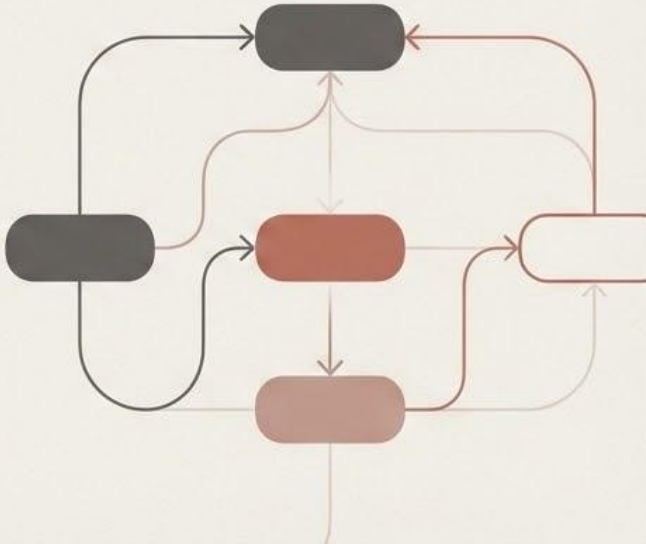
Chapter 7: Requirements Modeling

Flow, Behavior, Patterns, and WebApps

Subject: Software Engineering

Duration: 1 Hour



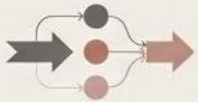
Introduction & Motivation

The Journey So Far:

- We have modeled structure (Classes/Data).
- We have modeled user interactions (Use Cases/Scenarios).

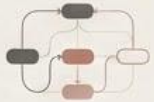
The Missing Dimensions:

Flow:



How does data transform as it moves through the system?

Behavior:



How does the system react to external events over time?

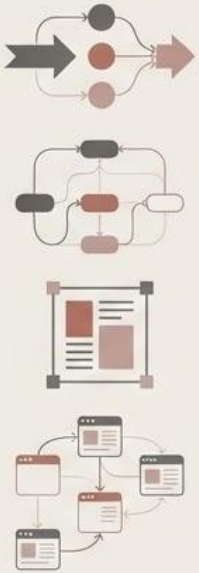
WebApps:

Addressing the unique challenges of web-based systems.

Learning Objectives

By the end of this lecture, you will be able to:

- Create a basic Data Flow Diagram (DFD) to model how data is transformed.
- Develop a State Diagram to model how the system behaves in response to specific events.
- Explain the concept of analysis patterns and how they accelerate modeling.
- Identify the specific models required for WebApps (Content, Interaction, Functional, Configuration, Navigation).



Overview of Today's Toolkit

- Flow-Oriented Modeling: Visualizing the input-process-output stream.
- Behavioral Modeling: Mapping the states and transitions of a system.
- Pattern-Based Modeling: Reusing proven analysis solutions.
- WebApp Modeling: Specializing requirements for content-heavy, interactive web environments.

Flow-Oriented Modeling

- Focus: Data Transformation (Section 7.2)
- Primary Goal: To visualize the flow of data and the specific processes that transform that data as it moves through the system.
- Key Question: "What happens to the input data to convert it into output?"
- The Core Diagram: The Data Flow Diagram (DFD).

- Note: The DFD is a structural diagram representing the "pipeline" of data processing.

Section 7.2.1: The Building Blocks

Process (Circle/Bubble): Transforms incoming data into outgoing data.

- Label: Verb phrase (e.g., "Validate Order", "Calculate Tax").

Data Flow (Arrow): Represents the movement of data packets.

- Label: Noun (e.g., "Customer Details", "Receipt").

Data Store (Parallel Lines): A repository where data rests (file, database, list).

- Label: Noun (e.g., "Customer DB", "Inventory Log").

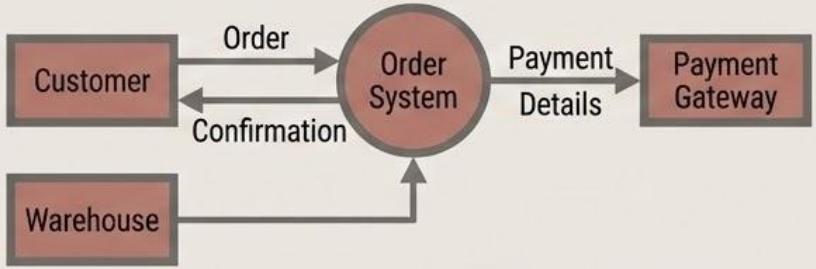
External Entity (Rectangle): A source or destination outside the system boundary (Producer/Consumer).

- Label: Noun (e.g., "Customer", "Bank API").

DFD Hierarchy: Leveling

Managing Complexity through Decomposition

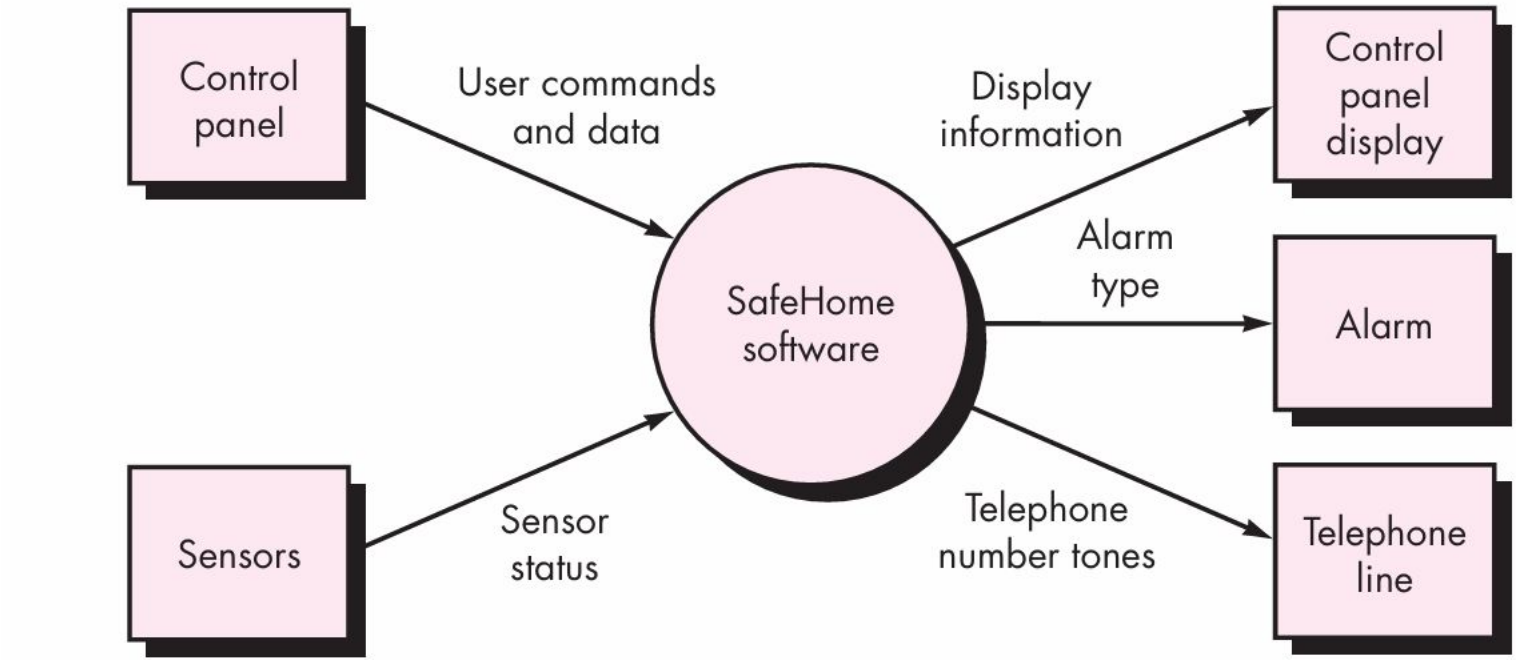
- Level 0 (Context Diagram):
  The highest level view.
- Represents the entire system as a single process bubble.
- Shows interaction with external entities only.
- Decomposition (Level 1, Level 2...):
  Breaking the "System" bubble down into smaller, detailed processes.
- Example (Online Order System):



To illustrate the use of the DFD and related notation, we again consider the *SafeHome* security function. A level 0 DFD for the security function is shown in Figure 7.1. The primary *external entities* (boxes) produce information for use by the system and consume information generated by the system. The labeled arrows represent data objects or data object hierarchies. For example, **user commands and data** encompasses all configuration commands, all activation/deactivation commands, all miscellaneous interactions, and all data that are entered to qualify or expand a command.

FIGURE 7.1

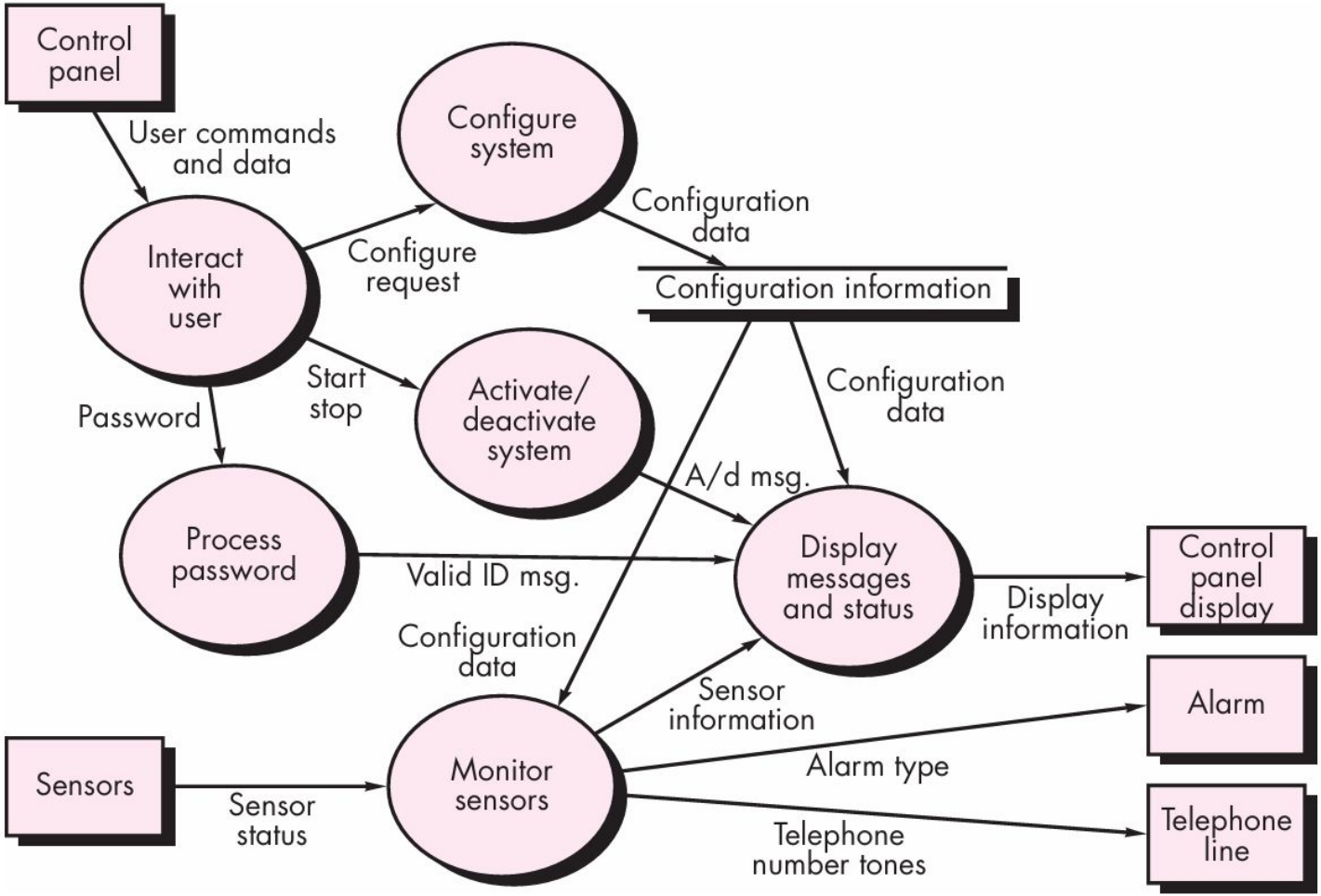
Context-level DFD for the *SafeHome* security function



Referring to the grammatical parse, verbs are *SafeHome* processes and can be represented as bubbles in a subsequent DFD. Nouns are either external entities (boxes), data or control objects (arrows), or data stores (double lines). From the discussion in Chapter 6, recall that nouns and verbs can be associated with one another (e.g., each sensor is assigned a number and type; therefore **number** and **type** are attributes of the data object **sensor**). Therefore, by performing a grammatical parse on the processing narrative for a bubble at any DFD level, you can generate much useful information about how to proceed with the refinement to the next level. Using this information, a level 1 DFD is shown in Figure 7.2. The context level process shown in Figure 7.1 has been expanded into six processes derived from an examination of the grammatical parse. Similarly, the information flow between processes at level 1 has been derived from the parse. In addition, information flow continuity is maintained between levels 0 and 1.

FIGURE 7.2

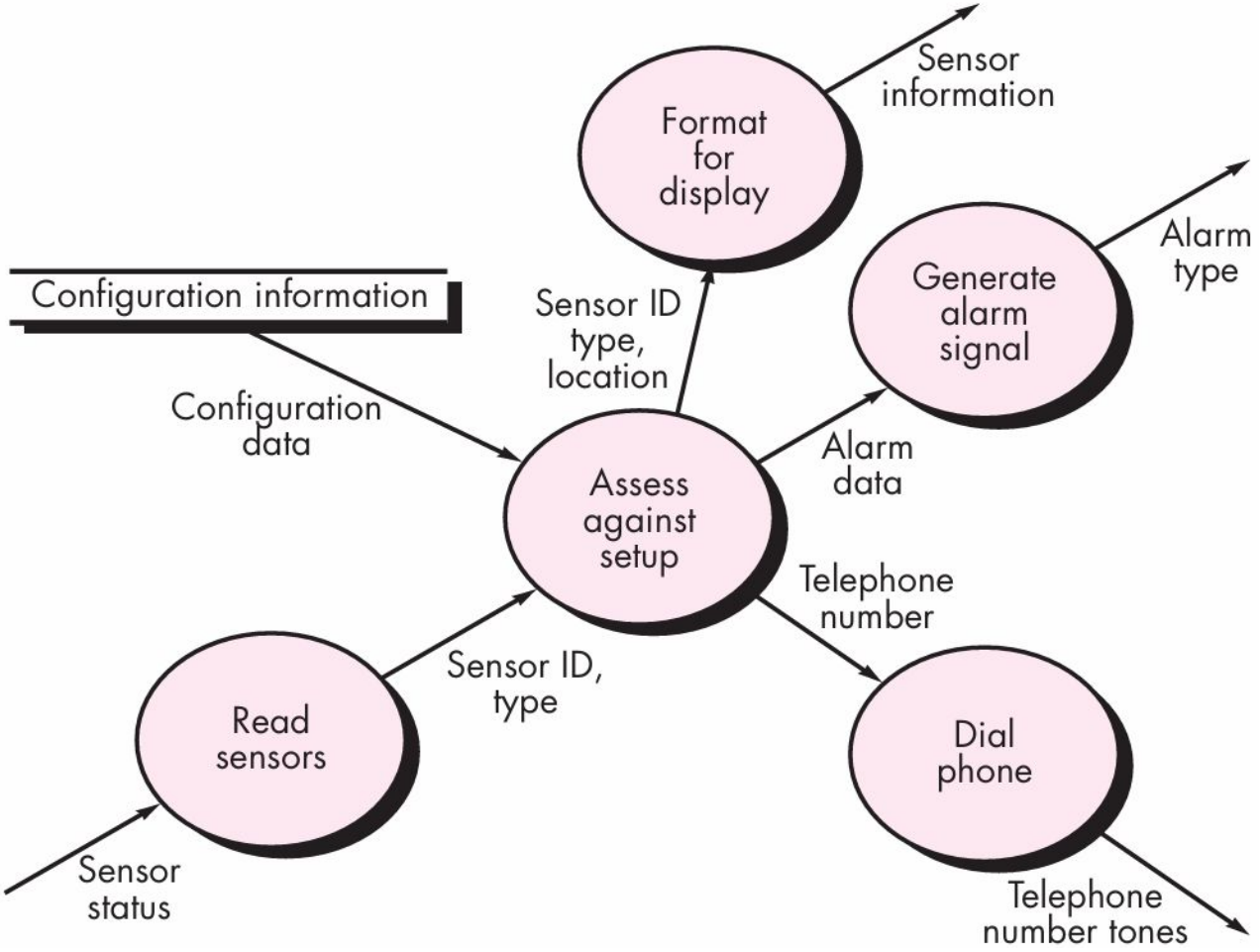
Level 1 DFD for *SafeHome* security function



The processes represented at DFD level 1 can be further refined into lower levels. For example, the process *monitor sensors* can be refined into a level 2 DFD as shown in Figure 7.3. Note once again that information flow continuity has been maintained between levels.

FIGURE 7.3

Level 2 DFD that refines the *monitor sensors* process



Control Flow Modeling

Section 7.2.2: Modeling Event-Driven Systems

Event-Driven Systems

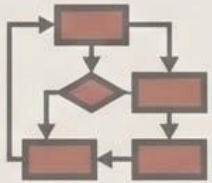
- **The Shift:** Some systems are driven by events (triggers) rather than just data streams.
- **Control Flow:** Represents control signals or events that cause a process to activate or deactivate.
- **Control Flow Diagram (CFD):**
  - Superimposed on the DFD.
  - **Notation:** Uses dashed arrows to represent control signals (vs. solid arrows for data).

Defining the Logic: CSPEC & PSPEC

Sections 7.2.3 & 7.2.4

Control Specification (CSPEC):

- Describes the logic of the control signals.
- Question: "What happens when this event triggers?"
- Format: Often a State Diagram or Decision Table.



Process Specification (PSPEC):

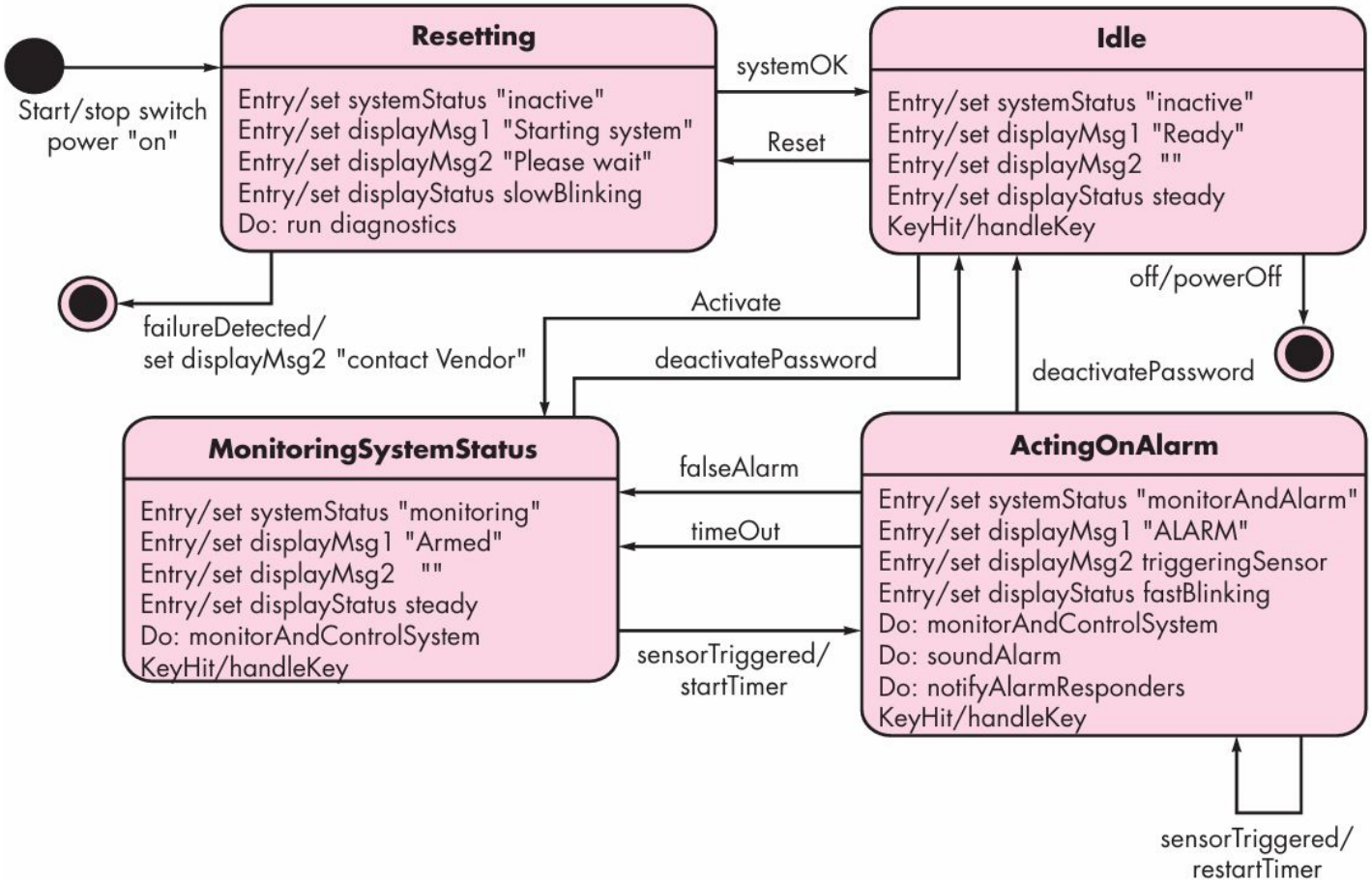
- Describes the logic of a single data process (Primitive Process).
- Question: "How exactly does this bubble convert Input A to Output B?"
- Format: Pseudocode, Structured English, or Formulas.

```
// pseuocode
IF condition THEN
  output = f(input)
...
...
```

Figure 7.4 depicts a preliminary state diagram[4] for the level 1 control flow model for *SafeHome*. The diagram indicates how the system responds to events as it traverses the four states defined at this level. By reviewing the state diagram, you can determine the behavior of the system and, more important, ascertain whether there are "holes" in the specified behavior.

For example, the state diagram (Figure 7.4) indicates that the transitions from the **Idle** state can occur if the system is reset, activated, or powered off. If the system is

FIGURE 7.4 State diagram for *SafeHome* security function



A somewhat different mode of behavioral representation is the process activation table. The PAT represents information contained in the state diagram in the context of processes, not states. That is, the table indicates which processes (bubbles) in the flow model will be invoked when an event occurs. The PAT can be used as a guide for a designer who must build an executive that controls the processes represented at this level. A PAT for the level 1 flow model of *SafeHome* software is shown in Figure 7.5.

FIGURE 7.5

Process activation table for *SafeHome* security function

input events						
sensor event	0	0	0	0	1	0
blink flag	0	0	1	1	0	0
start stop switch	0	1	0	0	0	0
display action status complete	0	0	0	1	0	0
in-progress	0	0	1	0	0	0
time out	0	0	0	0	0	1
output						
alarm signal	0	0	0	0	1	0
process activation						
monitor and control system	0	1	0	0	1	1
activate/deactivate system	0	1	0	0	0	0
display messages and status	1	0	1	1	1	1
interact with user	1	0	0	1	0	1

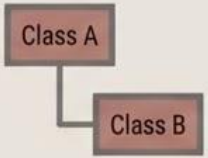
Creating a Behavioral Model

Section 7.3: Understanding System Reactivity

> The Core Question: “How does the system behave in response to events over time?”

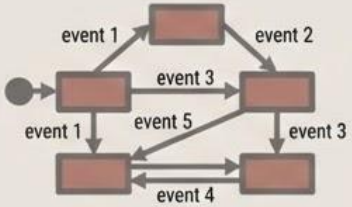
Context:

Static models (Class Diagrams) show what the system is.



Behavioral models show what the system does dynamically when poked or triggered.

Primary Tool: The State Diagram.



Identifying Events

Section 7.3.1: From Use Cases to Events

Definition of an Event:

- An occurrence that triggers a change of state in the system.

Where to find them?

- Look at your Use Cases. Every step where the actor interacts with the system, and the system must respond, represents a potential event.

Example (ATM Context):

Actor Action: “Customer inserts card.” → Event

System Reaction: “System requests PIN.” → Action/Response

State Representations

Section 7.3.2: Defining States

What is a State?

- An observable mode of behavior where the system (or a specific object) is waiting for an event to occur.

The Model: State Diagram (or Statechart Diagram).

- Used to visualize the lifecycle of an object or system:

Anatomy of a State Diagram

Key Elements & Notation

State (Rounded Rectangle):

- Represents a condition of being.

Transition (Arrow):

- The movement from one state to another.
- Label Format: Event [Guard] / Action

Initial & Final States:

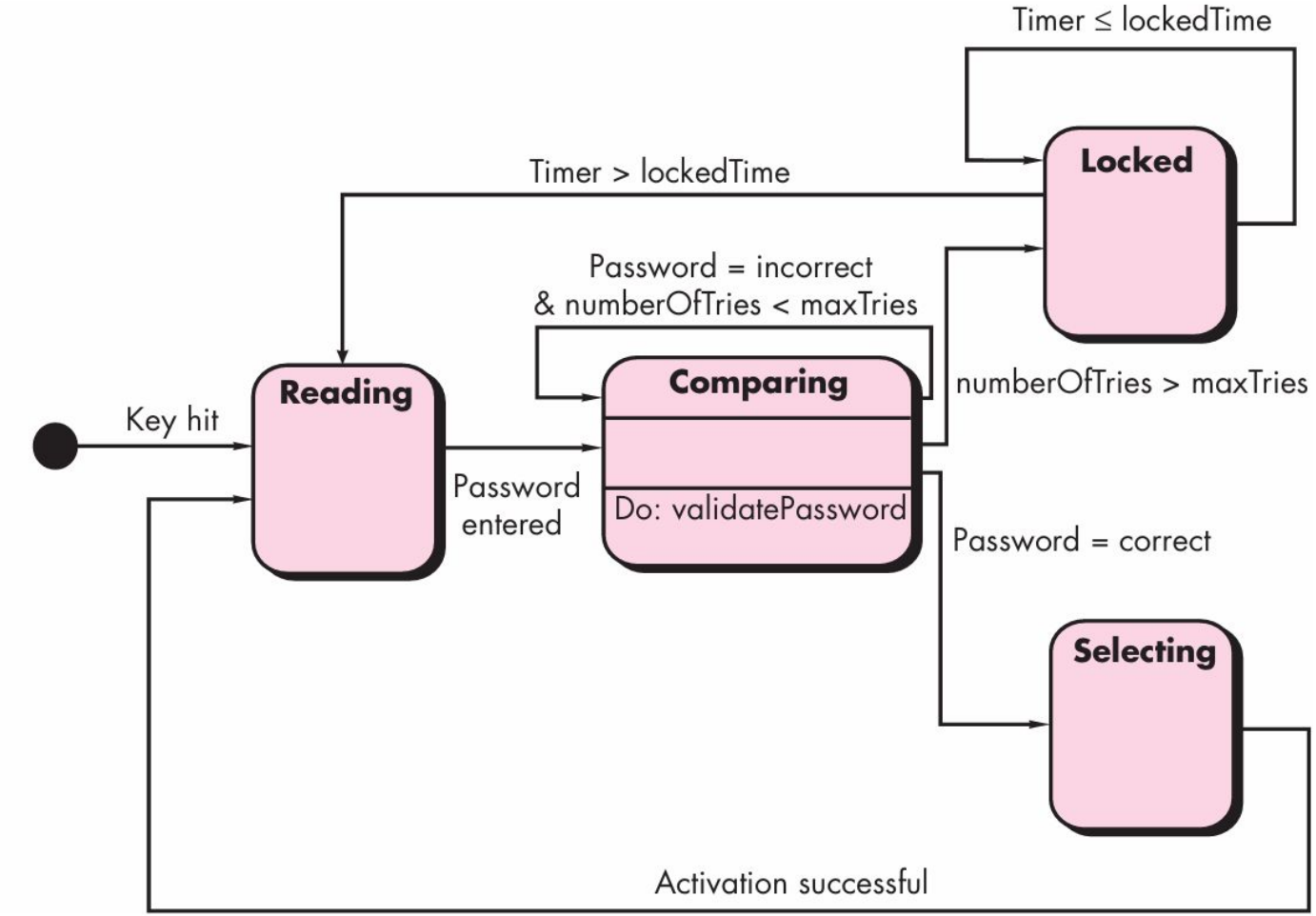
- Start: Solid filled circle.
- End: Bullseye (circle with a dot inside).

State diagrams for analysis classes. One component of a behavioral model is a UML state diagram[9] that represents active states for each class and the events (triggers) that cause changes between these active states. Figure 7.6 illustrates a state diagram for the **ControlPanel** object in the *SafeHome* security function.

Each arrow shown in Figure 7.6 represents a transition from one active state of an object to another. The labels shown for each arrow represent the event that

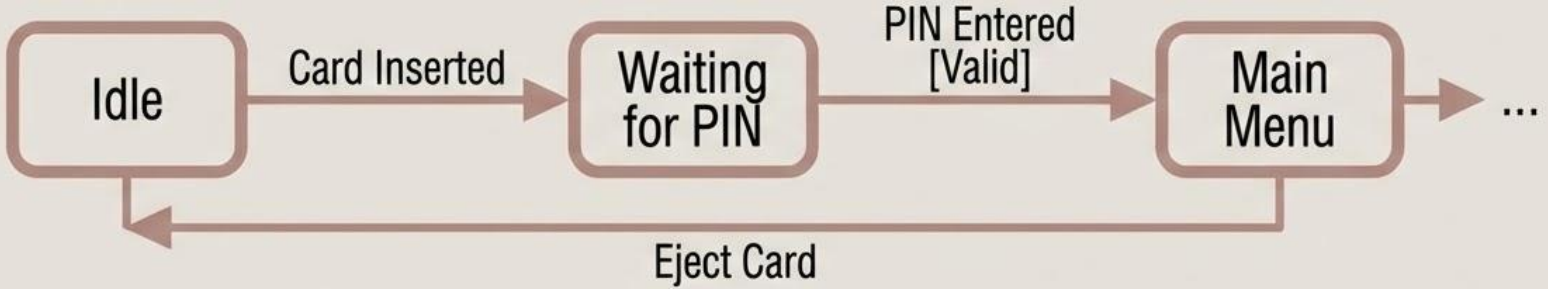
FIGURE 7.6

State diagram for the ControlPanel class



Example & Interactive Poll

Applied Example: ATM System

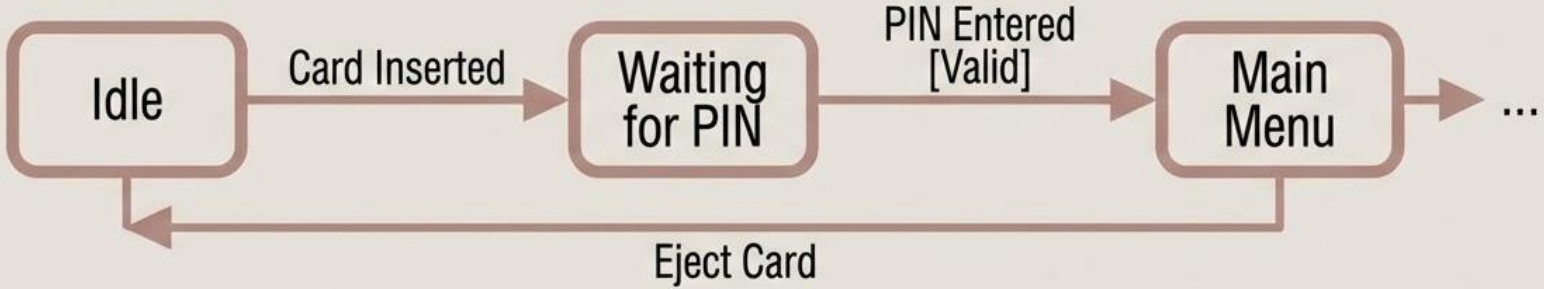


Quick Poll (2 Minutes)

- Scenario: Consider a "Network Printer."
- Question: What are two possible states for this object?

Example & Interactive Poll

Applied Example: ATM System

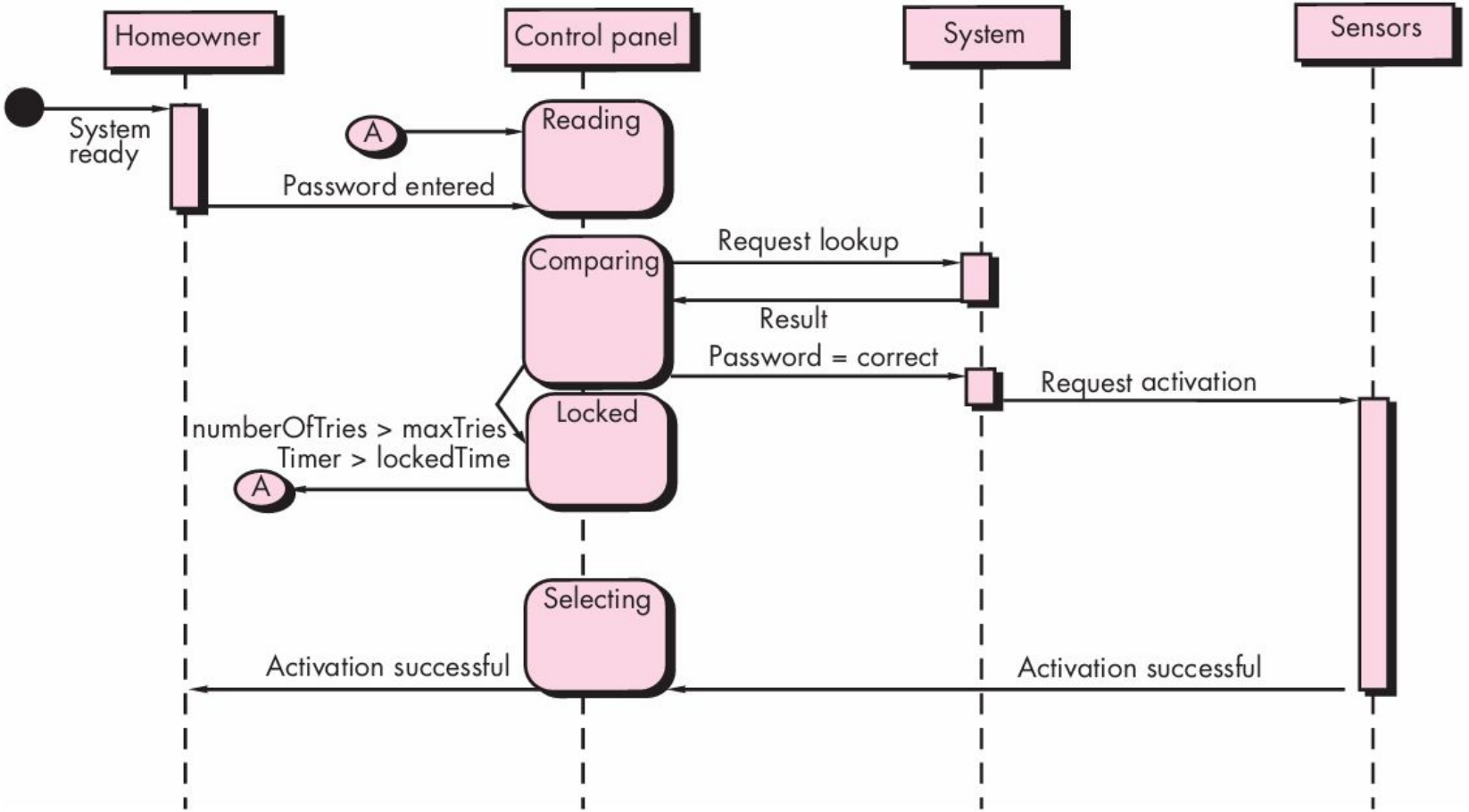


Quick Poll (2 Minutes)

- Scenario: Consider a "Network Printer."
- Question: What are two possible states for this object?
  Possible Answers: "Ready," "Printing," "Out of Paper," "Offline," "Warming Up."

Figure 7.7 illustrates a partial sequence diagram for the *SafeHome* security function. Each of the arrows represents an event (derived from a use case) and indicates how the event channels behavior between *SafeHome* objects. Time is measured vertically (downward), and the narrow vertical rectangles represent time spent in processing an activity. States may be shown along a vertical time line.

FIGURE 7.7 Sequence diagram (partial) for the *SafeHome* security function



Patterns for Requirements Modeling

Section 7.4: Accelerating Analysis with Patterns

Definition:

An Analysis Pattern is a named problem/solution pair that captures a recurring structure within a specific application domain.

The Goal:

Don’t reinvent the wheel. If a problem has been solved effectively before in a similar context, reuse that conceptual structure.

Discovering Analysis Patterns

Section 7.4.1: Finding Synergies

Where to look:

Identify conceptual synergies across projects within the same domain (e.g., Healthcare, Finance, Automotive).

Common Examples:

- The “Observer” Pattern: Useful for monitoring systems where one object needs to notify others of changes.
- The “Transaction” Pattern: Essential for financial systems requiring atomic operations (all-or-nothing execution).

Example: The Actuator-Sensor Pattern

Section 7.4.2: A Staple of Real-Time Systems

Context:

Extremely common in embedded systems and real-time software.

The Structure:

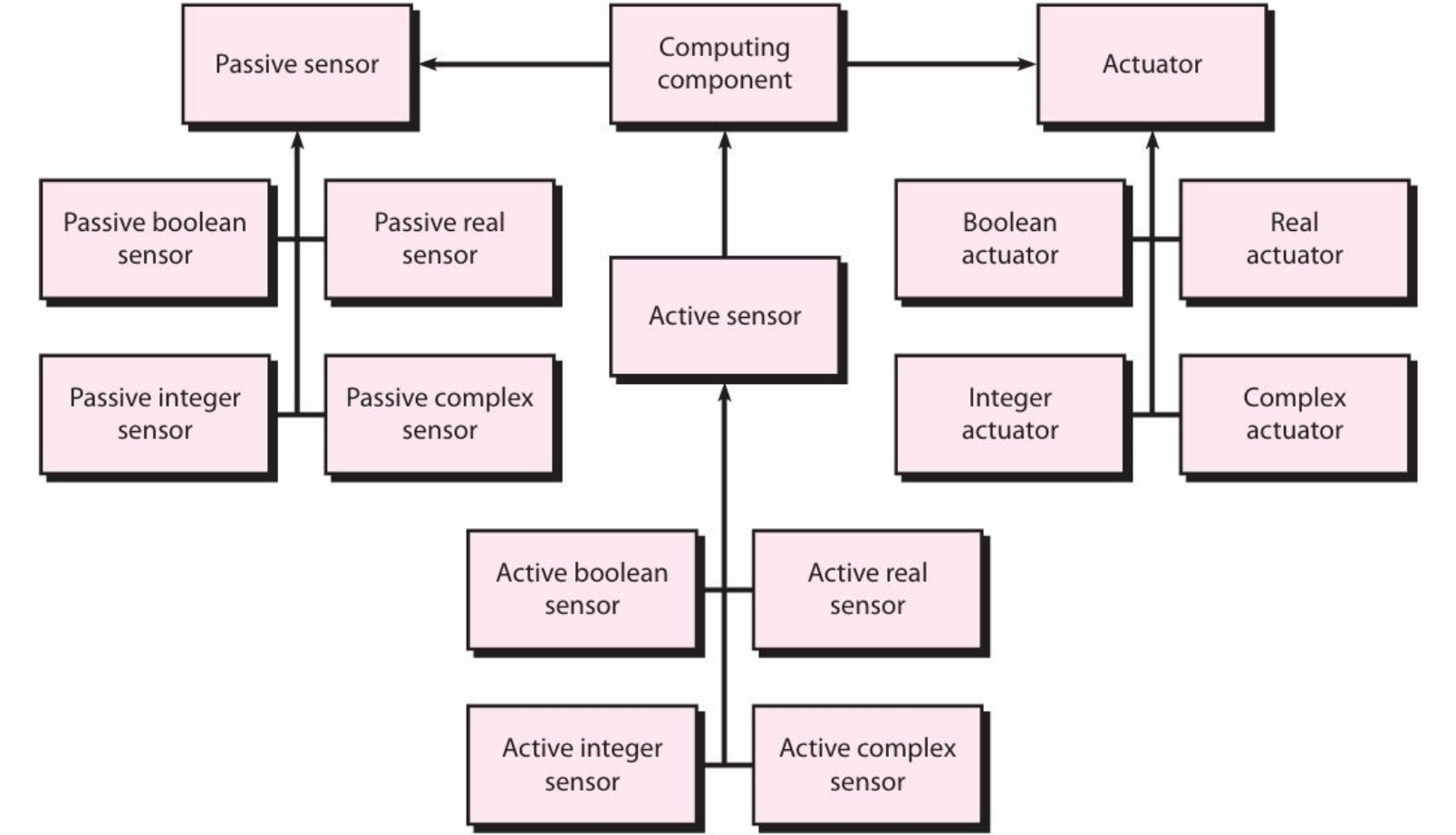
Sensor: An object (passive or active) that collects data from the physical environment.

Controller/Algorithm: The central logic that processes sensor data and decides on an action.

Actuator: An object that executes the command to cause a change in the environment.

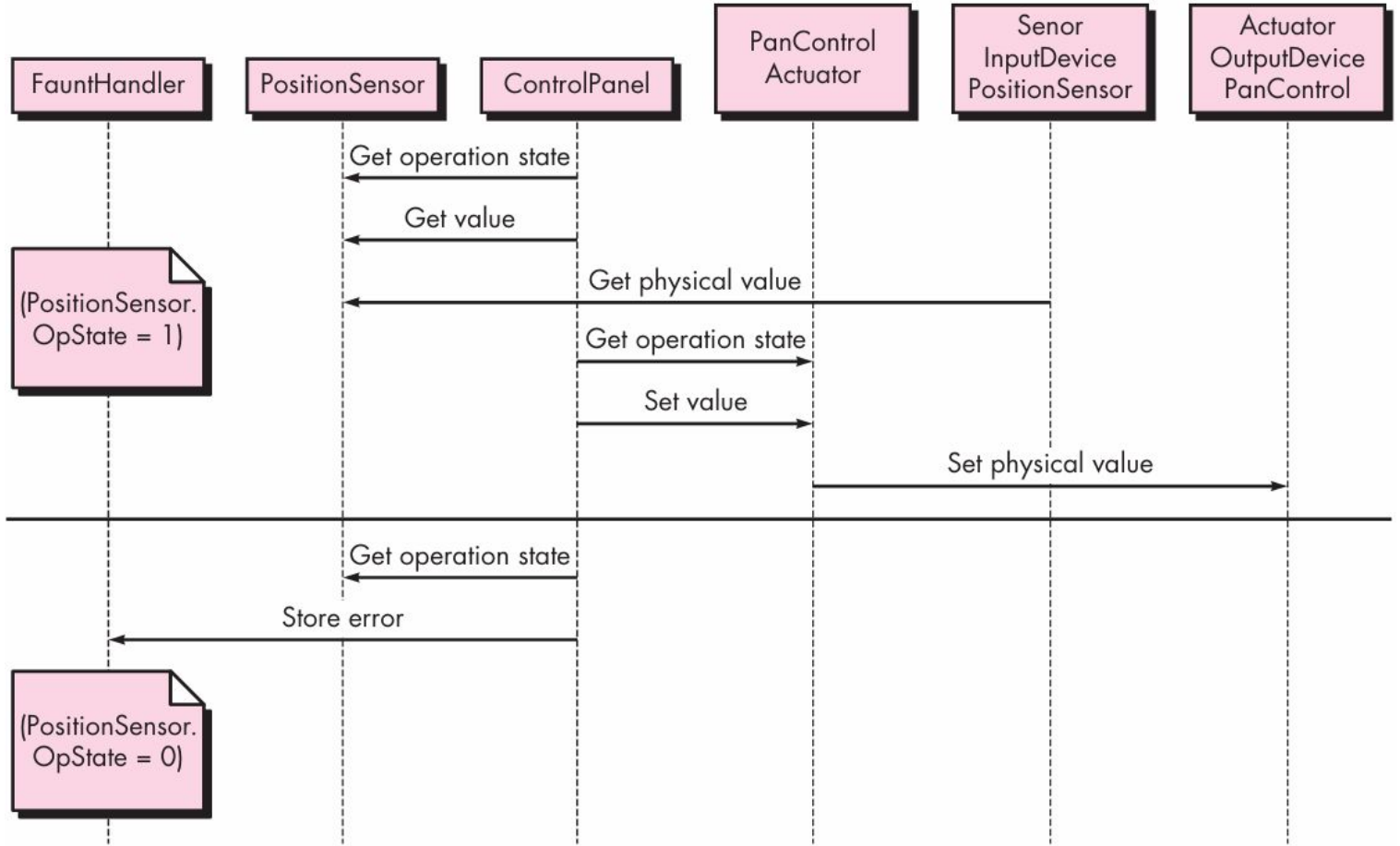
Structure. A UML class diagram for the **Actuator-Sensor** pattern is shown in Figure 7.8. **Actuator, PassiveSensor,** and **ActiveSensor** are abstract classes and denoted in italics. There are four different types of sensors and actuators in this pattern.

FIGURE 7.8 UML sequence diagram for the Actuator-Sensor pattern. *Source:* Adapted from [Kon02] with permission.



Behavior. Figure 7.9 presents a UML sequence diagram for an example of the **Actuator-Sensor** pattern as it might be applied for the *SafeHome* function that controls the positioning (e.g., pan, zoom) of a security camera. Here, the **ControlPanel**[13] queries a sensor (a passive position sensor) and an actuator (pan control) to check the operation state for diagnostic purposes before reading or setting a value. The messages *Set Physical Value* and *Get Physical Value* are not messages between objects. Instead, they describe the interaction between the physical devices of the system and their software counterparts. In the lower part of the diagram, below the horizontal line, the **PositionSensor** reports that the operation state is zero. The **ComputingComponent** (represented as **ControlPanel**) then sends the error code for a position sensor failure to the **FaultHandler** that will decide how this error affects the system and what actions are required. It gets the data from the sensors and computes the required response for the actuators.

FIGURE 7.9 UML Class diagram for the Actuator-Sensor pattern. *Source:* Reprinted from [Kon02] with permission.



Applications & Benefits

Why recognize this pattern?

Ubiquitous Application:

- **Thermostats**: Sensor (Thermometer) → Controller (CPU) → Actuator (Heater/AC).
- **Robotics**: Sensor (Camera/Lidar) → Controller (Pathfinding) → Actuator (Motor/Steering).
- **Industrial Automation**: Conveyor belts, assembly arms.

Benefit:

- Recognizing this pattern allows analysts to immediately plug in the standard structure, significantly speeding up the modeling process.

Requirements Modeling for WebApps

Section 7.5: Specialized Approaches for the Web

The Premise:

WebApps have unique characteristics that demand specialized modeling:

- Continuous evolution (always changing).
- Heavy focus on content (text, media).
- Complex navigation structures.

Scope & Input/Output

Sections 7.5.1 - 7.5.3

How Much Analysis is Enough?

The Key:

Provide sufficient detail to avoid ambiguity, but remain flexible enough to accommodate rapid change.

The Agile Approach:

“Just enough” modeling to understand scope and architecture. Over-modeling is considered waste.

The Inputs:

- Stakeholders, business context, user categories, and existing content assets.

The Outputs:

- A set of interrelated models (Content, Interaction, Functional, Navigation, Configuration).

The Content Model

Section 7.5.4: Structuring Information

Definition: A structural model of all content objects presented to the user.

What it includes:

- Text, graphics, video, audio, and dynamic data.

Representation: Often a Hierarchical Tree or UML Class Diagram.

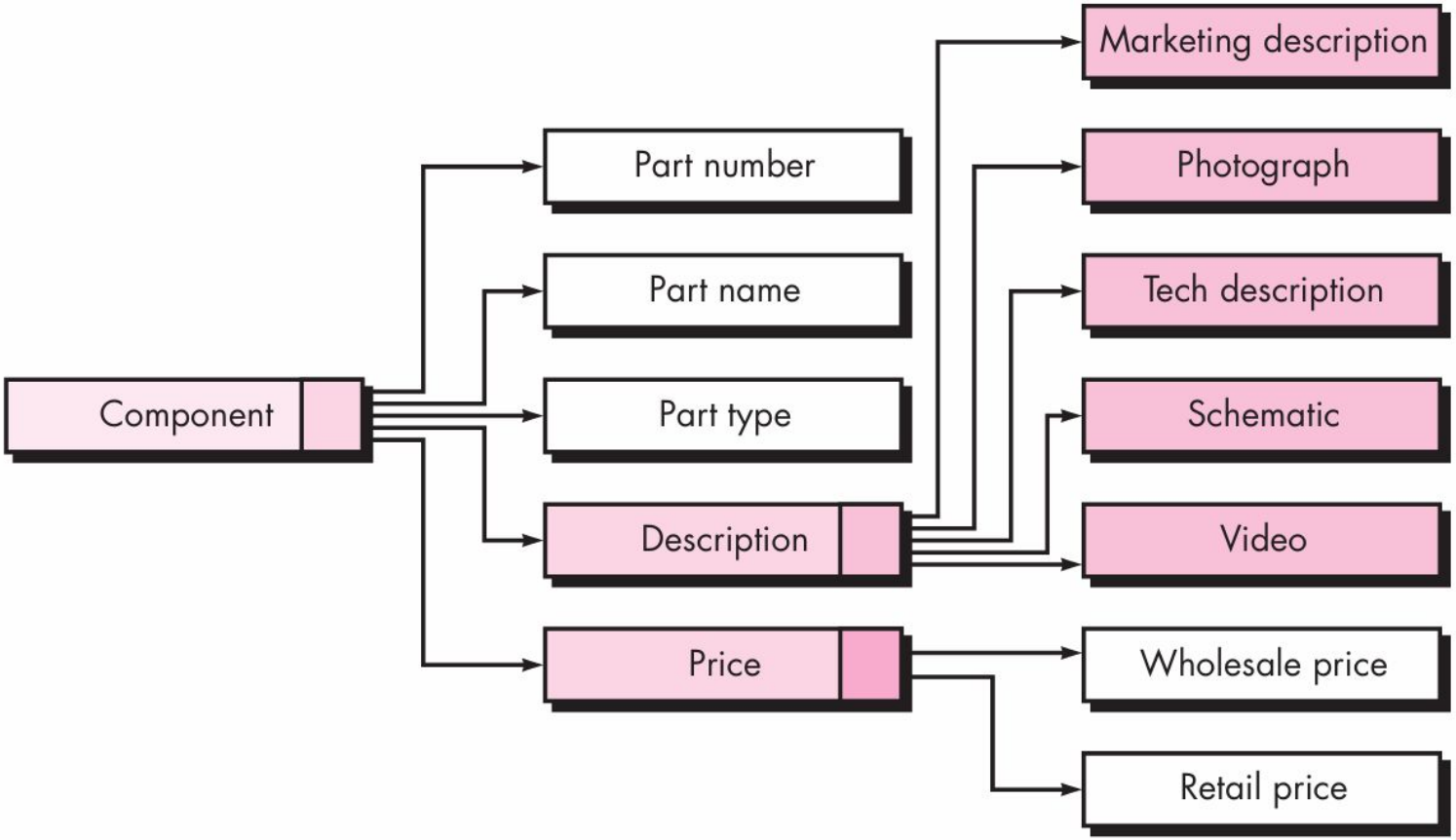
Example:

A Product object contains child objects like ImageGallery, Description, Specifications, and Reviews.

For example, consider the *data tree* [Sri01] created for a **SafeHomeAssured.com** component shown in Figure 7.10. The tree represents a hierarchy of information that is used to describe a component. Simple or composite data items (one or more data

FIGURE 7.10

Data tree for a **SafeHome-Assured.com** component



Interaction & Functional Models

Sections 7.5.5 - 7.5.6: Behavior and Processing

Interaction Model (User Focus)

What: Describes how the user interacts with the interface.

Tools: Use Cases (scenarios), Sequence Diagrams (complex interactions), and UI Prototypes/Wireframes.

Goal: Define the User Experience (UX) flow.

Functional Model (System Focus)

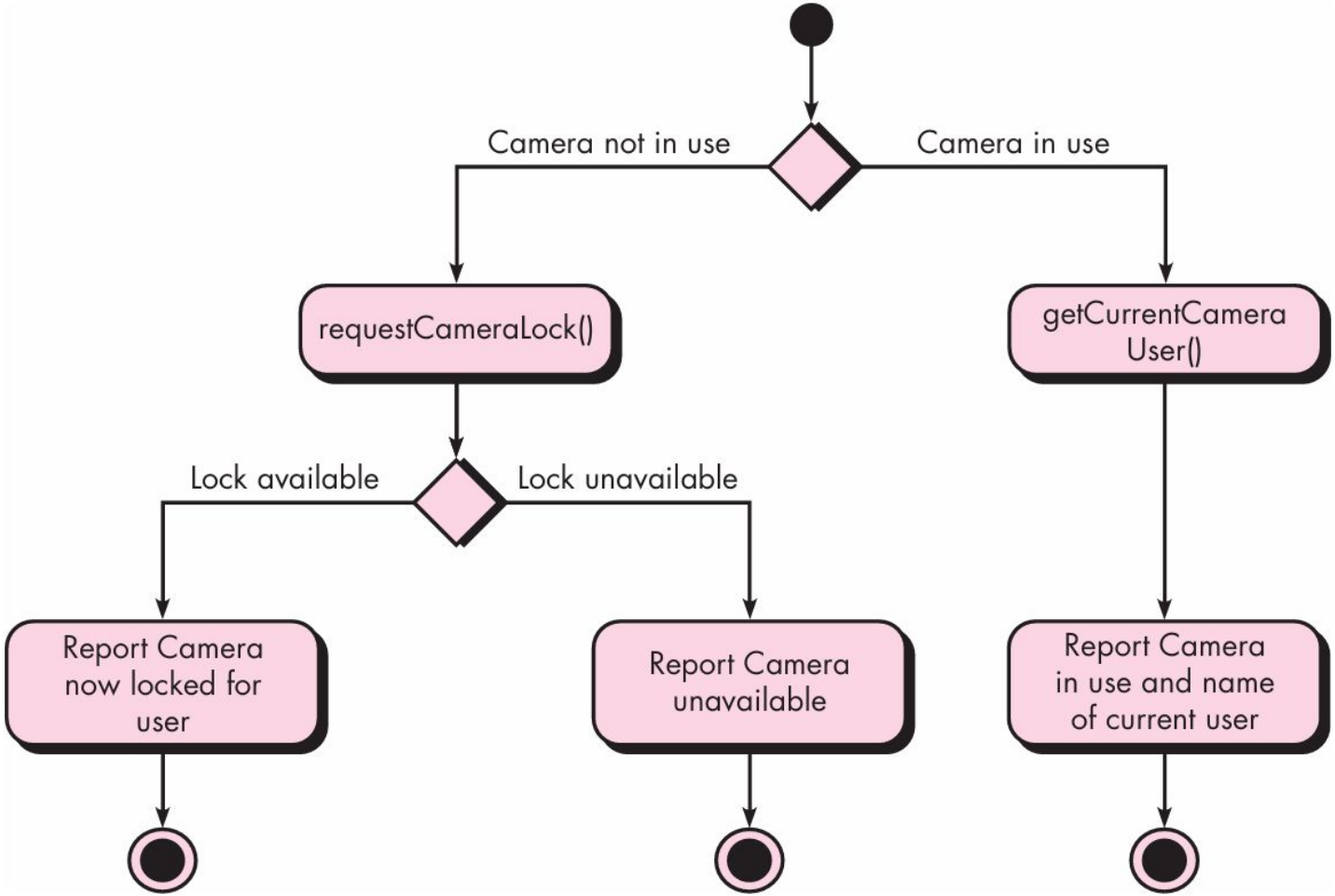
What: Describes operations applied to content objects and server-side processing.

Tools: DFD fragments or UML Activity Diagrams for business logic.

Figure 7.11 depicts an activity diagram for the *takeControlOfCamera()* operation that is part of the **Camera** analysis class used within the *Control cameras* use case. It should be noted that two additional operations are invoked with the procedural flow: *requestCameraLock(),* which tries to lock the camera for this user, and *getCurrentCameraUser(),* which retrieves the name of the user who is currently controlling the camera. The construction details indicating how these operations are invoked and the interface details for each operation are not considered until WebApp design commences.

FIGURE 7.11

Activity diagram for the *takeControlOf-Camera()* operation

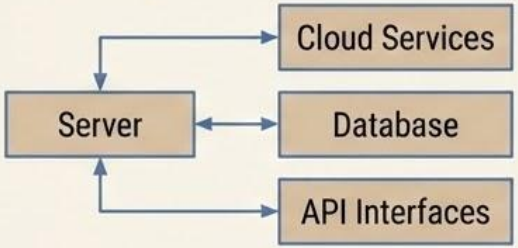


Configuration & Navigation Models

Sections 7.5.7 - 7.5.8: Environment and Movement

Configuration Model

Describes the environment and infrastructure (Server config, Cloud services, Database systems, API interfaces).



Navigation Model (Unique to Hypermedia)

Critical Question: “How does the user get from here to there?”

Tools: Navigation Maps (Directed graphs where nodes = content/functions).

Syntax: Defines patterns like "linear," "tree," or "network."

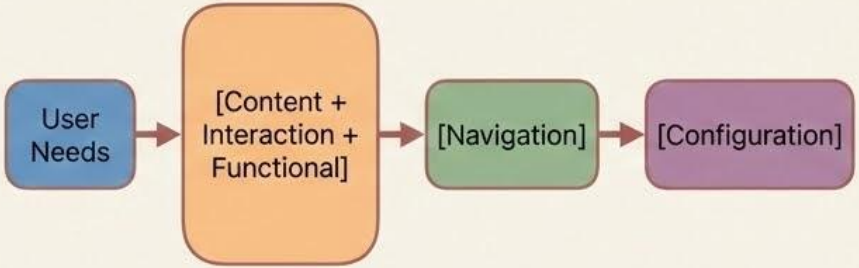
Synthesis: Putting it Together

The WebApp Modeling Flow

The Big Picture

- User Needs drive the creation of:
  - Content Model
  - Interaction Model
  - Functional Model
- The Navigation Model links these three together.
- The entire system is deployed within the Configuration Model.

Visual Logic

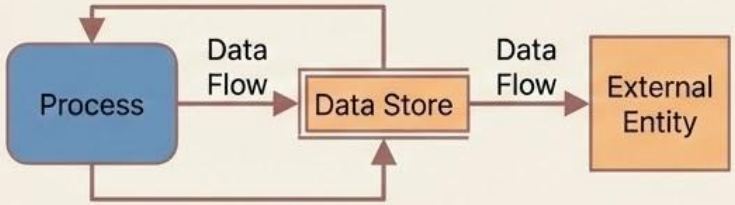


Summary of New Dimensions

Flow & Behavior

Flow Models (DFDs)

Visualize how data is transformed as it moves through the system.



- Key Elements: Processes, Data Stores, and External Entities.

Behavioral Models (State Diagrams)

Visualize how the system reacts to events over time.

- Key Concept: The system changes State in response to external triggers.

Advanced Modeling Strategies

Patterns & WebApps

Analysis Patterns

- Provide reusable templates for common domain problems (e.g., Actuator-Sensor).
- Benefit: Improves modeling efficiency and reduces errors.

WebApp Modeling

Requires a specialized, multi-model approach due to unique nature.

The 5 Key Models:

- Content: What is shown?
- Interaction: How do users act?
- Functional: Server-side logic.
- Configuration: Environment/Infrastructure.
- Navigation: How users move through the app (Crucial!).

The Complete Modeling Toolkit

Synthesizing the Views

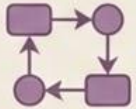
Structure

Class Diagrams, Data Models (What is it?)

Interaction

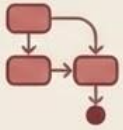
Use Cases (What does the user do?)

Flow



Data Flow Diagrams (How does data move?)

Behavior



State Diagrams (How does it react?)

Reuse

Analysis Patterns (Has this been solved before?)

Web-Specific

Navigation Maps, Content Models (How is it navigated?)

Final Thought

> "A great software architect is a master modeler. They can look at a complex problem and choose the right combination of these views—structure, flow, behavior—to create a blueprint that is comprehensible, precise, and builds confidence before a single line of code is written."