投票 最多可选1项

设置

Are you ready?

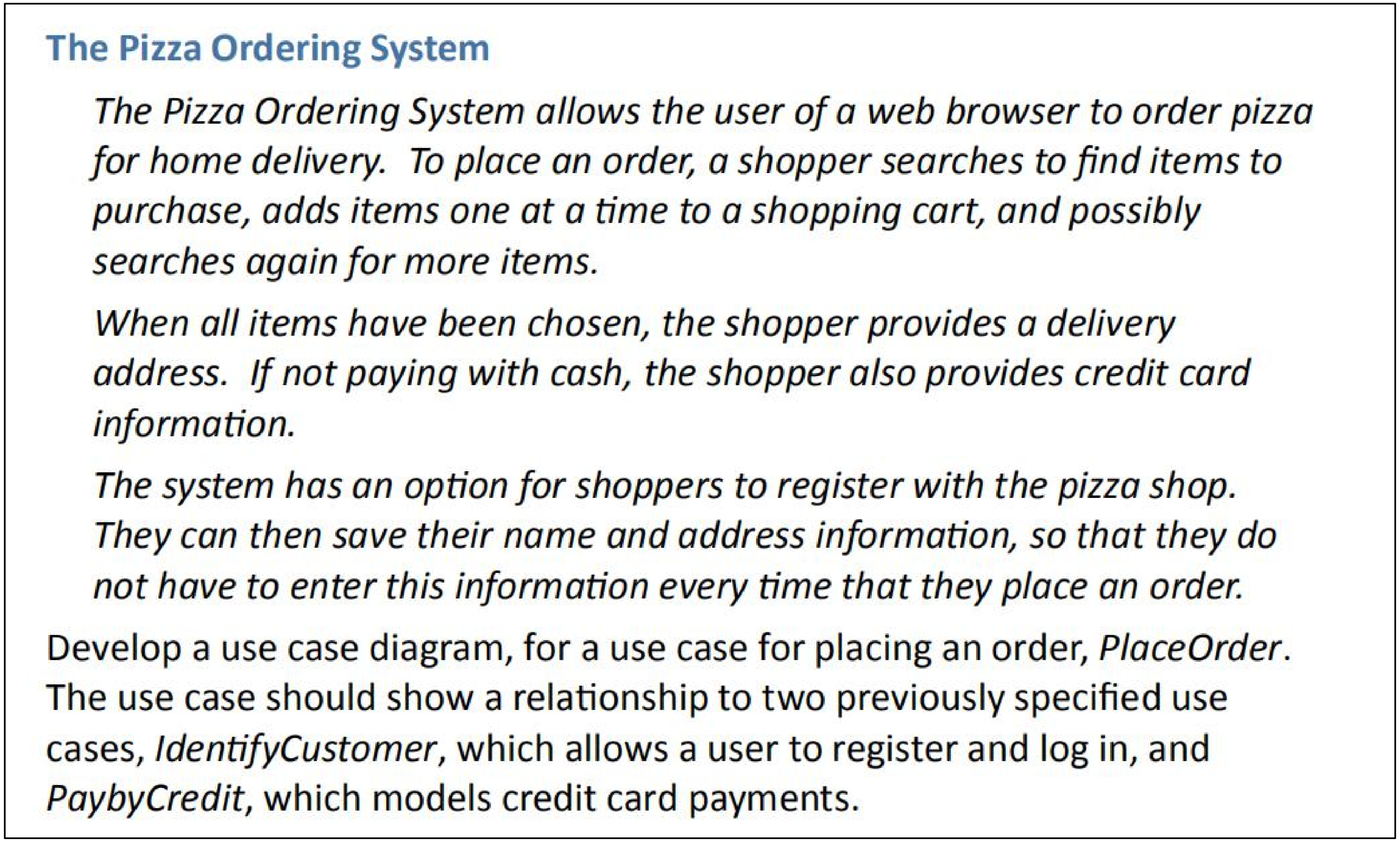
A Yes

B No

提交

Review – Ch9 Use case Diagram

- Quiz: An Old Examination Question



The Pizza Ordering System

The Pizza Ordering System allows the user of a web browser to order pizza for home delivery. To place an order, a shopper searches to find items to purchase, adds items one at a time to a shopping cart, and possibly searches again for more items.

When all items have been chosen, the shopper provides a delivery address. If not paying with cash, the shopper also provides credit card information.

The system has an option for shoppers to register with the pizza shop. They can then save their name and address information, so that they do not have to enter this information every time that they place an order.

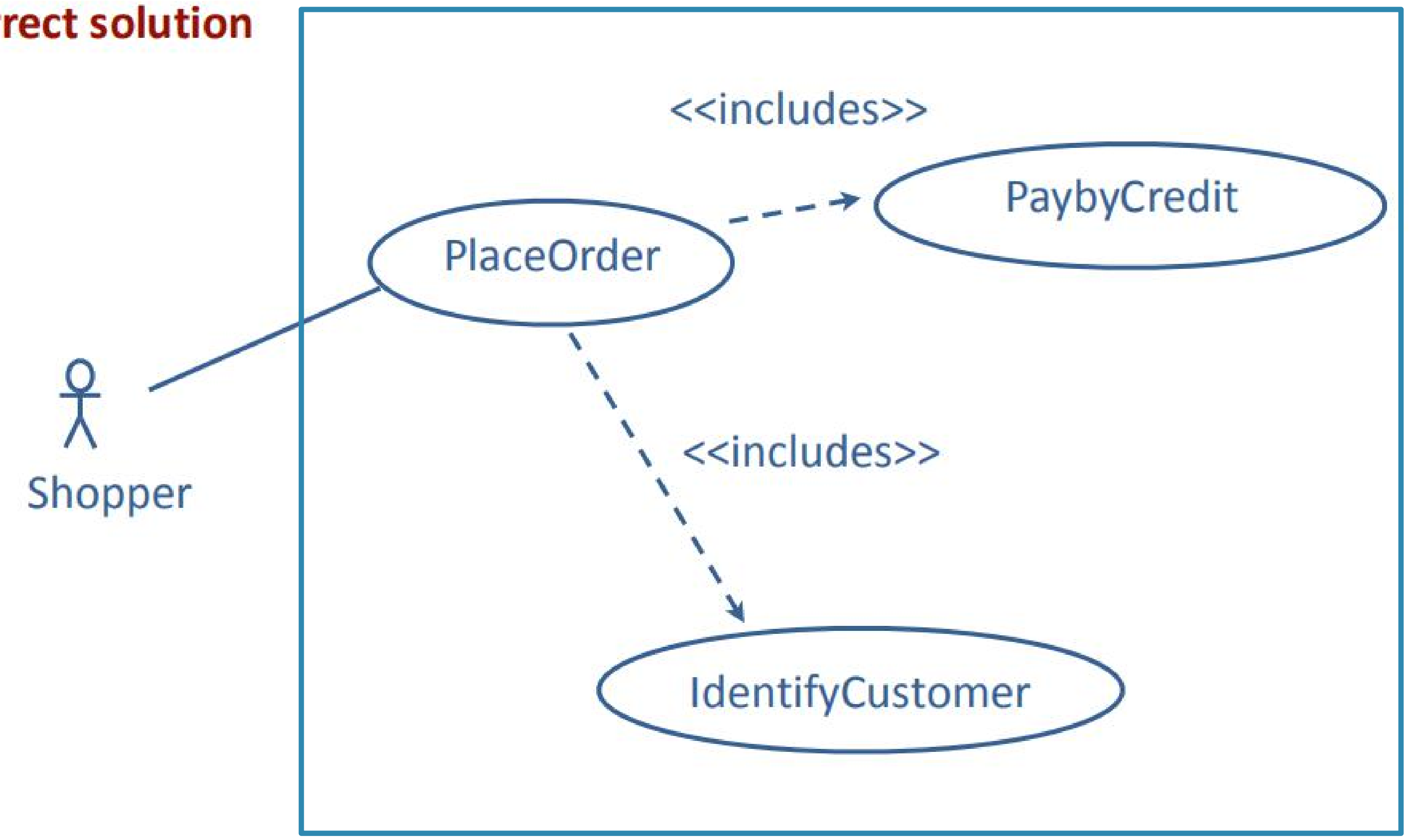
Develop a use case diagram, for a use case for placing an order, *PlaceOrder*. The use case should show a relationship to two previously specified use cases, *IdentifyCustomer*, which allows a user to register and log in, and *PaybyCredit*, which models credit card payments.

Cited from: CS5150 Cornel University slide

Review – Ch9 Use case Diagram

- An Old Examination Question

Correct solution



Cited from: CS5150 Cornel University slide

Software Engineering

Part 2
Modeling

Chapter 11
Requirements Modeling: Behavior, Patterns, and Web/Mobile Apps

Reproduced by Ning Li , 2022

Contents

11.1 Creating behavioral model

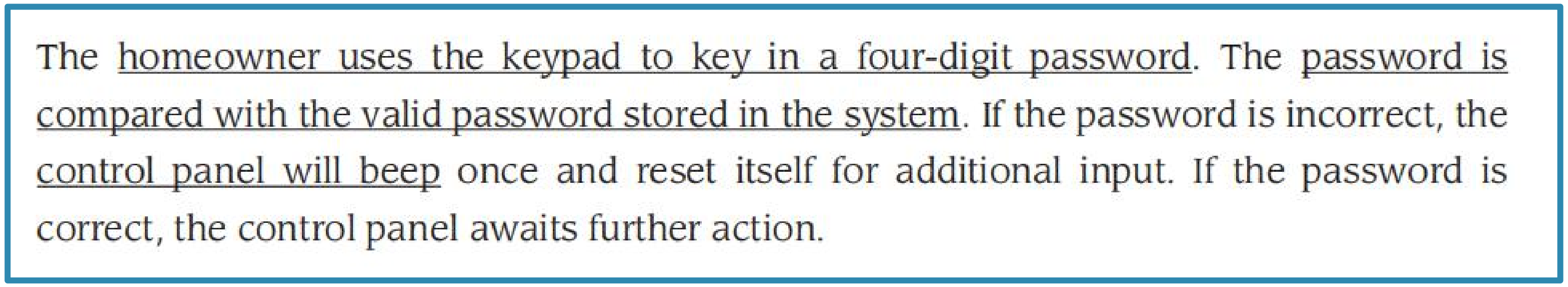
- The behavioral model indicates how software will respond to external events. To create the model, the analyst must perform the following steps:
  1. Evaluate all use cases to fully understand the sequence of interaction within the system.
  2. Identify events that drive the interaction sequence and 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.

11.2 Identifying Events with the Use Case

1. Identifying events.
2. Allocated events to the objects.

    Objects can be responsible for

    - generating events
    - recognizing events



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.

Event1: password entered (Homeowner: generating events)

Event2: password compare (ControlPanel: recognizing events)

Event3: control panel warning (ControlPanel: generating events)

11.3 State Representations

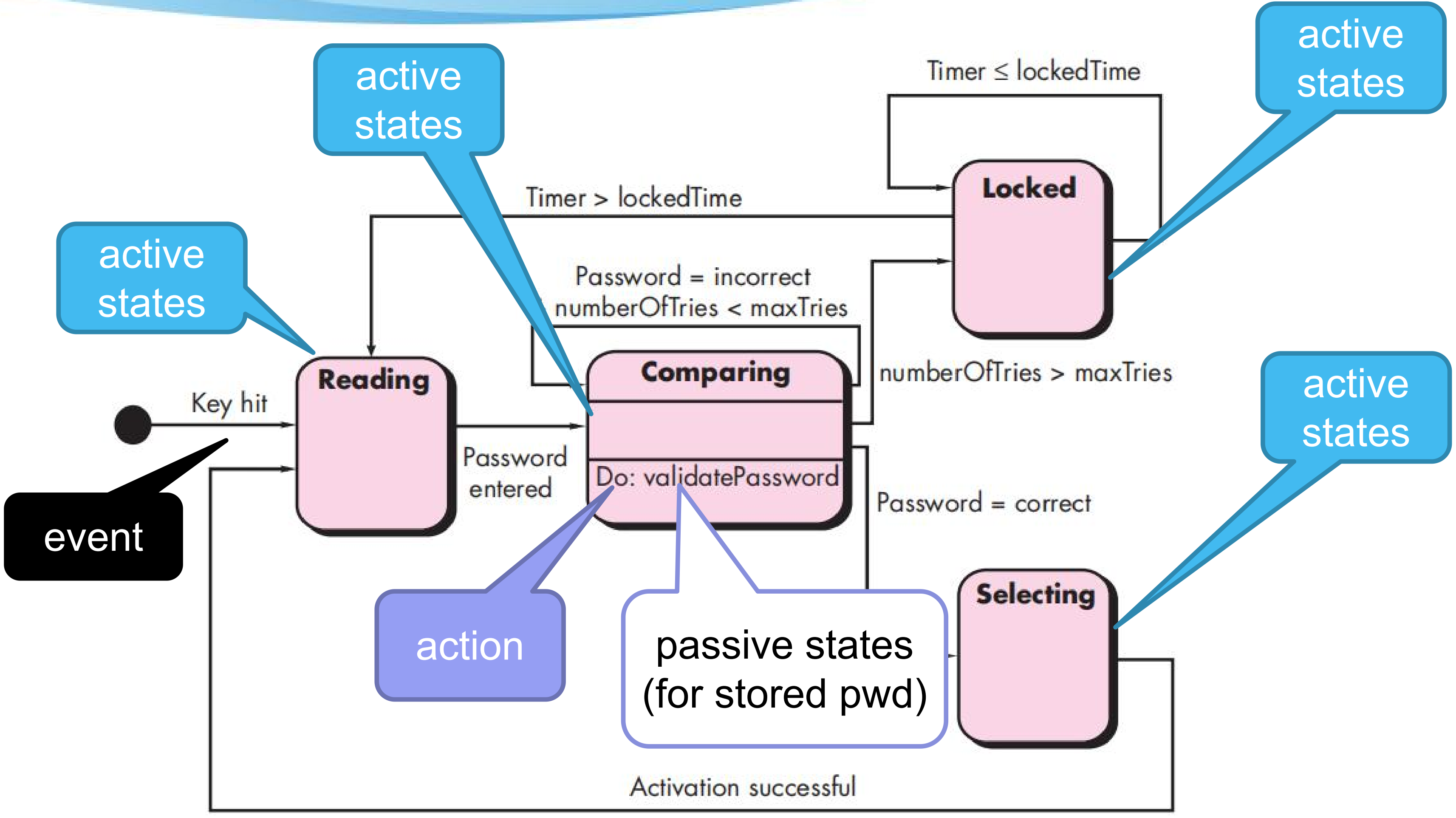
- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function
  - the state of the system as observed from the outside

- The state of a class takes on both passive and active characteristics.
  - *passive state:*
    simply the current status of all of an object's attributes.
    (Player: position and orientation ...)
  - *active state:*
    the current status of the object as it undergoes a continuing transformation or processing.
    (Player: moving, at rest, injured, trapped, lost...)

11.3 The States of a System

- state—a set of observable circum-stances that characterizes the behavior of a system at a given time
- state transition—the movement from one state to another
- event—an occurrence that causes the system to exhibit some predictable form of behavior
- action—process that occurs as a consequence of making a transition

11.3 State Diagram for the ControlPanel Class



how an individual class changes state based on external events

Take a break

One minutes

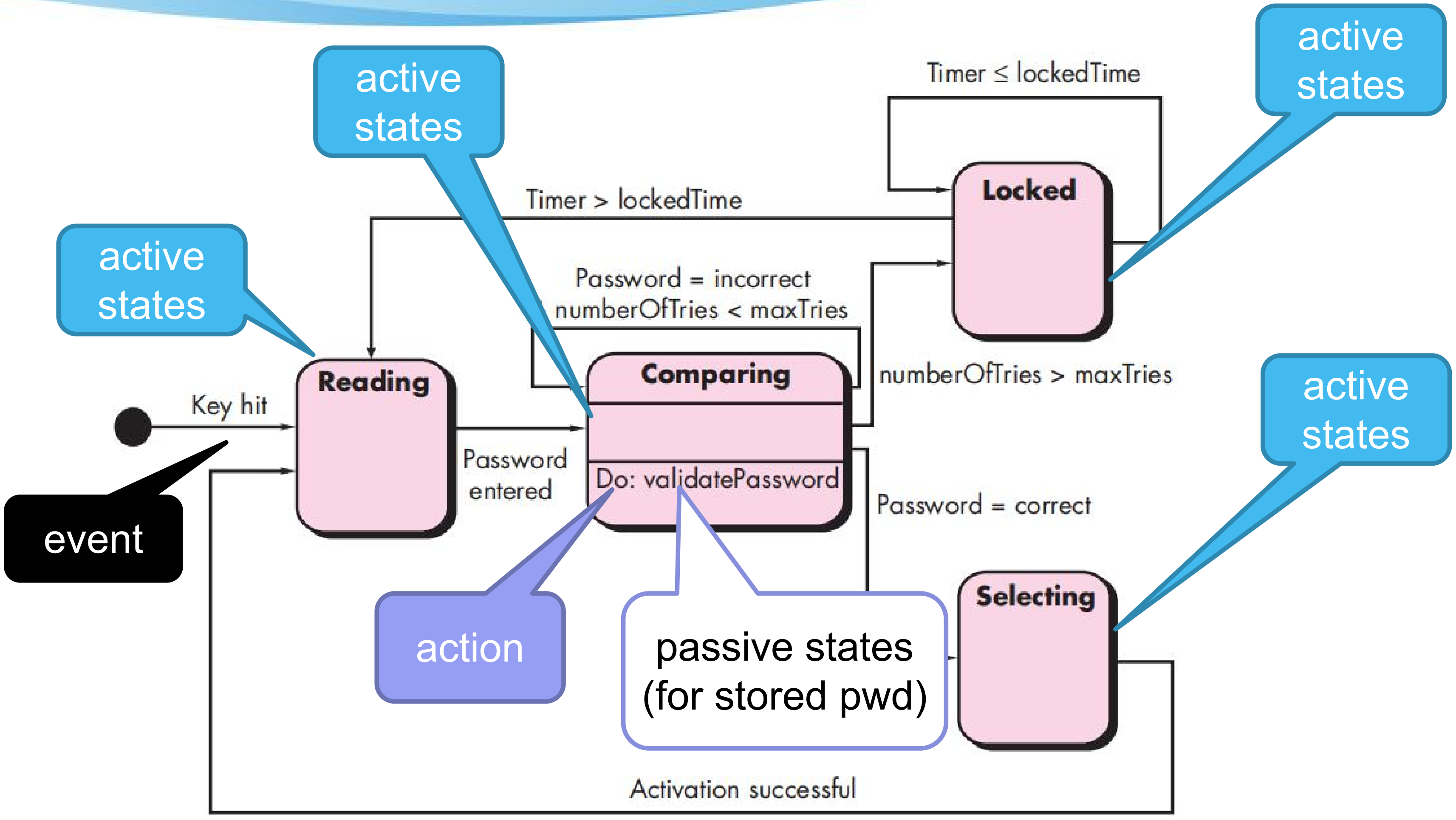
主观题 10分

设置

There are two different types of “states” that behavioral models can represent. What are they?

作答

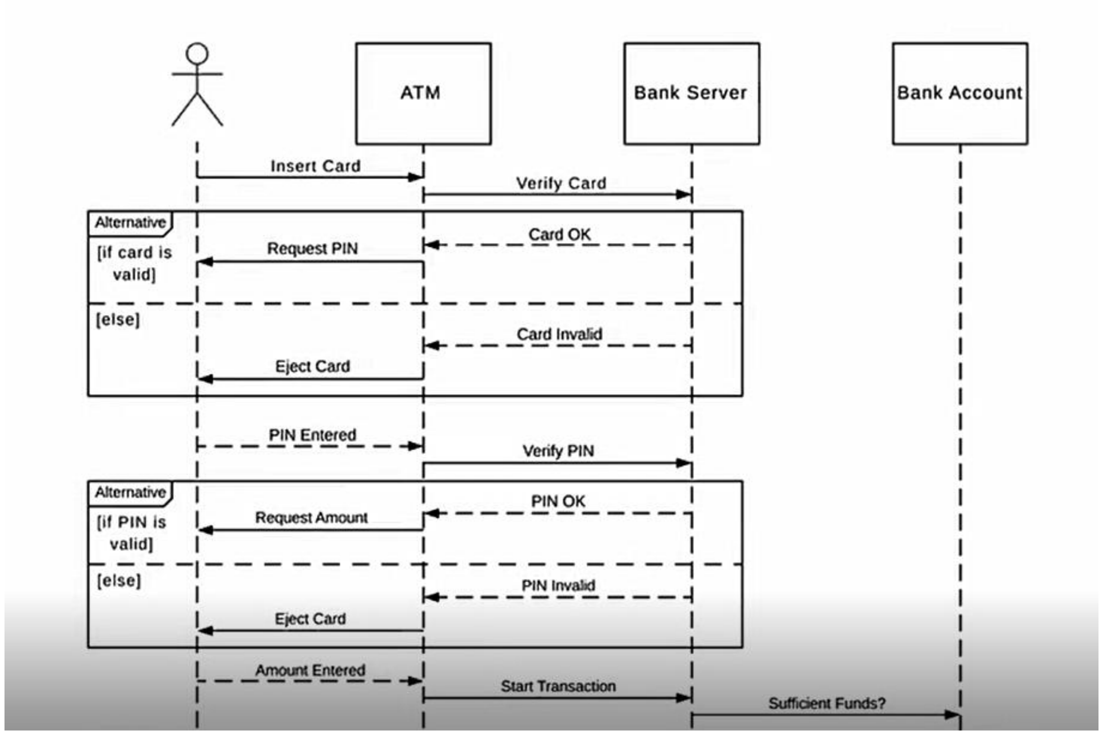
11.3 State Diagram for the ControlPanel Class



how an individual class changes state based on external events

11.3 Sequence diagram

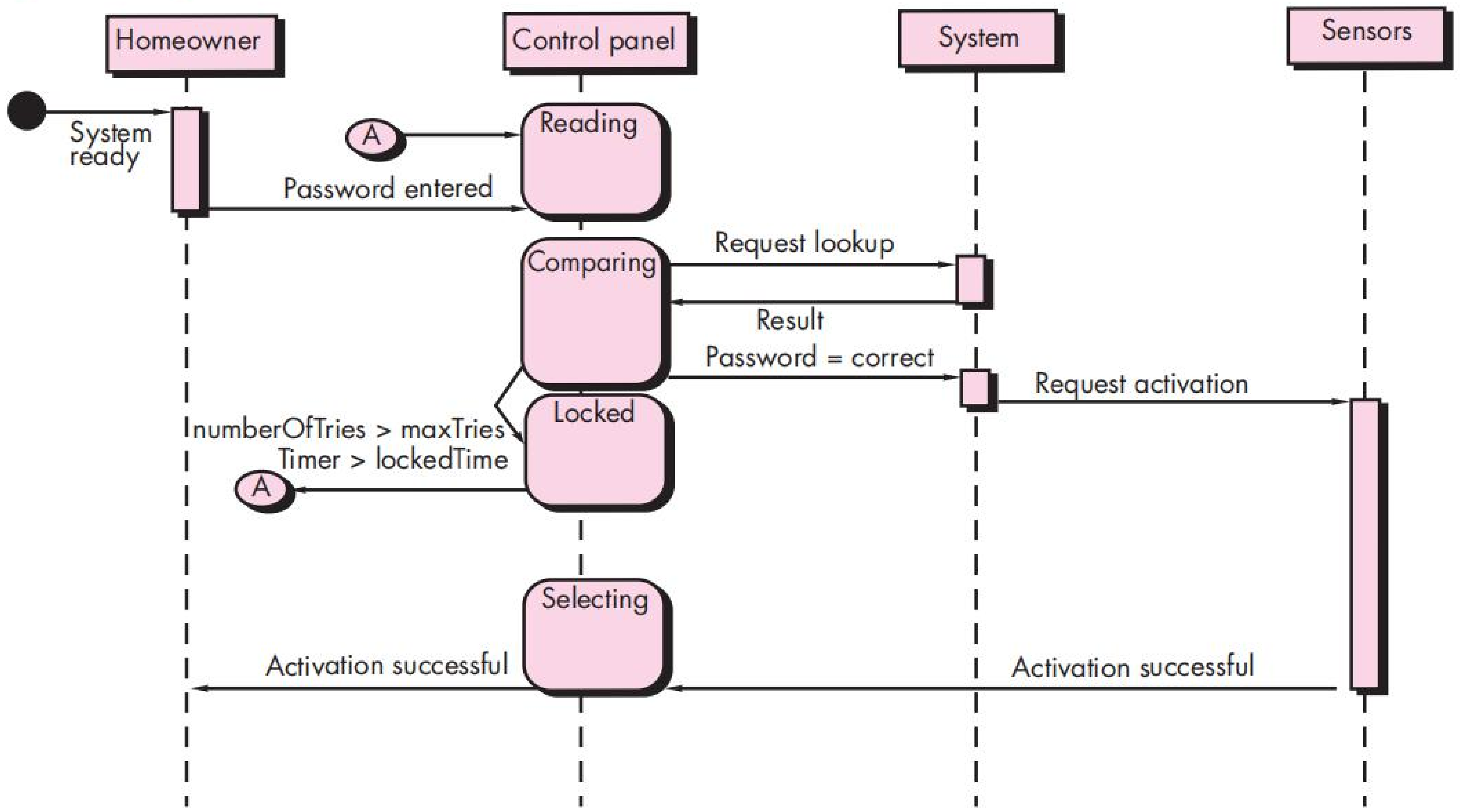
Let's watch a video!



Sequence diagram indicates how events cause transitions from object to object.

11.3 sequence diagram

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



the behavior of the software as a function of time

主观题 10分

设置

How does a sequence diagram differ from a state diagram? How are they similar?

similarity： behavioral changes description
difference: sequence: event & object; state: event & state

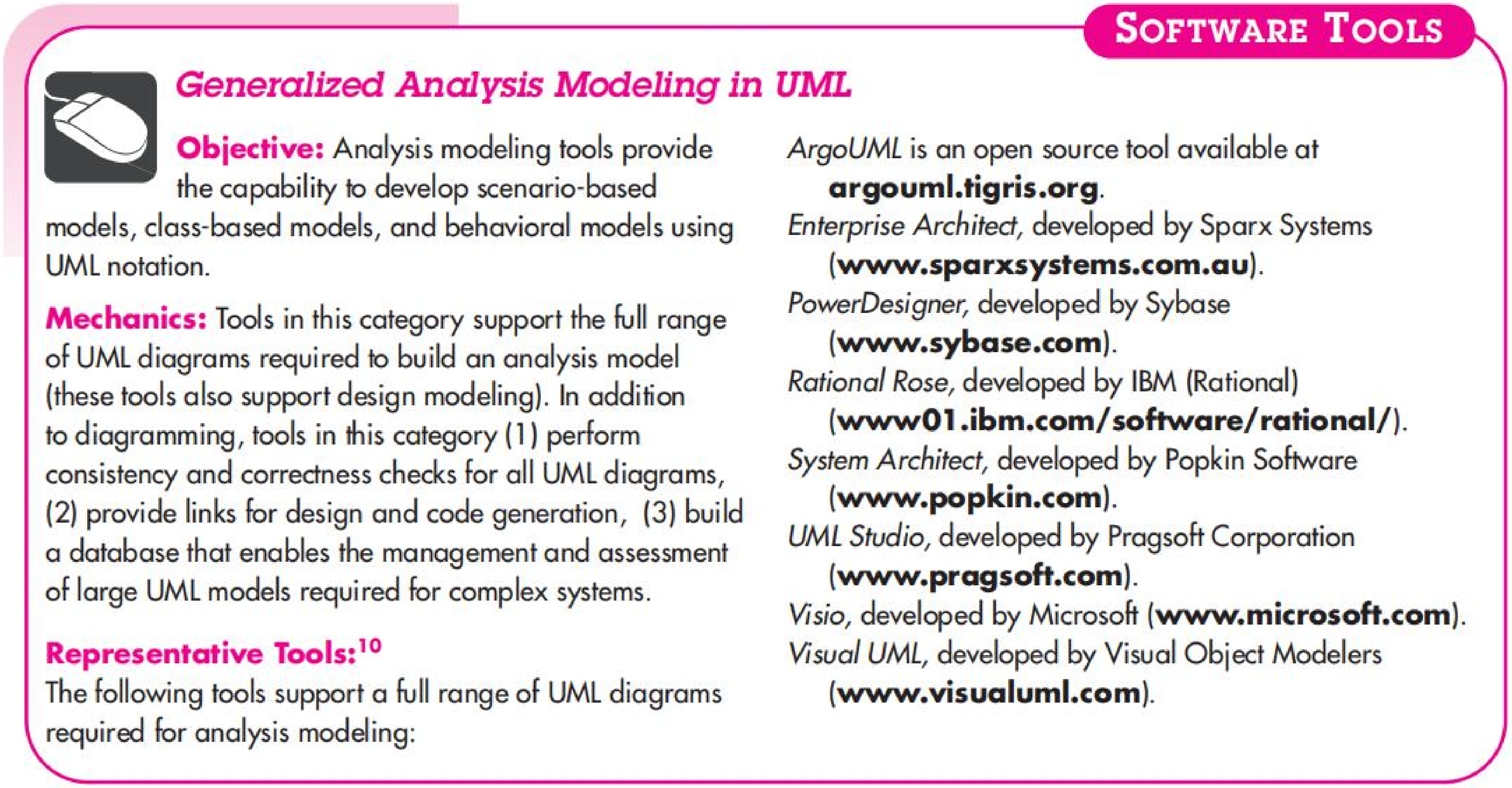
作答

11.3 Behavioral Modeling

How to get behavioral model?

1. Make a list of the different states of a system (How does the system behave?)
2. Indicate how the system makes a transition from one state to another (How does the system change state?)
   - indicate event
   - indicate action
3. draw a state diagram or a sequence diagram

11.3 Behavioral Modeling



SOFTWARE TOOLS

Generalized Analysis Modeling in UML

Objective: Analysis modeling tools provide the capability to develop scenario-based models, class-based models, and behavioral models using UML notation.

Mechanics: Tools in this category support the full range of UML diagrams required to build an analysis model (these tools also support design modeling). In addition to diagramming, tools in this category (1) perform consistency and correctness checks for all UML diagrams, (2) provide links for design and code generation, (3) build a database that enables the management and assessment of large UML models required for complex systems.

Representative Tools:[10]
The following tools support a full range of UML diagrams required for analysis modeling:

ArgoUML is an open source tool available at **argouml.tigris.org**.
Enterprise Architect, developed by Sparx Systems (**www.sparxsystems.com.au**).
PowerDesigner, developed by Sybase (**www.sybase.com**).
Rational Rose, developed by IBM (Rational) (**www01.ibm.com/software/rational/**).
System Architect, developed by Popkin Software (**www.popkin.com**).
UML Studio, developed by Pragsoft Corporation (**www.pragsoft.com**).
Visio, developed by Microsoft (**www.microsoft.com**).
Visual UML, developed by Visual Object Modelers (**www.visualuml.com**).

https://www.kdocs.cn/

https://www.lucidchart.com/

11.4 Patterns for Requirements Modeling

- Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be reapplied when a new problem is encountered
  - domain knowledge can be applied to a new problem within the same application domain
  - the domain knowledge captured by a pattern can be applied by analogy to a completely different application domain.

- Once the pattern has been discovered, it is documented

Pattern

11.4 Discovering Analysis Patterns

- The most basic element in the description of a requirements model is the use case.
- A coherent set of use cases may serve as the basis for discovering one or more analysis patterns.
- A *semantic analysis pattern* (SAP) "is a pattern that describes a small set of coherent use cases that together describe a basic generic application."

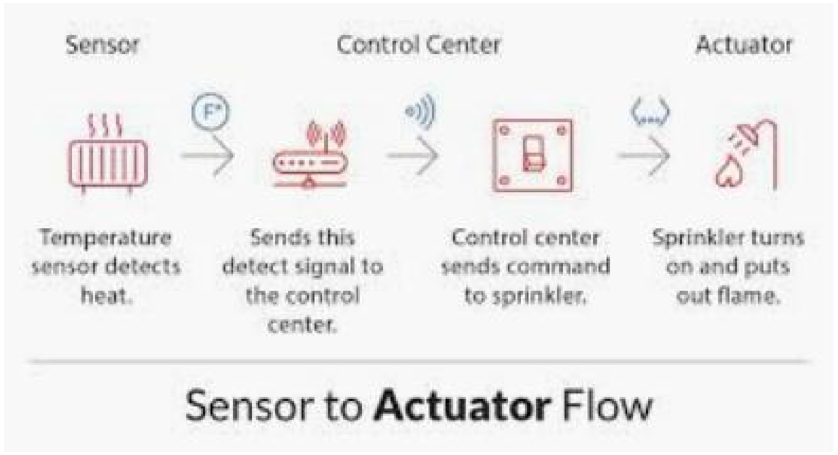
11.4 Analysis Patterns：An Example

Rear view system monitor on dash camera ...

- Consider the following preliminary use case for software required to control and monitor a real-view camera and proximity sensor for an automobile:
  - **Use case:** *Monitor reverse motion*
  - **Description:** When the car is placed in *reverse* gear(倒挡), the control software enables a video feed from a rear-placed video camera to the dashboard display. The control software superimposes a variety of distance and orientation lines on the dashboard display so that the vehicle operator can maintain orientation as the vehicle moves in reverse. The control software also monitors a proximity sensor to determine whether an object is inside 10 feet of the rear of the vehicle. It will automatically break the vehicle if the proximity sensor indicates an object within 3 feet of the rear of the vehicle.

11.4 Analysis Patterns：An Example

- **Motivation :** Embedded systems usually have various kinds of sensors and actuators
- Two main categories of sensors:
  - **PassiveSensors** : *pull* (explicit request for information)
  - **ActiveSensors :** *push* (broadcast of information)


Sensor
Control Center
Actuator
F°
Temperature sensor detects heat.
Sends this detect signal to the control center.
Control center sends command to sprinkler.
Sprinkler turns on and puts out flame.
Sensor to **Actuator** Flow

- Sensors:
  provide information
- Actuator:
  react on sensor's information

11.4 Analysis Patterns： An Example

- This use case implies a variety of functionality that would be refined and during requirements gathering and modeling.

- Regardless of how much elaboration is accomplished, the use case(s) suggest(s) a simple, yet widely applicable SAP—the software-based monitoring and control of sensors and actuators in a physical system.

- But in a more general case, a widely applicable pattern is discovered --> **Actuator - Sensor**

11.4 Actuator-Sensor Pattern—I

Pattern Name: ***Actuator-Sensor***

Intent: Specify various kinds of sensors and actuators in an embedded system.

Motivation: Embedded systems usually have various kinds of sensors and actuators. These sensors and actuators are all either directly or indirectly connected to a control unit. Although many of the sensors and actuators look quite different, their behavior is similar enough to structure them into a pattern. The pattern shows how to specify the sensors and actuators for a system, including attributes and operations.

Constraints:

Each passive sensor must have some method to read sensor input and attributes that represent the sensor value.

Each active sensor must have capabilities to broadcast update messages when its value changes.

Each active sensor should send a *life tick*, a status message issued within a specified time frame, to detect malfunctions.

Each actuator must have some method to invoke the appropriate response determined by the **ComputingComponent**.

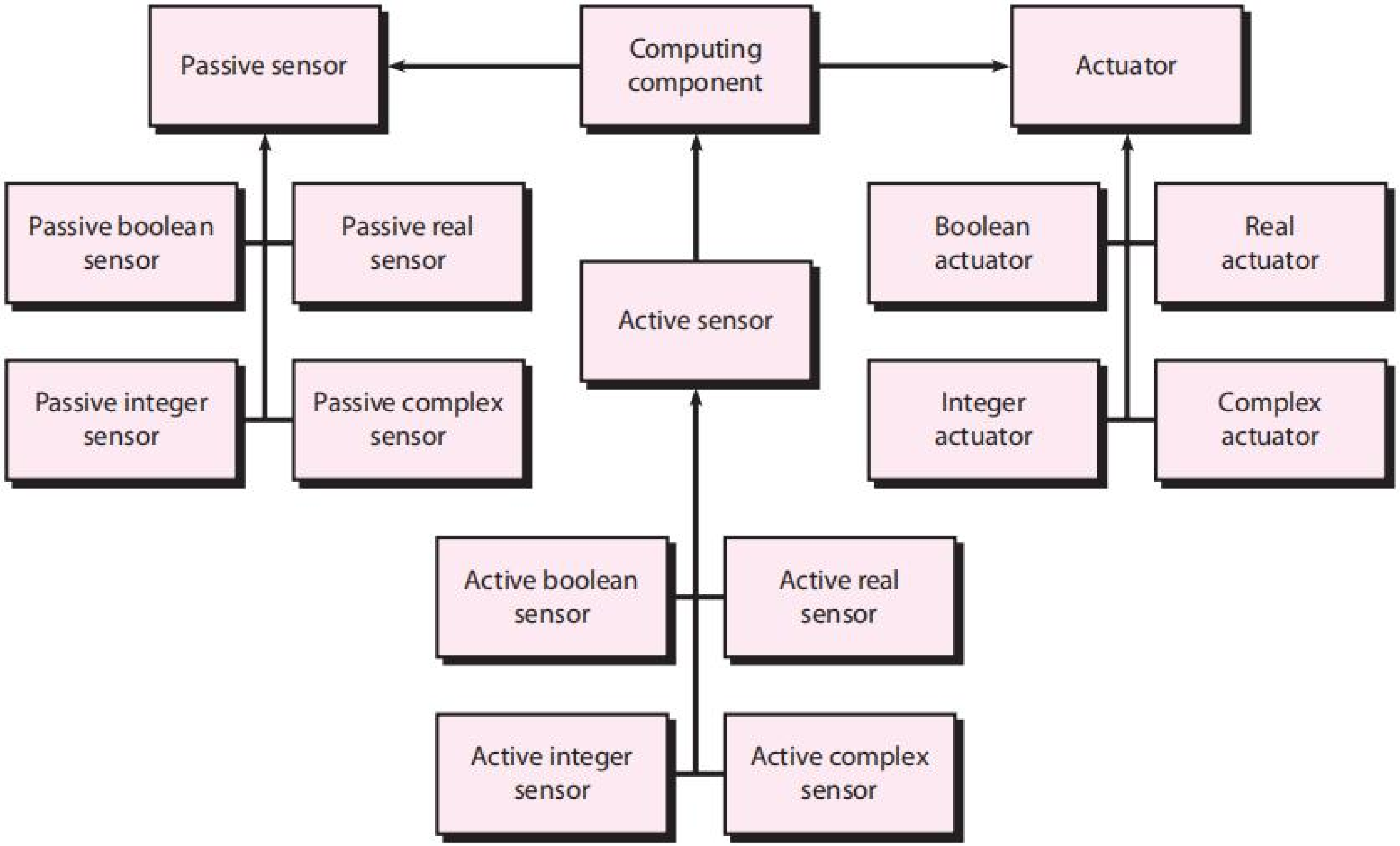
Each sensor and actuator should have a function implemented to check its own operation state.

Each sensor and actuator should be able to test the validity of the values received or sent and set its operation state if the values are outside of the specifications.

11.4 Actuator-Sensor Pattern—II

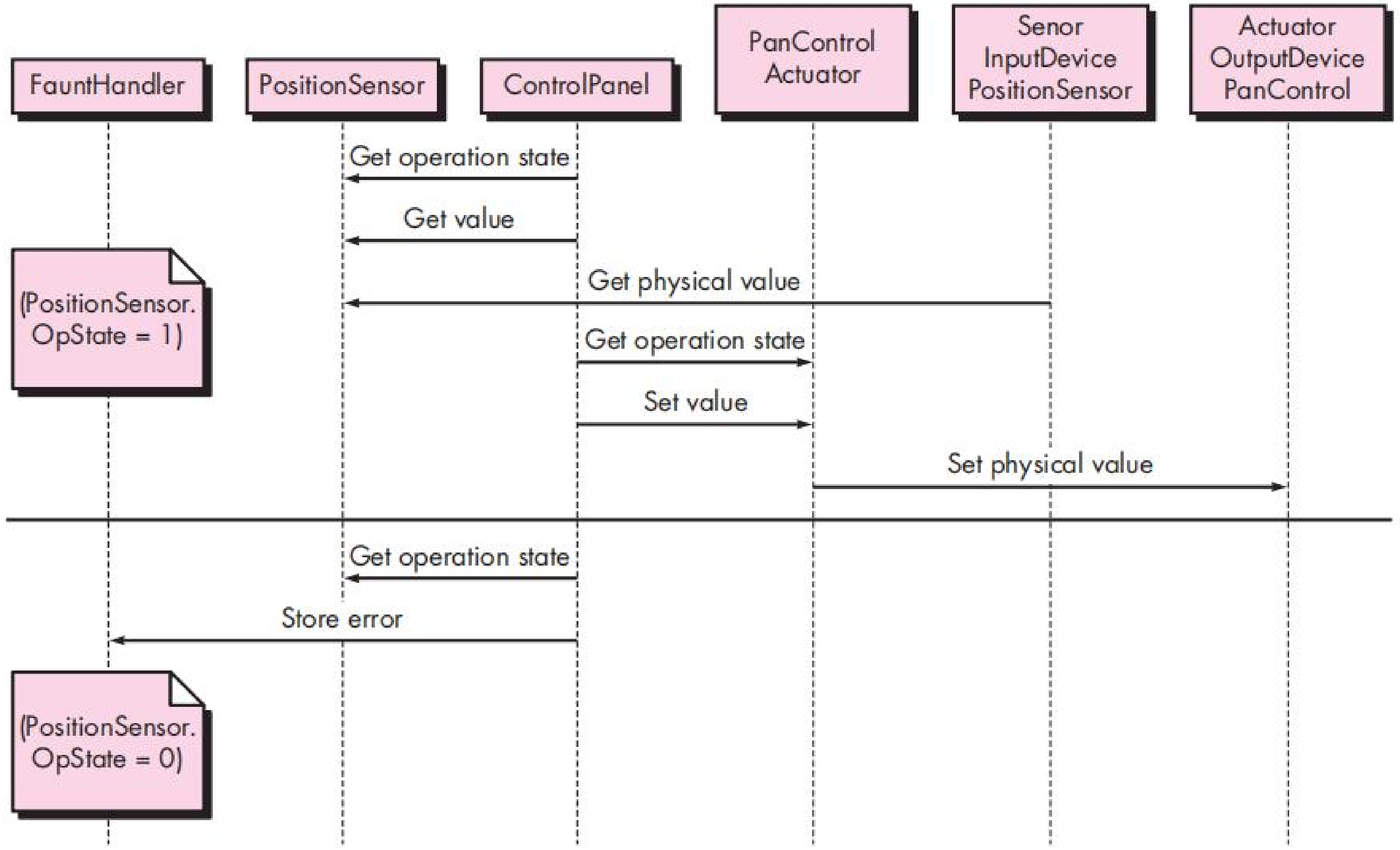
Applicability: Useful in any system in which multiple sensors and actuators are present.

Structure: A UML class diagram



11.4 Actuator-Sensor Pattern—II

Behavior: sequence diagram



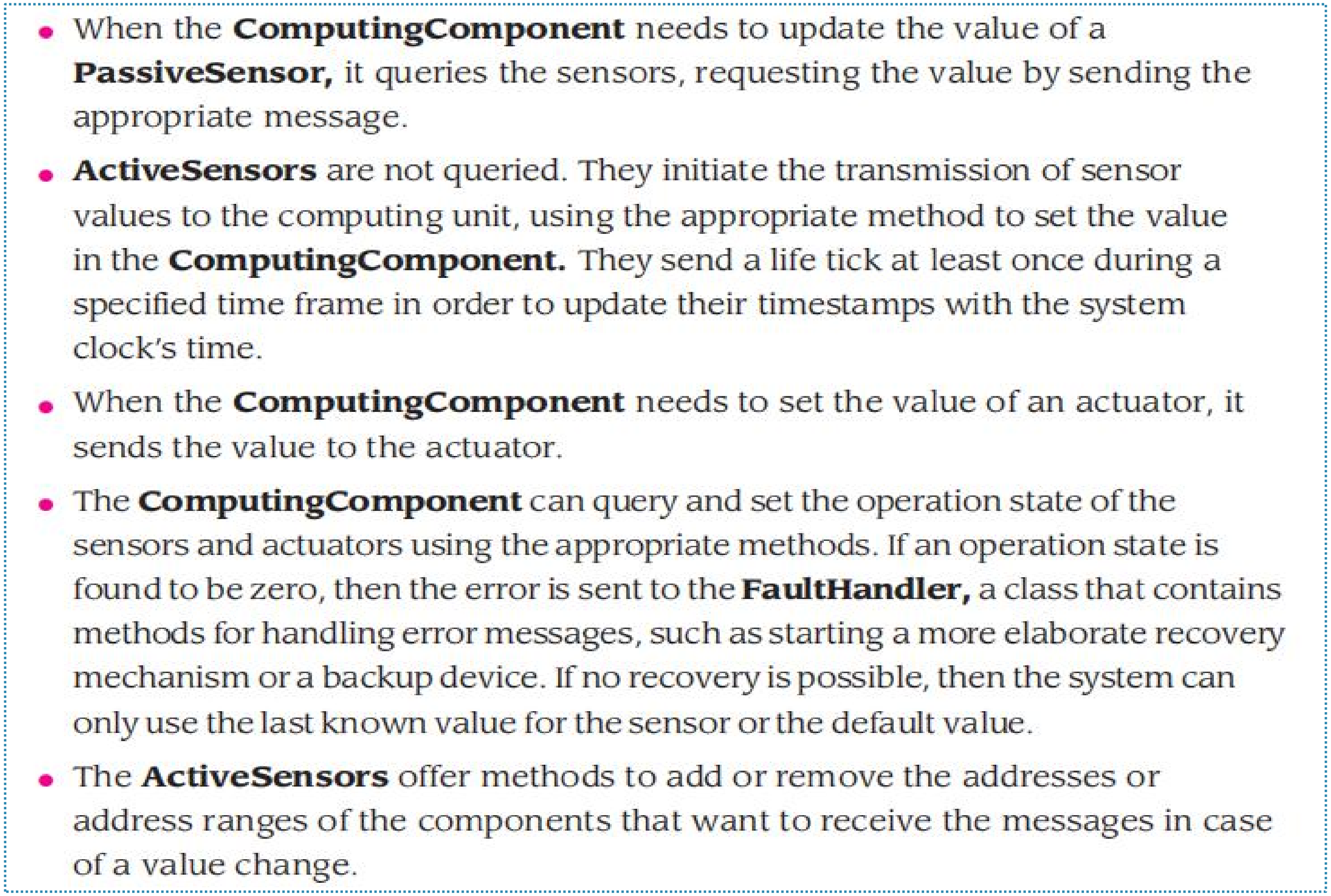
11.4 Actuator-Sensor Pattern—III

Participants:

- **PassiveSensor abstract:** Defines an interface for passive sensors.
- **PassiveBooleanSensor:** Defines passive Boolean sensors.
- **PassiveIntegerSensor:** Defines passive integer sensors.
- **PassiveRealSensor:** Defines passive real sensors.
- **ActiveSensor abstract:** Defines an interface for active sensors.
- **ActiveBooleanSensor:** Defines active Boolean sensors.
- **ActiveIntegerSensor:** Defines active integer sensors.
- **ActiveRealSensor:** Defines active real sensors.
- **Actuator abstract:** Defines an interface for actuators.
- **BooleanActuator:** Defines Boolean actuators.
- **IntegerActuator:** Defines integer actuators.
- **RealActuator:** Defines real actuators.
- **ComputingComponent:** The central part of the controller; it gets the data from the sensors and computes the required response for the actuators.
- **ActiveComplexSensor:** Complex active sensors have the basic functionality of the abstract **ActiveSensor** class, but additional, more elaborate, methods and attributes need to be specified.
- **PassiveComplexSensor:** Complex passive sensors have the basic functionality of the abstract **PassiveSensor** class, but additional, more elaborate, methods and attributes need to be specified.
- **ComplexActuator:** Complex actuators also have the base functionality of the abstract **Actuator** class, but additional, more elaborate methods and attributes need to be specified.

11.4 Actuator-Sensor Pattern—III

Collaborations: how objects and classes interact with one another and how each carries out its responsibilities.

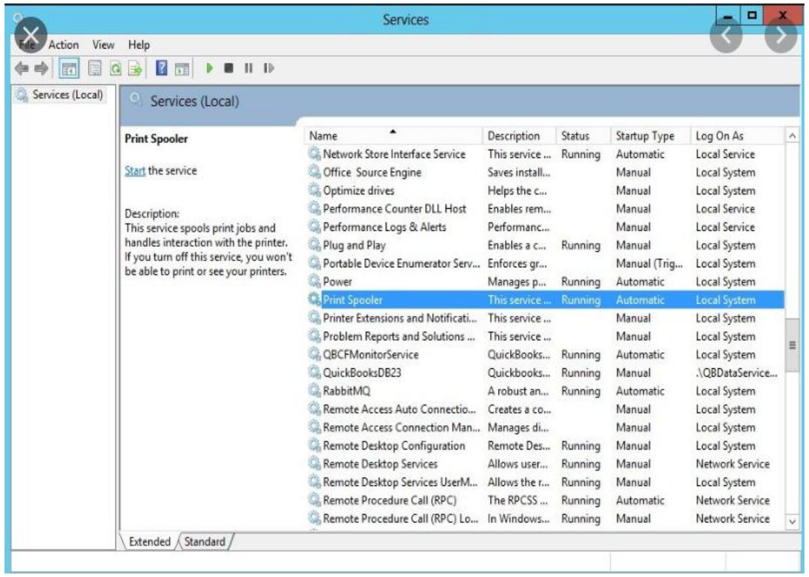


- When the **ComputingComponent** needs to update the value of a **PassiveSensor,** it queries the sensors, requesting the value by sending the appropriate message.
- **ActiveSensors** are not queried. They initiate the transmission of sensor values to the computing unit, using the appropriate method to set the value in the **ComputingComponent.** They send a life tick at least once during a specified time frame in order to update their timestamps with the system clock's time.
- When the **ComputingComponent** needs to set the value of an actuator, it sends the value to the actuator.
- The **ComputingComponent** can query and set the operation state of the sensors and actuators using the appropriate methods. If an operation state is found to be zero, then the error is sent to the **FaultHandler,** a class that contains methods for handling error messages, such as starting a more elaborate recovery mechanism or a backup device. If no recovery is possible, then the system can only use the last known value for the sensor or the default value.
- The **ActiveSensors** offer methods to add or remove the addresses or address ranges of the components that want to receive the messages in case of a value change.

11.4 Actuator-Sensor Pattern—III

Consequences:

1. Sensor and actuator classes have a common interface.
2. Class attributes can only be accessed through messages, and the class decides whether or not to accept the message. For example, if a value of an actuator is set above a maximum value, then the actuator class may not accept the message, or it might use a default maximum value.
3. The complexity of the system is potentially reduced because of the uniformity of interfaces for actuators and sensors.



Print Spooler

Start the service

Description:
This service spools print jobs and handles interaction with the printer. If you turn off this service, you won't be able to print or see your printers.

Name	Description	Status	Startup Type	Log On As
Network Store Interface Service	This service ...	Running	Automatic	Local Service
Office Source Engine	Saves install...		Manual	Local System
Optimize drives	Helps the c...		Manual	Local System
Performance Counter DLL Host	Enables rem...		Manual	Local Service
Performance Logs & Alerts	Performanc...		Manual	Local Service
Plug and Play	Enables a c...	Running	Manual	Local System
Portable Device Enumerator Serv...	Enforces gr...		Manual (Trig...	Local System
Power	Manages p...	Running	Automatic	Local System
Print Spooler	This service ...	Running	Automatic	Local System
Printer Extensions and Notificati...	This service ...		Manual	Local System
Problem Reports and Solutions ...	This service ...		Manual	Local System
QBCFMonitorService	QuickBooks...	Running	Automatic	Local System
QuickBooksDB23	Quickbooks...	Running	Manual	.\QBDataService...
RabbitMQ	A robust an...	Running	Automatic	Local System
Remote Access Auto Connectio...	Creates a co...		Manual	Local System
Remote Access Connection Man...	Manages di...		Manual	Local System
Remote Desktop Configuration	Remote Des...	Running	Manual	Local System
Remote Desktop Services	Allows user...	Running	Manual	Network Service
Remote Desktop Services UserM...	Allows the r...	Running	Manual	Local System
Remote Procedure Call (RPC)	The RPCSS ...	Running	Automatic	Network Service
Remote Procedure Call (RPC) Lo...	In Windows...	Running	Manual	Network Service

$\Rightarrow$ PassiveSensor
(To check whether values changed?)

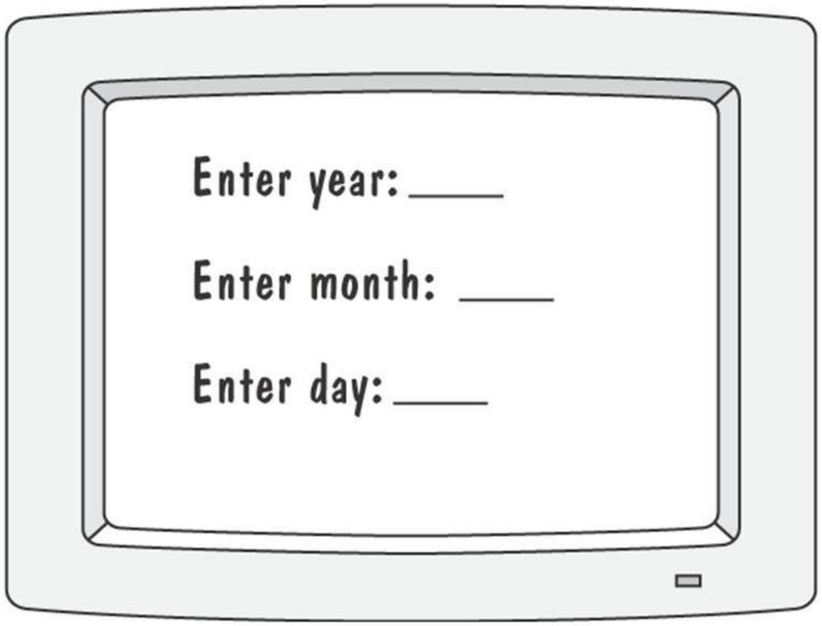
$\Leftarrow$ ActiveSensor
(values change -> report to controller)

Supplement: Prototyping Requirements

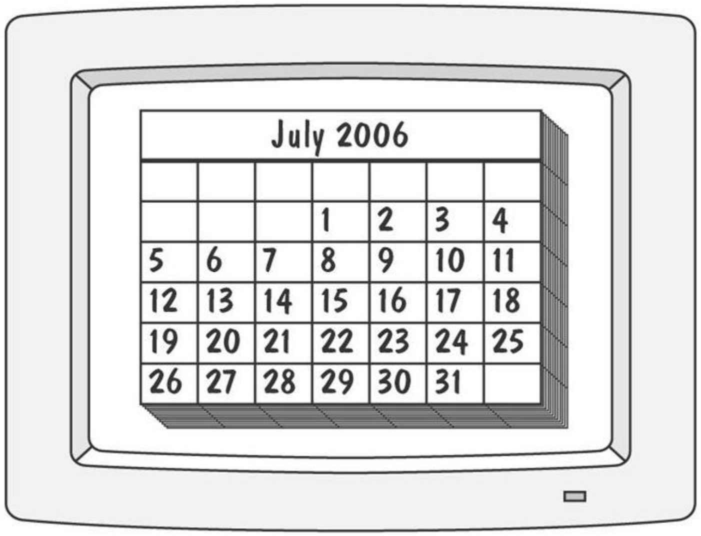
- To elicit the details of proposed system
- To solicit feedback from potential users about
  - what aspects they would like to see improve
  - which features are not so useful
  - what functionality is missing
- Determine whether the customer's problem has a feasible solution
- Assist in exploring options for otimizing quality requirements

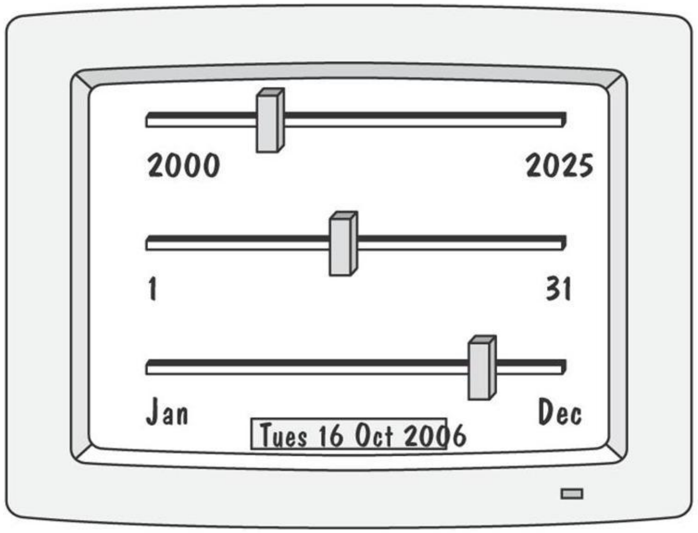
Supplement: Prototyping Requirements

- Prototype for building a tool to track how much a user exercises each day
- Graphical respresentation of first prototype, in which the user must type the day, month and year



Supplement: Prototyping Requirements





Summary

- State diagram (active and passive)
- Sequence diagram
- Analysis Patterns : Actuator-Sensor
- Prototyping

Assignment2 Deadline: 29 Feb

For your planned software, do the following requirement analysis：

1) Function requirement list;
2) Non-function quality requirement list;
3) Usecase diagram;
4) State diagram
5) Activity diagram

Optional diagram:
class diagram, sequence diagram

UML Tool: http://www.umlet.com/ or other tools

THE END