thing

not objects, attributes of sensor

thing

thing

occurrence

external entity

organizational unit or external entity

Coad and Yourdon suggest **six** selection characteristics that should be used as you consider each potential class for inclusion in the analysis model

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 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.

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 requirements model

6.5.2 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.

For example

- If we were to build a system that tracks baseball statistics for professional baseball players, the attributes of the class Player would be quite different than the attributes of the same class when it is used in the context of the professional baseball pension system.
- In the former, attributes such as name, position, batting average, fielding percentage, years played, and games played might be relevant.



- For the latter, some of these attributes would be meaningful, but others would be replaced by attributes like average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.
- To develop a meaningful set of attributes for an analysis class, you should study each use case and select those “things” that reasonably “belong” to the class.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

The following question should be answered for each class:

- To illustrate, we consider the System class defined for SafeHome.
- A homeowner can configure the security function to reflect sensor information, alarm response information, activation/deactivation information, identification information, and so forth.
- We can represent these composite data items in the following manner:

identification information = system ID + verification phone number + system status

alarm response information = delay time + telephone number

activation/deactivation information = master password + number of allowable tries +
temporary password

6.5.3 Defining Operations

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
- (4) operations that monitor an object for the occurrence of a controlling event.

6.5.4 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 requirements.
- Ambler 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.*

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Responsibilities are the attributes and operations that are relevant for the class.
- Stated simply, a responsibility is “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, a collaboration implies either a request for information or a request for some action..

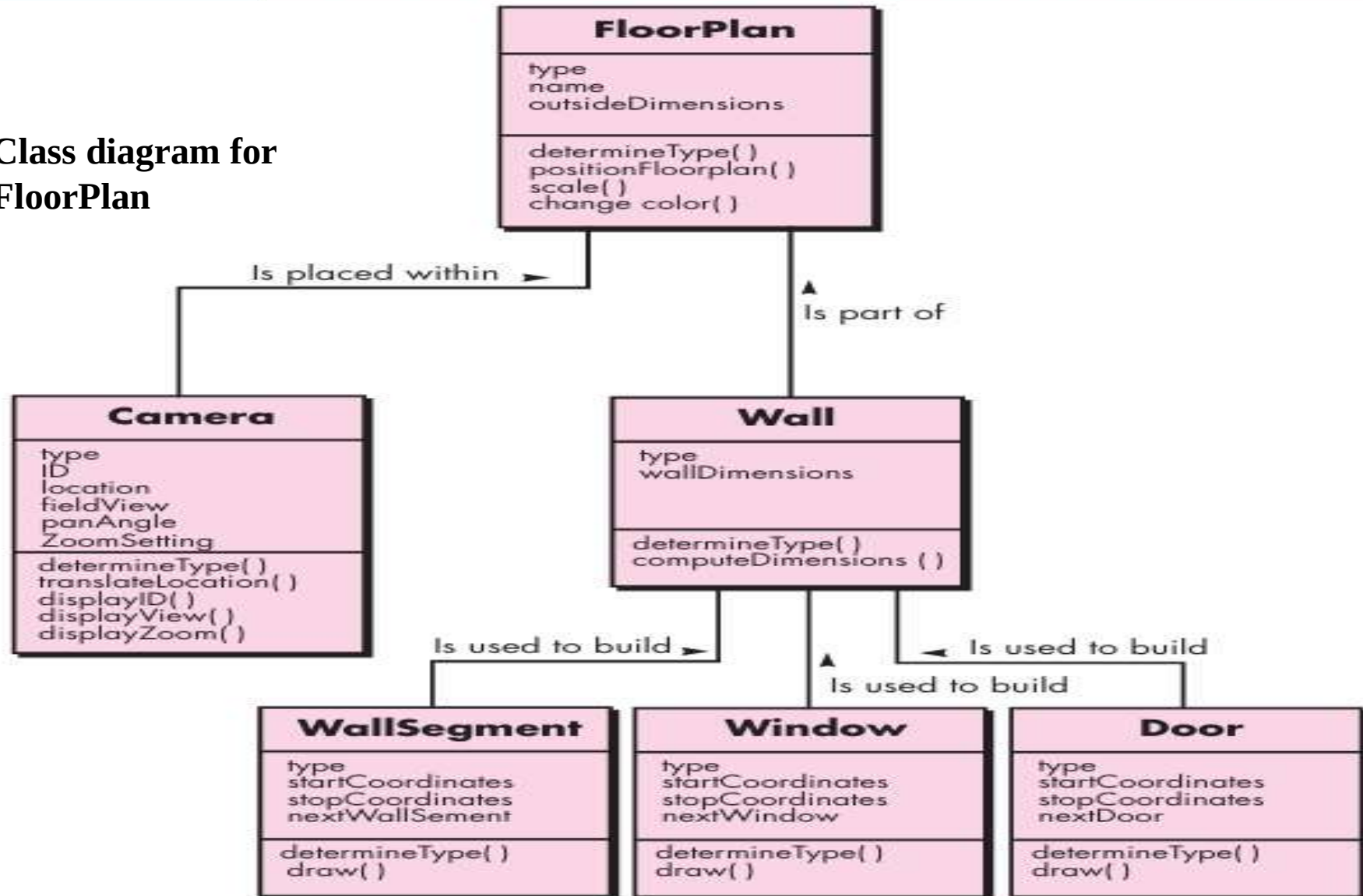


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- A simple CRC index card for the FloorPlanclass is illustrated in Figure
- The list of responsibilities shown on the CRC card is preliminary and subject to additions or modification. The classes Wall and Camera are noted next to the responsibility that will require their collaboration

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Class diagram for FloorPlan



Classes

- Entity classes, also called model or business classes, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
- These presented can be extended by considering the following categories

1.Entity classes also called model or business classes, are extracted directly from the statement of the problem (e.g., Floor Plan and Sensor). These classes typically represent things that are to be stored in a database and persist throughout the duration of the application.

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

2. 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.
- Entity objects contain information that is important to users, but they do not display themselves. Boundary classes are designed with the responsibility of managing the way entity objects are represented to users.
- For example, a boundary class called Camera Window would have the responsibility of displaying surveillance camera output for the SafeHome system.

3.Controller classes

- Manage a “unit of work” from start to finish.
- That is, controller classes can be designed to manage
 - (1) the creation or update of entity objects,
 - (2) the instantiation of boundary objects as they obtain information from entity objects,
 - (3) complex communication between sets of objects,
 - (4) validation of data communicated between objects or between the user and the application. In general, controller classes are not considered until the design activity has begun

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,
 - (2) a class can collaborate with other classes.
- Wirfs-Brock and her colleagues [Wir90] define collaborations in the following way:



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- Collaborations represent requests from a client to a server in fulfillment of a client responsibility. A collaboration is the embodiment of the contract between the client and the server....We say that an object collaborates with another object if, to fulfill a responsibility, it needs to send the other object any messages. A single collaboration flows in one direction—representing a request from the client to the server. From the client's point of view, each of its collaborations is associated with a particular responsibility implemented by the server.
- Collaborations are identified by determining whether a class can fulfill each responsibility itself. If it cannot, then it needs to interact with another class. Hence, a collaboration.

6.5.5 Associations and Dependencies

❑ In many instances, two analysis classes are related to one another in some fashion, much like two data objects may be related to one another. In UML these relationships are called associations.

❑ The FloorPlan class is defined by identifying a set of associations between FloorPlan and two other classes Camera and Wall.

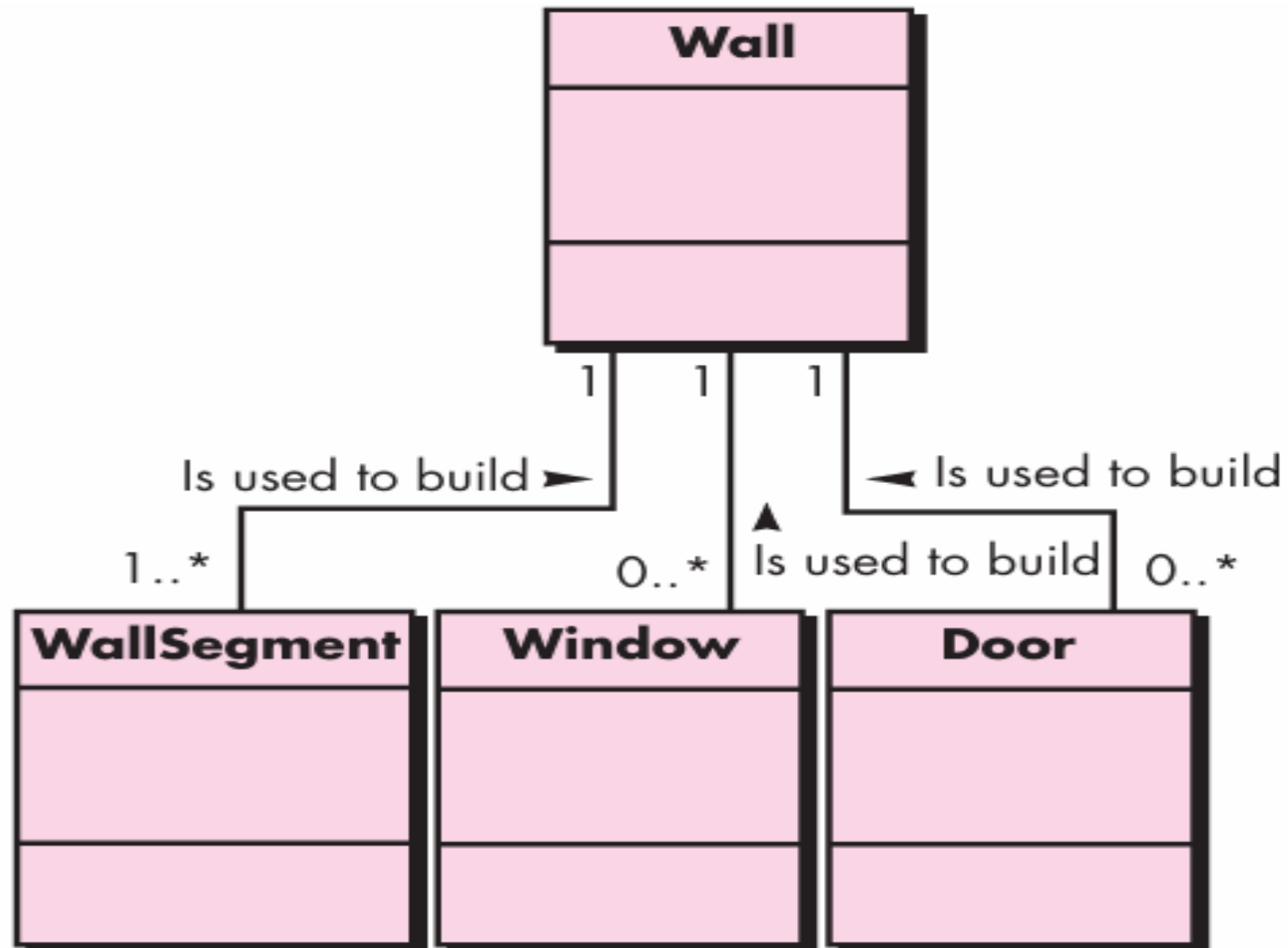
❑ The class Wall is associated with three classes that allow a wall to be constructed, WallSegment, Window, and Door.

Multiplicity

- In some cases, an association may be further defined by indicating **multiplicity**.
- Referring to Figure, a Wall object is constructed from one or more WallSegment objects.
- In addition, the Wallobject may contain 0 or more Window objects and 0 or more Door objects.
- multiplicity constraints are illustrated in Figure where “one or more” is represented using 1..*, and “0 or more” by 0..*. In UML, the asterisk indicates an unlimited upper bound on the range

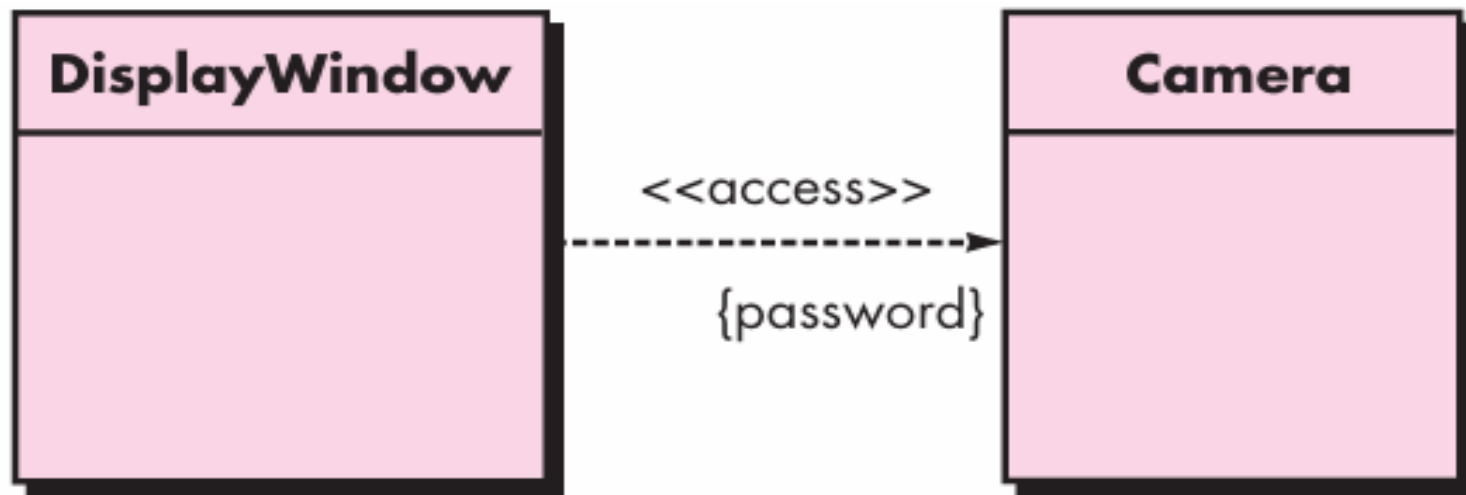
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Multiplicity



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

As an illustration of a simple dependency within the SafeHome surveillance system, a Camera object (the server class) provides a video image to a DisplayWindow object (the client class). The relationship between these two objects is not a simple association, yet a dependency association does exist.



Dependencies

- 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.
- Dependencies are defined by a stereotype.
- A stereotype is an “extensibility mechanism” within UML that allows you to define a special modeling element whose semantics are custom defined.
- In UML stereotypes are represented in double angle brackets (e.g., <>).

6.5.6 Analysis Packages

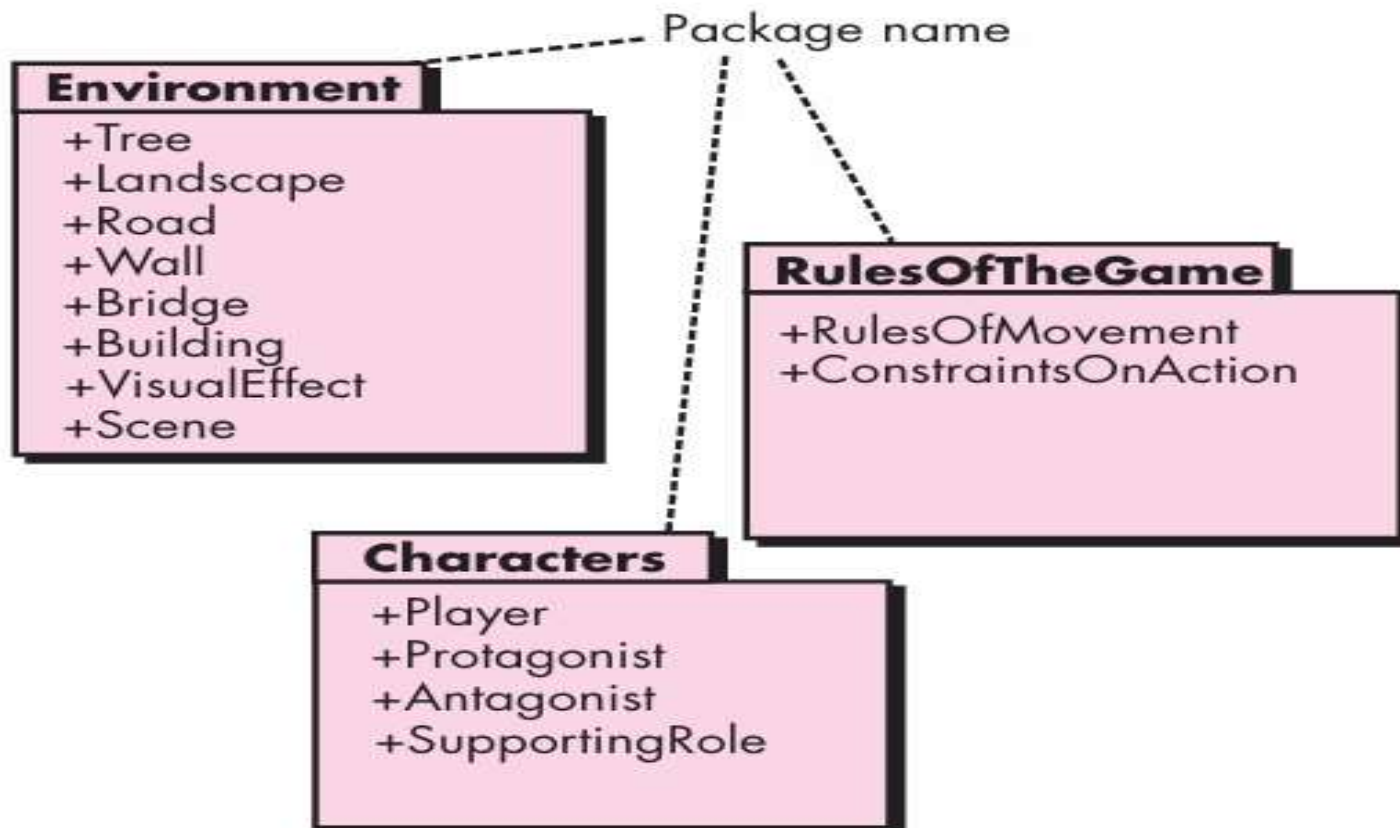
- That is, various elements of the analysis model (e.g., use cases, analysis classes) are categorized in a manner that packages them as a grouping—called an analysis package—that is given a representative name.

Ex: video game

- As the analysis model for the video game is developed, a large number of classes are derived.
- Some focus on the **game environment** —*the visual scenes* that the user sees as the game is played.
- **Classes** such as *Tree, Landscape, Road, Wall, Bridge, Building, and Visual Effect* might fall within this category.
- Others focus on the **characters** within the game, describing their *physical features, actions, and constraints*.
- Classes such as **Player**, *Protagonist, Antagonist, and Supporting Roles* might be defined.
- Others describe the **rules of the game** —how a *player navigates through the environment*.
- **Classes** such as *RulesOfMovement and ConstraintsOnAction* are candidates here. Many other categories might exist.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

These classes can be grouped in analysis packages as shown in Figure



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- 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.
- A minus sign indicates that an element is hidden from other packages
- A # symbol indicates that an element is accessible only to packages contained within a given package.

Chapter 3

- Requirement Modeling Strategies : Flow oriented Modeling , Behavioral Modeling

7.1 REQUIREMENTS MODELING STRATEGIES

Overview

- Requirements modeling involves creating representations of software requirements that help in understanding, analyzing, and designing the system effectively.



Structured Analysis Approach

- **Focus:** Treats **data** and **processes** as **separate entities**.
- **Data Modeling:**
 - Defines **data objects**.
 - Identifies their **attributes** and **relationships**.
- **Process Modeling:**
 - Shows **how data is transformed** as it flows through the system.
 - Models **processes that manipulate data objects**.

Combined Approach

- The book proposes an **analysis model** that **integrates features** from both **structured analysis** and **OOA**.
- However, in practice, **software teams** often:
 - **Choose one approach** exclusively.
 - **Exclude representations** from the other.

Key Consideration

- The focus should **not** be on selecting the “**best**” approach.
- Instead, determine:
 - **Which combination** of modeling techniques
 - Provides **stakeholders** with the **clearest understanding** of the software requirements.
 - Acts as the **most effective bridge** to **software design**.

7.2 FLOW-ORIENTED MODELING

Overview

- **Flow-oriented modeling** focuses on how **data moves through a system**.
- Though seen as **outdated by some**, it remains **widely used** in requirements analysis.

Data Flow Diagrams (DFDs)

- Core tool used in **flow-oriented modeling**.
- Not a formal part of **UML**, but often used **alongside UML** to enhance understanding of:
 - **System requirements**
 - **Data flow**

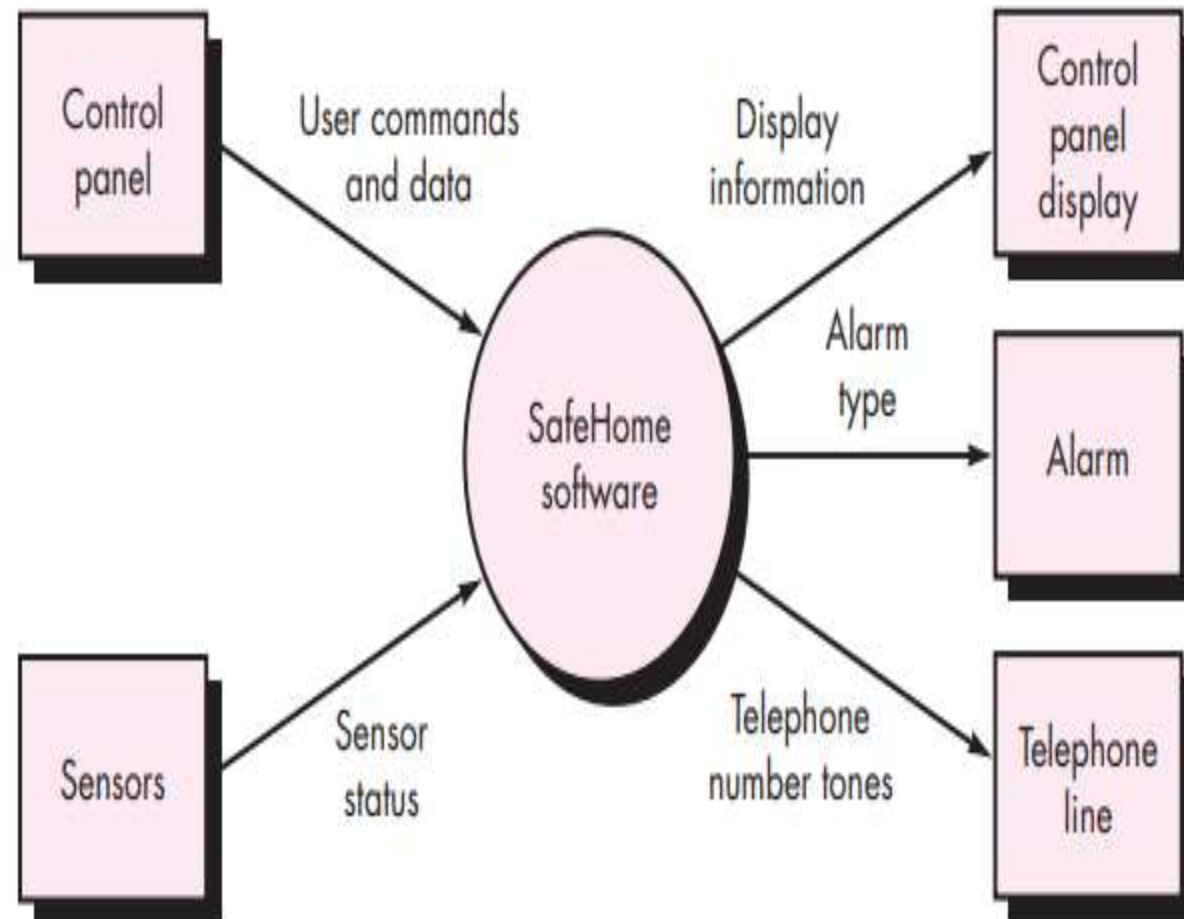
DFD – Input-Process-Output View

- A system is viewed as:
 - **Input:** Data objects flow **into** the software.
 - **Process:** Data is **transformed** by processing elements.
 - **Output:** Transformed data flows **out** of the software.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

FIGURE 7.1

Context-level
DFD for the
SafeHome
security
function





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

◆ DFD Elements

Element	Symbol	Description
Data Object	Labeled arrows	Represents data moving between processes
Processing Element	Circles or bubbles	Represents transformation of data
Data Store	Open-ended rectangles	Stores data for later use (not mentioned above, but commonly included)

Hierarchical Structure

- DFDs are **presented hierarchically**:
 - **Level 0 DFD** (also called **Context Diagram**):
 - Shows the **entire system as a single process**.
 - Identifies external entities and major data flows.
 - **Lower-level DFDs**:
 - Provide **more detail** by decomposing processes.
 - Each level adds **refined functionality and data flow**.

Key Point

- **Flow-oriented modeling** helps in understanding:
 - **How data is transformed**
 - **Where it flows within the system**
- **Aids in visualizing system behavior** before actual design begins.

7.2.1 Creating a Data Flow Model – Point-wise Notes

- **Purpose of a Data Flow Diagram (DFD)**
- **Models the information and functional domains** of a system.
- **Performs functional decomposition** as DFDs are refined into deeper levels.
- Helps in **visualizing how data moves** through the system and is processed.

Key Guidelines for DFD Creation

1. **Level 0 DFD** (Context Diagram) shows the entire system as a **single bubble**.
2. Clearly **identify primary inputs and outputs**.
3. Begin refinement by isolating:
 1. **Candidate processes**
 2. **Data objects**
 3. **Data stores**
4. Label **all arrows and bubbles meaningfully**.
5. Ensure **information flow continuity** between DFD levels.
6. Refine **one bubble at a time** to avoid over-complication.
7. Do **not show procedural detail**—focus on **data and information flow** only.

Role of Grammatical Parse

- Analyze the **use case narrative** to extract:
 - **Nouns**: Become data objects, external entities, or data stores.
 - **Verbs**: Represent system **processes** (bubbles).
- Use parsing to guide **refinement to next DFD level**.
- Example: In SafeHome,
 - “Sensor” (noun) → data object.
 - “Monitor” (verb) → process.

SafeHome Security System – DFD Overview

- **Level 0 DFD (not shown here, but implied):**
- Represents **SafeHome** system as **one single process**.
- External entities: **User / Homeowner, Control Panel, Sensors, Telephone line**, etc.
- Major data flows: **Commands, Status updates, Sensor input, Alarm messages**.

Level 1 DFD for SafeHome Security Function

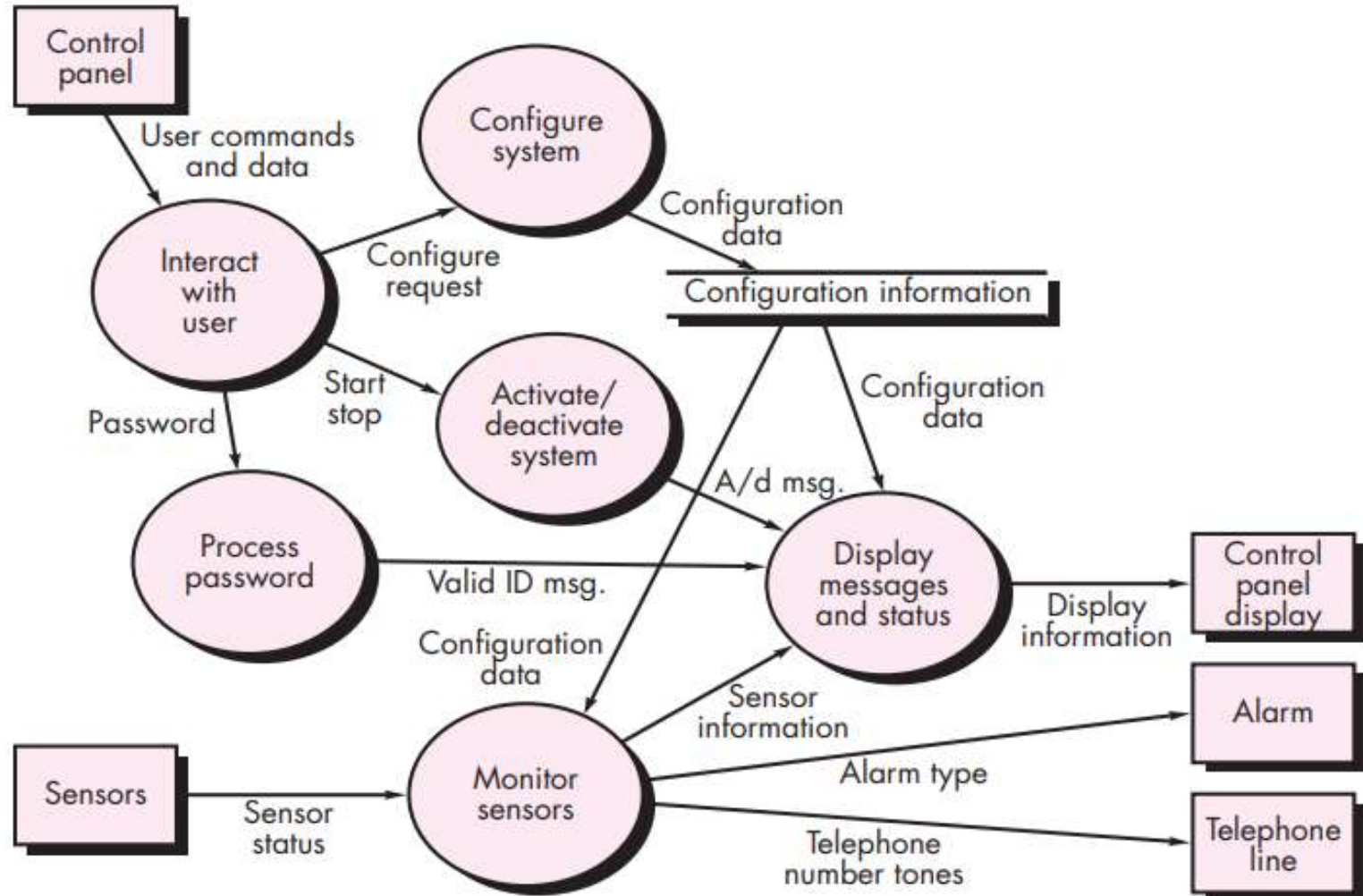
Main Processes:

- **Interact with user** – Receives user commands and sends configuration/password.
- **Process password** – Validates password.
- **Configure system** – Accepts config requests and generates configuration data.
- **Activate/deactivate system** – Starts/stops the system.
- **Monitor sensors** – Checks sensor status and triggers alarms.
- **Display messages and status** – Shows messages to user/display panel.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

FIGURE 7.2

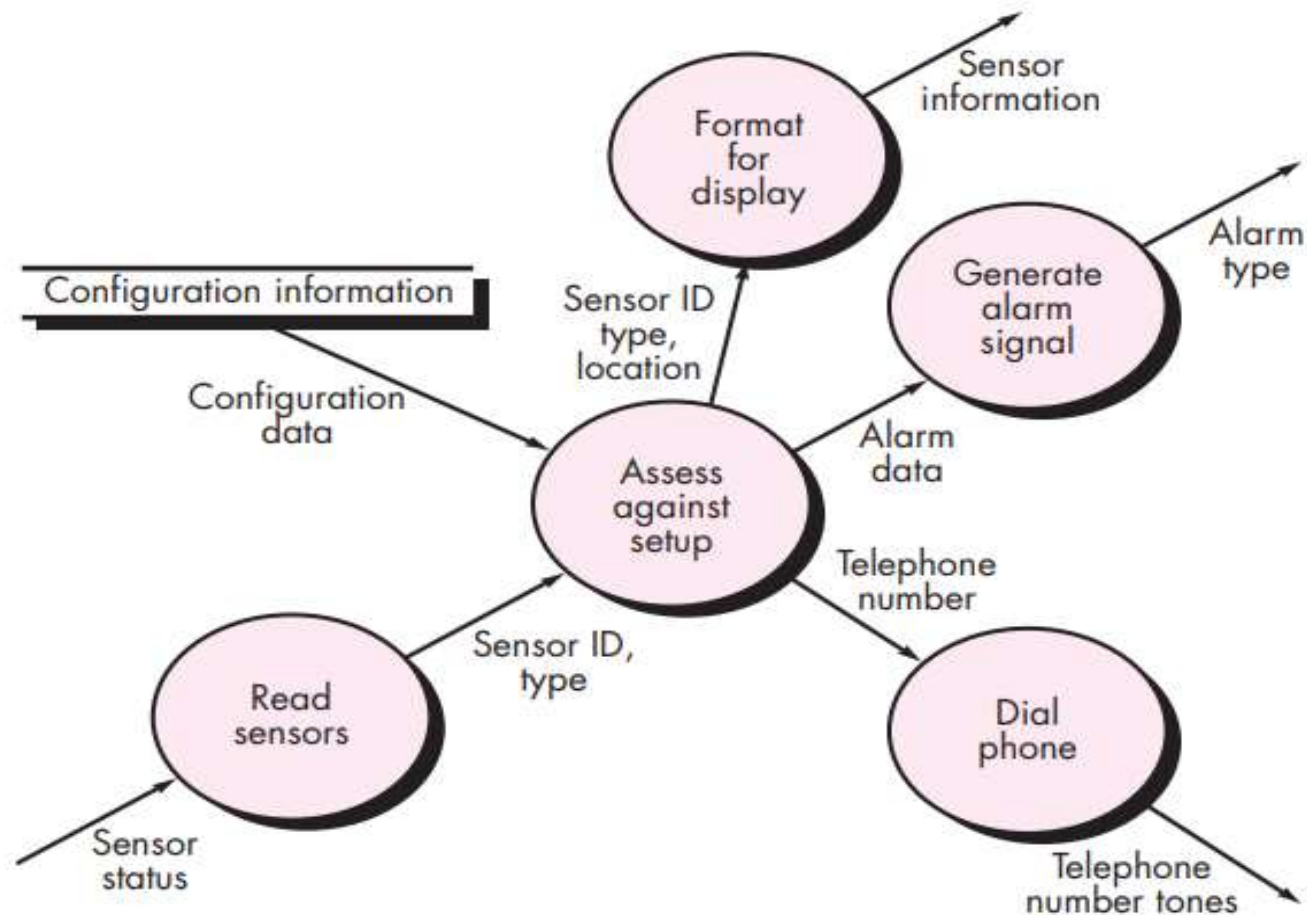
Level 1 DFD for
SafeHome
security
function



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

FIGURE 7.3

Level 2 DFD
that refines
the *monitor
sensors* process



Data Flows:

- Password → Process password → Valid ID message
- Sensor status → Monitor sensors → Alarm type, Telephone number tones
- Configuration data flows among several processes
- Outputs go to Alarm, Telephone line, Control panel display

External Entities:

- **Control panel**
- **Sensors**
- **Alarm**
- **Telephone line**
- **Control panel display**

Figure 7.3 – Level 2 DFD (Refinement of "Monitor Sensors")

Refined Processes:

- **Read sensors** – Receives sensor status.
- **Assess against setup** – Compares sensor data with config info.
- **Format for display** – Sends displayable sensor info.
- **Generate alarm signal** – Sends appropriate alarm type.
- **Dial phone** – Sends telephone number tones.

Data Flows:

- Sensor status → Read sensors
- Sensor ID/type → Assess against setup
- Alarm data → Generate alarm signal
- Telephone number → Dial phone
- Sensor ID/type/location → Format for display
- **Data Stores / External Inputs:**
- **Configuration information**

Refinement Strategy

- Continue refining each bubble until it performs a **single function**.
- Aim for "**single-minded**" processes (focused and cohesive).
- Stop refinement when processes can be **directly implemented as program components**.

7.2.2 Creating a Control Flow Model

When is Control Flow Modeling Needed?

- Some applications are **data-driven**, and **data models + DFDs** are sufficient.
- However, many applications are **event-driven** or **time-sensitive**.
- Such systems require **control flow modeling** in addition to data flow modeling.

Characteristics of Event-Driven Applications

- **Triggered by events** rather than just data input/output.
- **Produce control information**, not just reports or displays.
- **Highly concerned with time and performance constraints.**
- Often used in **real-time or embedded systems** (e.g., alarms, automation).

Control Items (Events)

- Represented as:
 - **Boolean values:** true/false, on/off, 1/0
 - **Discrete states:** empty, jammed, full, etc.

Guidelines to Identify Candidate Events / Control Items

- **List all sensors** monitored/read by the software.
- **List all interrupt conditions** (e.g., hardware/software interrupts).
- **List all switches** actuated by the user or operator.
- **List all data conditions** that may trigger control changes.
- Use **noun/verb parsing** from processing narratives to extract control items.
 - Identify:
 - **System states**
 - **Transitions** between states
 - **Conditions** that trigger state changes
- Always check for **possible omissions**:
- Ask: “Can the system reach or exit this state in **any other way**?”

Example Control Events in SafeHome

- **Sensor event:** Triggered when a sensor is tripped.
- **Blink flag:** Used to blink the display as an alert.
- **Start/stop switch:** Used to turn the system on or off.

7.2.3 The Control Specification (CSPEC)

Purpose of CSPEC

- The **Control Specification (CSPEC)** defines how the system behaves in response to **events**.
- It represents system behavior in two forms:
 - **State Diagram** – *Sequential* behavior representation.
 - **Process Activation Table (PAT)** – *Combinatorial* behavior representation.

1. State Diagram

- A graphical representation showing:
 - **System states**
 - **Events** triggering **transitions** between those states
 - **Actions** executed during transitions or while in a state

Figure 7.4: State Diagram for SafeHome

- **System States:**
- **Resetting**
 - Initializes system on power-on.
 - Displays "Starting system", "Please wait", runs diagnostics.
- **Idle**
 - Waiting state; system not monitoring.
 - Ready to activate or power off.
- **MonitoringSystemStatus**
 - Armed mode; actively monitoring sensors.
 - Performs monitor and control operations.
- **ActingOnAlarm**
 - Triggered when a sensor detects an event.
 - Sounds alarm, notifies responders

Important Transitions:

- Power "on" → Resetting
- systemOK → Idle
- Activate → MonitoringSystemStatus
- sensorTriggered → ActingOnAlarm
- deactivatePassword → Returns to Idle from either MonitoringSystemStatus or ActingOnAlarm
- falseAlarm, timeOut → Return from ActingOnAlarm to MonitoringSystemStatus
- failureDetected → Message displayed: "contact Vendor"
- off/powerOff → Ends operation from Idle

2. Process Activation Table (PAT)

- **Alternative behavioral model** showing:
 - Which **processes (bubbles)** are activated
 - Under what **event conditions**
- Useful for **designers** implementing control logic/executives.
- Complements the **state diagram** by offering a **process-centric view**.

Benefits of Using CSPEC

- Helps identify **missing behavior** (e.g., missed transitions or states).
- Ensures **completeness** in specifying event-driven logic.
- Serves as a **bridge** between analysis and design:
 - **State diagram** → sequential control logic.
 - **PAT** → what processes get triggered when.

Limitations

- CSPEC only defines **external behavior**.
- It does **not specify internal workings** of each process.
 - Internal logic is modeled using techniques discussed in **Section 7.2.4** (like Process Specification/PSPEC).

7.3 CREATING A BEHAVIORAL MODEL

To create the model, you should perform the following steps:

1. Evaluate all use cases to fully understand the sequence of interaction within the system.
2. Identify events that drive the interaction sequence and understand how these events relate to specific objects.
3. Create a sequence for each use case.
4. Build a state diagram for the system.
5. Review the behavioral model to verify accuracy and consistency.

7.3.1 Identifying Events with the Use Case

Use Cases and Events:

- A use case represents a sequence of interactions between the system and external actors.
- An **event** occurs whenever information is exchanged between the system and an actor.

Definition of an Event:

- An event is **not** the information exchanged, but the **fact** that the information was exchanged.

Analyzing Use Cases for Events:

- Use cases are examined to identify **points of information exchange**.
- Example used: SafeHome security system use case involving a homeowner entering a password.

Illustrative Example - Password Entry:

- Event: *“Homeowner uses the keypad to key in a four-digit password.”*
- Actor: **Homeowner**
- Receiver: **ControlPanel**
- Event name: **password entered**
- Data transferred: Four-digit password (not essential to behavioral model)

Event Impact on Control Flow:

- Some events affect the flow of control in the use case; others do not.
- Example:
 - **password entered**: No direct impact on control flow.
 - **password compared**: Has a direct effect on control and information flow.

Event Allocation to Objects:

- Events are assigned to the objects involved:
 - **Homeowner**: Generates the **password entered** event.
 - **ControlPanel**: Recognizes and processes events like **password compared**.

Behavioral Modeling Insight:

- The behavioral model focuses on the **flow of events** and how they impact system behavior, rather than the data exchanged.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

Two Types of State Representations in Behavioral Modeling:

- (1) **State of each class** during system operation (internal view).
- (2) **State of the entire system** as observed externally during operation (external view).

Passive vs Active State of a Class:

- **Passive State:**
 - Reflects the **current values of all attributes** of an object.
 - Example: In the class Player, passive state includes position, orientation, and other attributes like "magic wishes remaining".
- **Active State:**
 - Describes the **dynamic status or behavior** of an object during processing or transformation.
 - Example active states of Player: *moving, at rest, injured, being cured, trapped, lost*, etc.

State Transitions:

- A **trigger or event** is needed to **change** from **one active state to another**.

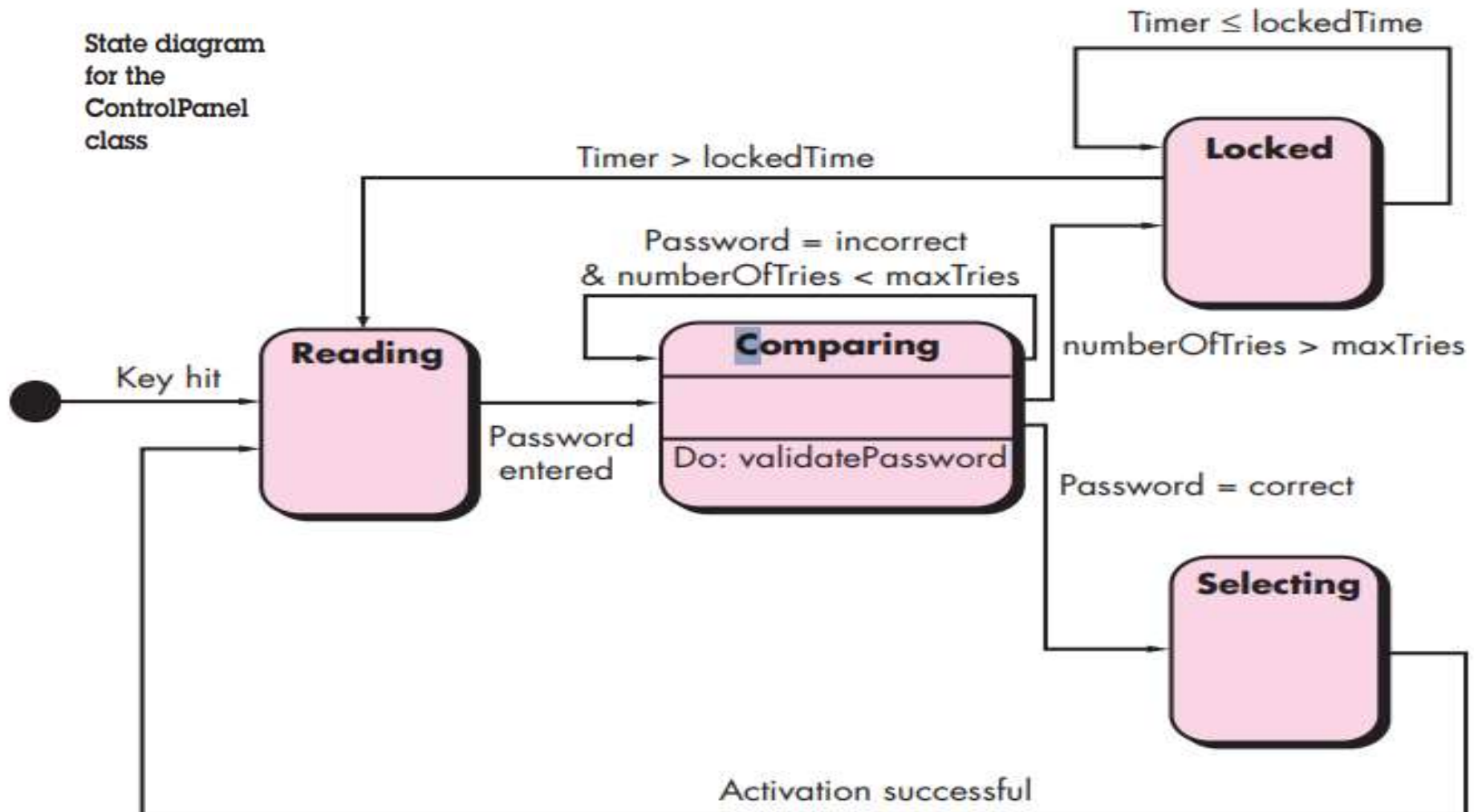
Two Types of Behavioral Representations:

- **Class-level Behavior:**
 - Shows how an **individual class** **changes state** in response to external events.
- **System-level Behavior:**
 - Depicts how the **entire software** **behaves over time**.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

State diagrams for analysis classes.

State diagram
for the
ControlPanel
class



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING - AI & ML

- The diagram is a **UML state diagram** used for **behavioral modeling** of the ControlPanel class in the **SafeHome security system**.
- It represents **active states** of the object and the **events (triggers)** that cause transitions between those states.

Key Elements in the State Diagram

- **States Identified:**
 - **Reading:** Waiting for password input.
 - **Comparing:** Validates the password using validatePassword() operation.
 - **Locked:** Control panel is locked due to multiple incorrect attempts.
 - **Selecting:** Access granted; waiting for further actions.

- **Initial State:**

- Triggered by "Key hit", leading to the **Reading** state.

- **Transitions and Events:**

- **Reading → Comparing:** Triggered by event "Password entered".
- **Comparing → Reading:** If password is incorrect **AND** $\text{numberOfTries} < \text{maxTries}$.
- **Comparing → Locked:** If $\text{numberOfTries} > \text{maxTries}$.
- **Comparing → Selecting:** If password is correct → "Activation successful".
- **Locked → Reading:** If $\text{Timer} > \text{lockedTime}$.
- **Reading → Locked:** If $\text{Timer} \leq \text{lockedTime}$.

Guards and Actions

- **Guard Conditions:**

- A **guard** is a **Boolean condition** that must be true for a transition to occur.
- Example Guard: if (password input = 4 digits) → Then transition to Comparing state.

- **Actions:**

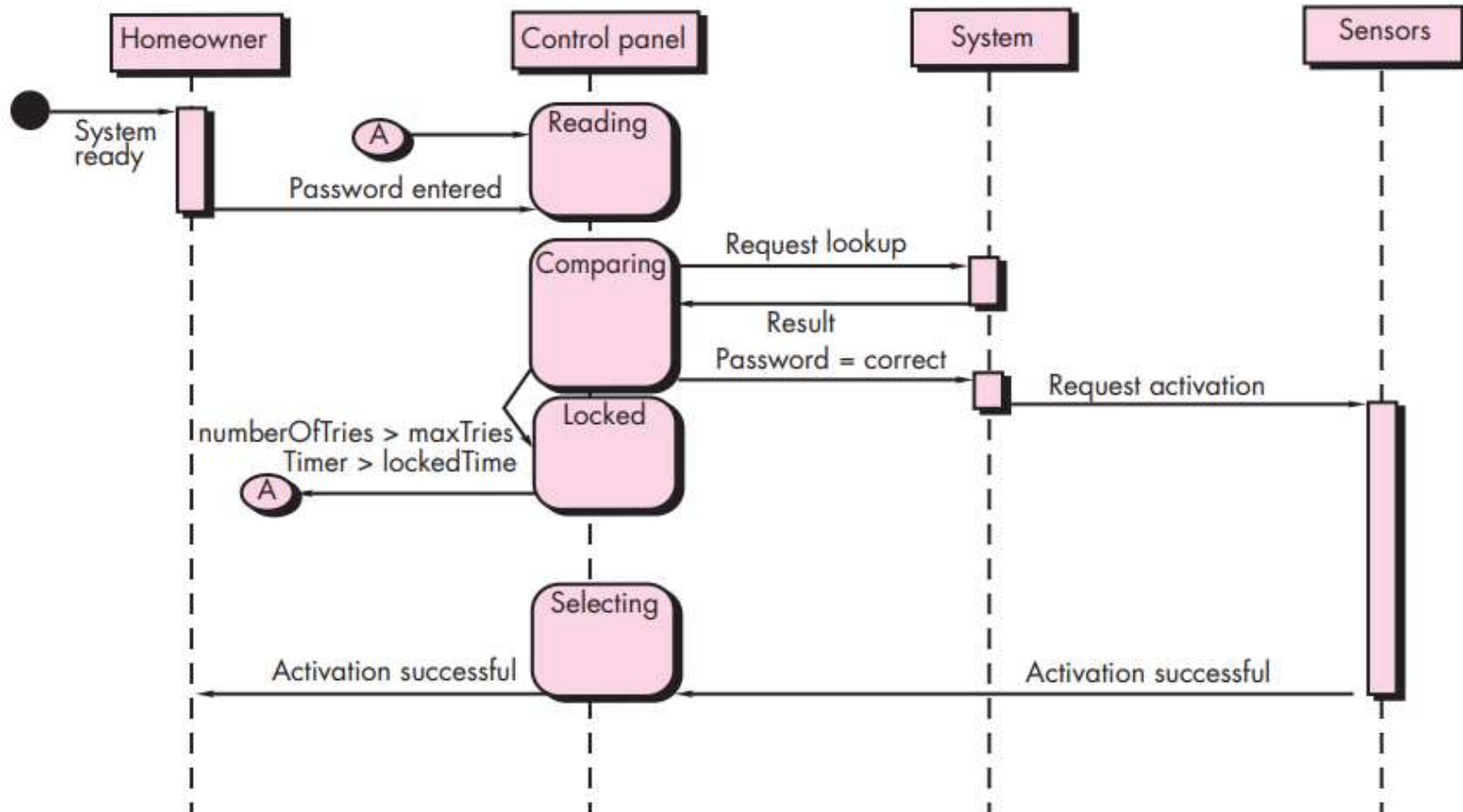
- An **action** occurs with a state transition or as its consequence.
- Example: In **Comparing** state, the action is `validatePassword()` which compares entered password with the stored one.

Insights from the Model

- The diagram provides:
 - A **"life history"** of the ControlPanel object.
 - Understanding of how the object **responds to external events**.
 - Clarity on both **internal states** and their **transitions based on conditions** and **user interaction**.

Sequence diagrams

Sequence diagram (partial) for the SafeHome security function



Purpose of Sequence Diagram

- **Definition:**
 - A sequence diagram in UML shows how **events cause transitions between objects over time.**
- **Relation to Use Case:**
 - It is a **shorthand version** of a use case.
 - Represents the **flow of behavior across key classes** as events occur.
- **Time Axis:**
 - Time flows **vertically downward.**
 - **Narrow vertical rectangles** indicate **time spent processing** an activity.

Key Components in Figure 7.7

- **Objects Involved:**

- Homeowner
- Control Panel
- System
- Sensors

- **Initial Trigger:**

- The external event "**System ready**" activates the Homeowner.

Sequence of Events:

- **Homeowner enters password** → Event passed to ControlPanel (Reading state).
- ControlPanel enters **Comparing state** → Sends **request lookup** to System.
- System performs lookup and sends **result** (e.g., password = correct) to ControlPanel.
- If password is **correct**, ControlPanel sends **request activation** to Sensors.
- Sensors respond with **activation successful** → Passed back to Homeowner.

Alternative Flow (Invalid password):

- If `numberOfTries > maxTries` and `Timer > lockedTime` → **ControlPanel** moves to **Locked** state.

Key Concepts Illustrated

- **Events as Drivers:**

- Events (from the use case) **channel behavior between objects.**

- **State Transitions:**

- State changes such as **Reading, Comparing, Locked, and Selecting** occur within the ControlPanel.

- **Input and Output Events:**

- Sequence diagrams help **identify and organize all input and output events** for system design.

Design Utility

- **Importance for Design:**
- The collated set of **events and transitions** is essential for creating an **effective system design**.
- Helps trace and verify how objects **interact and respond** to events over time.