Requirements Modeling: Flow, behavior, and pattern (III)

Behavioral modeling

- State of analysis class
  - Passive vs. active state
    - Examples?
- Elements
  - State
  - State transition
  - Event
  - Action
- Representations
  - State diagram
  - Sequence diagram

The state of a class takes on both passive and active characteristics [CHA93].

A *passive state*is simply the current status of all of an object's attributes.

•E.g.) the class Player: the current position & orientation attributes

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

•E.g.) the class Player: moving, at rest, injured, being cured, trapped, lost

state—a set of observable circumstances that characterizes the behavior of a system at a given time

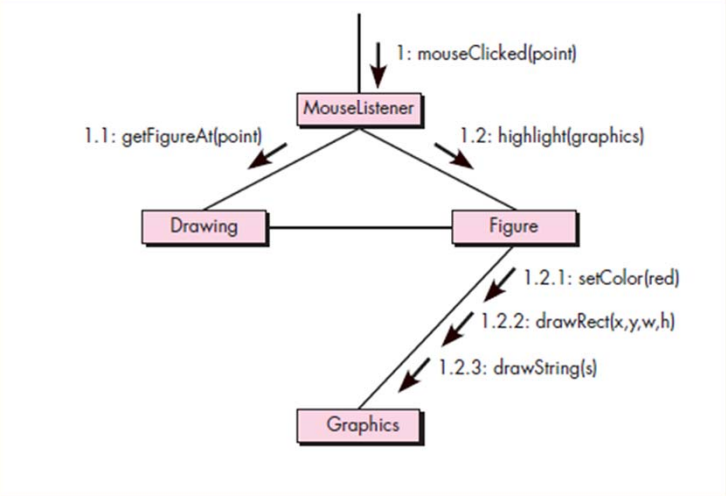
state transition—the movement from one state to another

event—an occurrence that causes the system to exhibit some predictable form of behavior

action—process that occurs as a consequence of making a transition

Communication (collaboration) diagram

- Relationship between objects/classes with temporal order info

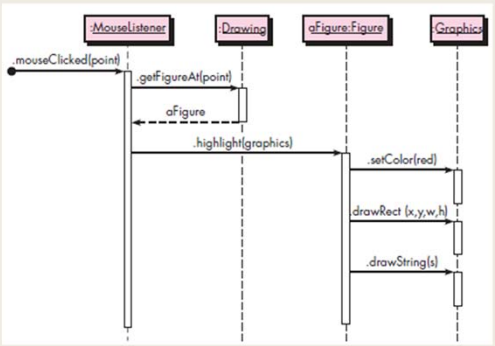


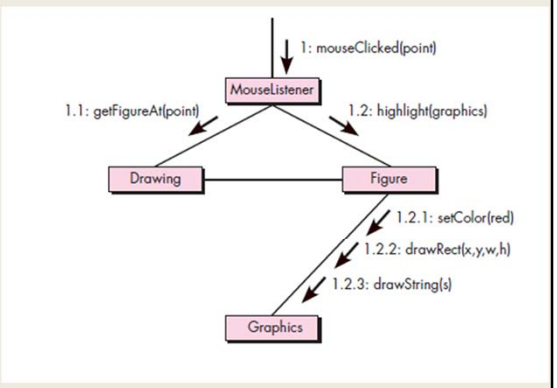
interacting objects are represented by rectangles.
Associations between objects are represented by lines connecting the rectangles. There is typically an incoming arrow to one object in the diagram that starts the sequence of message passing. That arrow is labeled with a number and a message name.

If the incoming message is labeled with the number 1 and if it causes the receiving object to invoke other messages on other objects, then those messages are represented by arrows from the sender to the receiver along an association line and are given numbers 1.1, 1.2, and so forth, in the order they are called

The numbering in each label shows the **nesting** as well as the **sequential** nature of each message.

Sequence vs. communication diagram





- Temporal order of messaging
- Relationship + message exchange

If you are interested in showing the relationships among the objects in addition to the messages being sent between them, the communication diagram is probably a better option than the sequence diagram.

If you are more interested in the temporal order of the message passing, then a sequence diagram is probably better.

Patterns for requirements modeling

- Capture domain knowledge
  - Such that it can be reapplied for new problem
    - In the same domain
    - In a different domain (by analogy)
- Discover -> document -> reuse

-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.

The original author of an analysis pattern does not “create” the pattern, but rather, *discovers* it as
requirements engineering work is being conducted.

Once the pattern has been discovered, it is documented

Discover analysis pattern

- How?
  - Basis: a coherent set of use cases
  - *semantic analysis pattern* (SAP)

Use case: *Monitor reverse motion*
Description: When the vehicle is placed in *reverse* gear, 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 ……. It will automatically break the vehicle if the proximity sensor indicates an object within 3 feet of the rear of the vehicle.

- ***Actuator-Sensor*** pattern
  - Which other software application may reuse this pattern?

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."

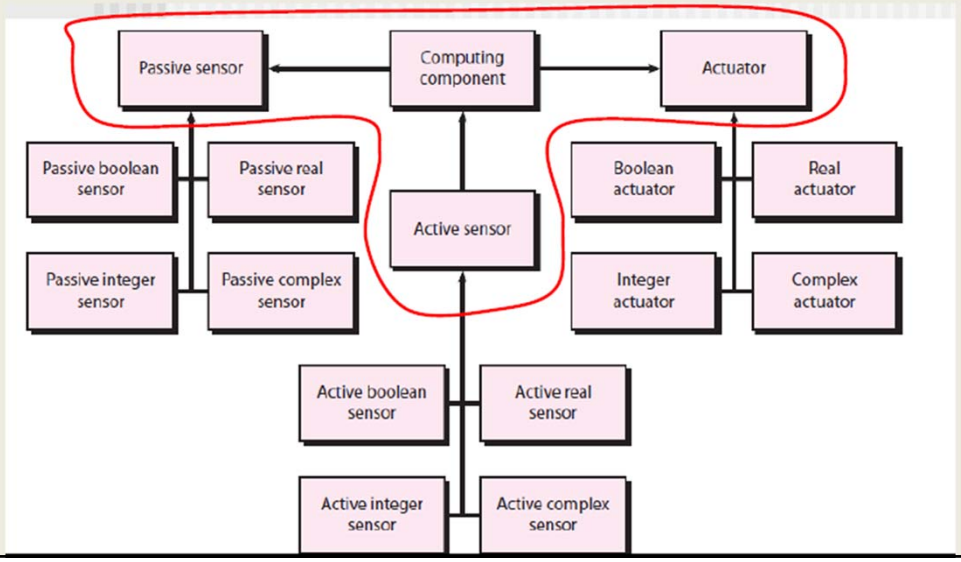
This use case implies a variety of functionality that would be refined and elaborated (into a coherent set of use cases).

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.

the "sensors" provide information about proximity and video information. The "actuator" is the breaking system of the vehicle (invoked if an object is very close to the vehicle. The pattern, called **Actuator-Sensor,** would be applicable as part of the requirements model for *SafeHome*

Discover analysis pattern

- The pattern (in terms of data model)



Applicability: Useful in any system in which multiple sensors and actuators are present.
Structure: A UML class diagram for the *Actuator-Sensor* Pattern is shown in Figure. **Actuator, PassiveSensor** and **ActiveSensor** are abstract classes and denoted in italics. There are four different types of sensors and actuators in this pattern. The Boolean, integer, and real classes represent **the most common types of sensors and actuators**. The complex classes are sensors or actuators that use values that cannot be easily represented in terms of primitive data types, such as a radar device. Nonetheless***, these devices should still inherit the interface from the abstract classes*** since they should have basic functionalities such as querying the operation states.

Discover analysis pattern

- The pattern (in terms of behavioral model)

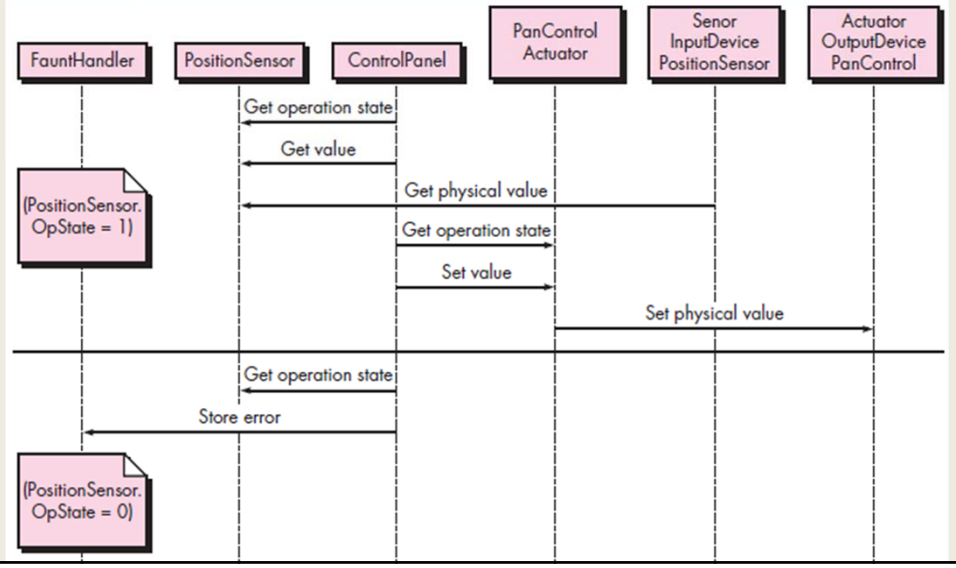


Figure presents a UML sequence diagram for an example of the **Actuator-Sensor** pattern as it might be applied for the *SafeHome* function that controls the positioning (e.g., pan, zoom) of a security camera.

the **ControlPanel (ComputingComponent)** 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.

Discover analysis pattern

- Document the pattern
  - **Pattern Name.**
  - **Intent.**
  - **Motivation.**
  - **Constraints**
  - **Applicability.**
  - **Structure.** (e.g., the data model)
  - **Behavior.** (e.g., the behavior model)
  - **Participants.**
  - **Collaborations**
  - **Consequences.**

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. The *Actuator-Sensor*pattern uses a *pull*mechanism (explicit request for information) for **PassiveSensors**and a *push*mechanism (broadcast of information) for the **ActiveSensors**.

Constraints:
Each passive sensor must have some method to read sensor input and attributesthat represent the sensor value.
Each active sensor must have capabilities to broadcast update messageswhen 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.

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

Structure:A UML class diagram for the *Actuator-Sensor*Pattern is shown in Figure. **Actuator, PassiveSensor**and **ActiveSensor**are abstract classes and denoted in italics. There are four different types of sensors and actuators in this pattern. The Boolean, integer, and real classes represent the most common types of sensors and actuators. The complex classes are sensors or actuators that use values that cannot be easily represented in terms of primitive data types, such as a radar device. Nonetheless, these devices should still inherit the interface from the abstract classes since they should have basic functionalities such as querying the operation states.

Behavior. Here, the **ControlPanel**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.

Summary

- Behavioral modeling
  - Communication diagram
- Patterns in Requirements modeling
  - What is this pattern
  - How do we discover the pattern
  - How do we document it

Behavioral modeling depicts dynamic behavior. The behavioral model uses input from scenario-based and class-based elements to represent the states of analysis classes and the system as a whole.

Analysis patterns enable a software engineer to use existing domain knowledge to facilitate the creation of a requirements model.