Because learning changes everything.®

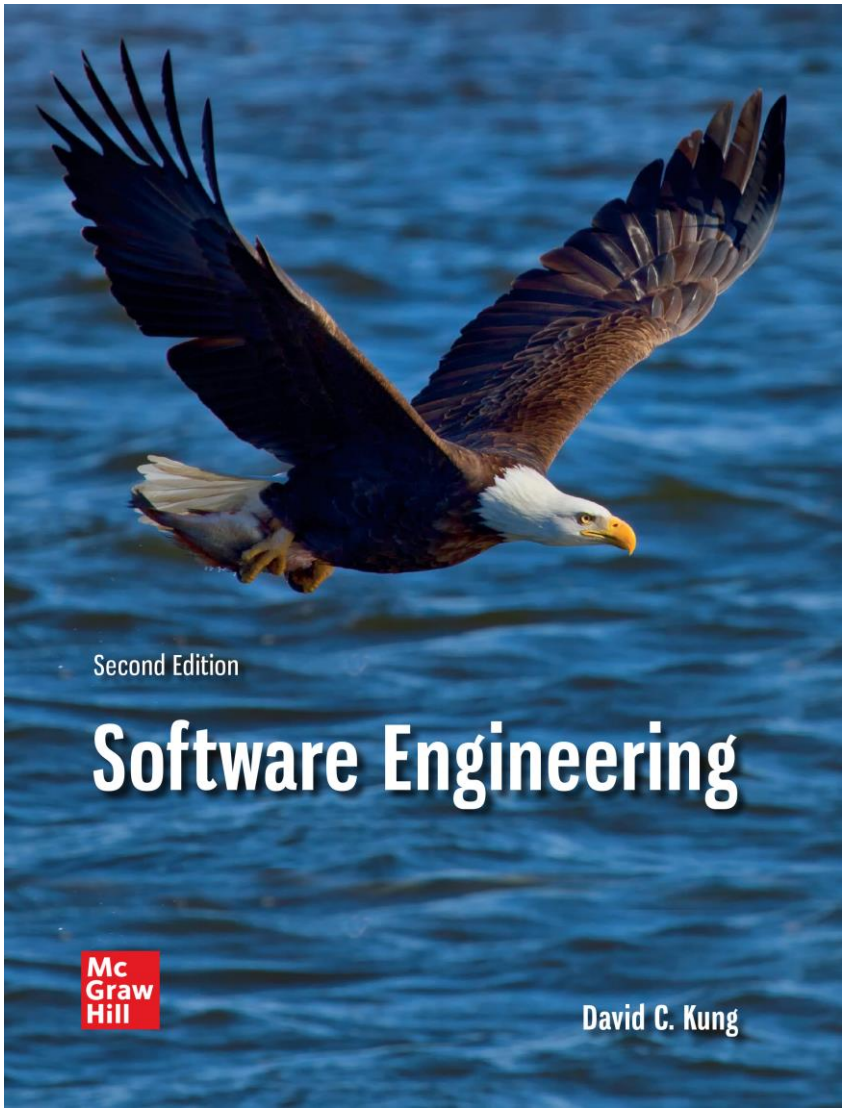
Chapter 13

Modeling and Design of Event-Driven Systems

Software Engineering

Second Edition

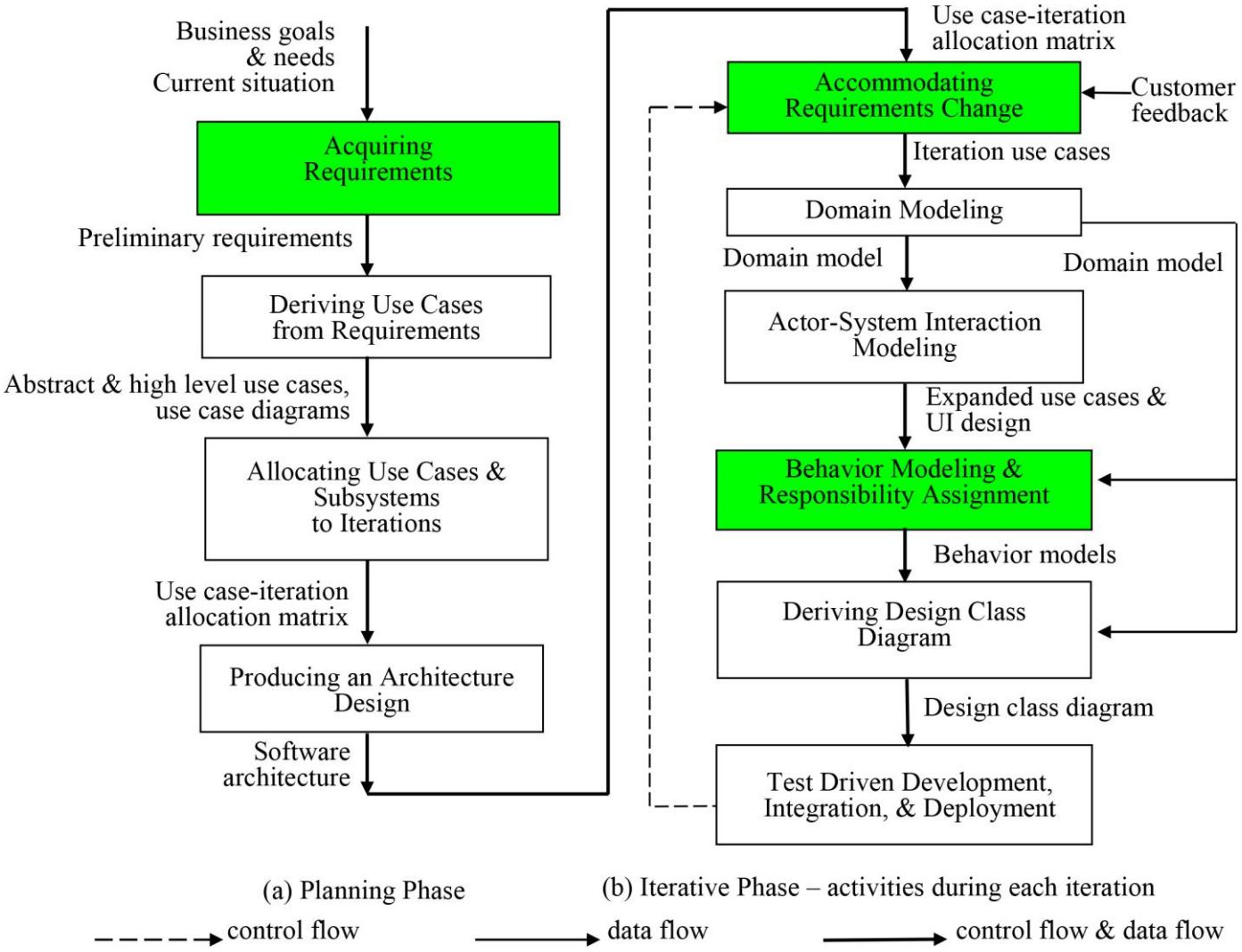
David C. Kung



Key Takeaway Points

- Object state modeling is concerned with the identification, modeling, design, and specification of state-dependent, reactive behavior of objects.
- The state pattern reduces the complexity of state behavior design and implementation, and makes it easy to change.

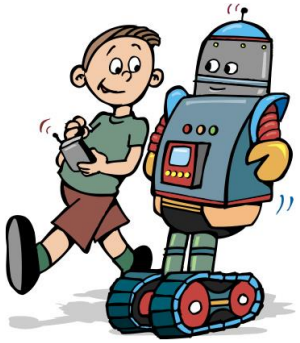
UI Design in the Methodology Context



Access the text alternative for these images

Object State Modeling

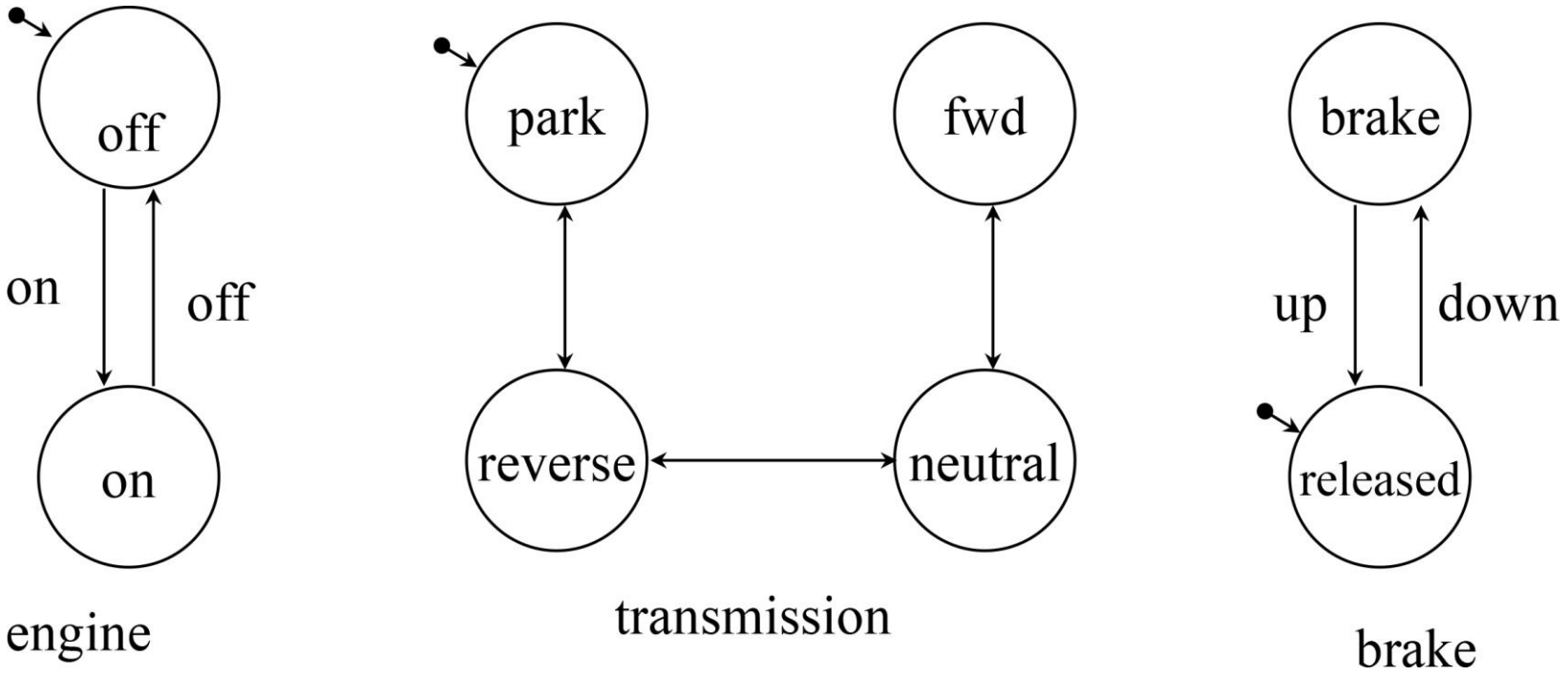
Identification, modeling, analysis, design, and specification of state-dependent reactions of objects to external stimuli.



- What are the external stimuli of interest?
- What are the states of an object?
- How does one characterize the states to determine whether an object is in a certain state?
- How does one identify and represent the states of a complex object?
- How does one identify and specify the state-dependent reactions of an object to external stimuli?
- How does one check for desired properties of a state behavioral model?

State Behavior Modeling

- State dependent behavior is common in software systems.



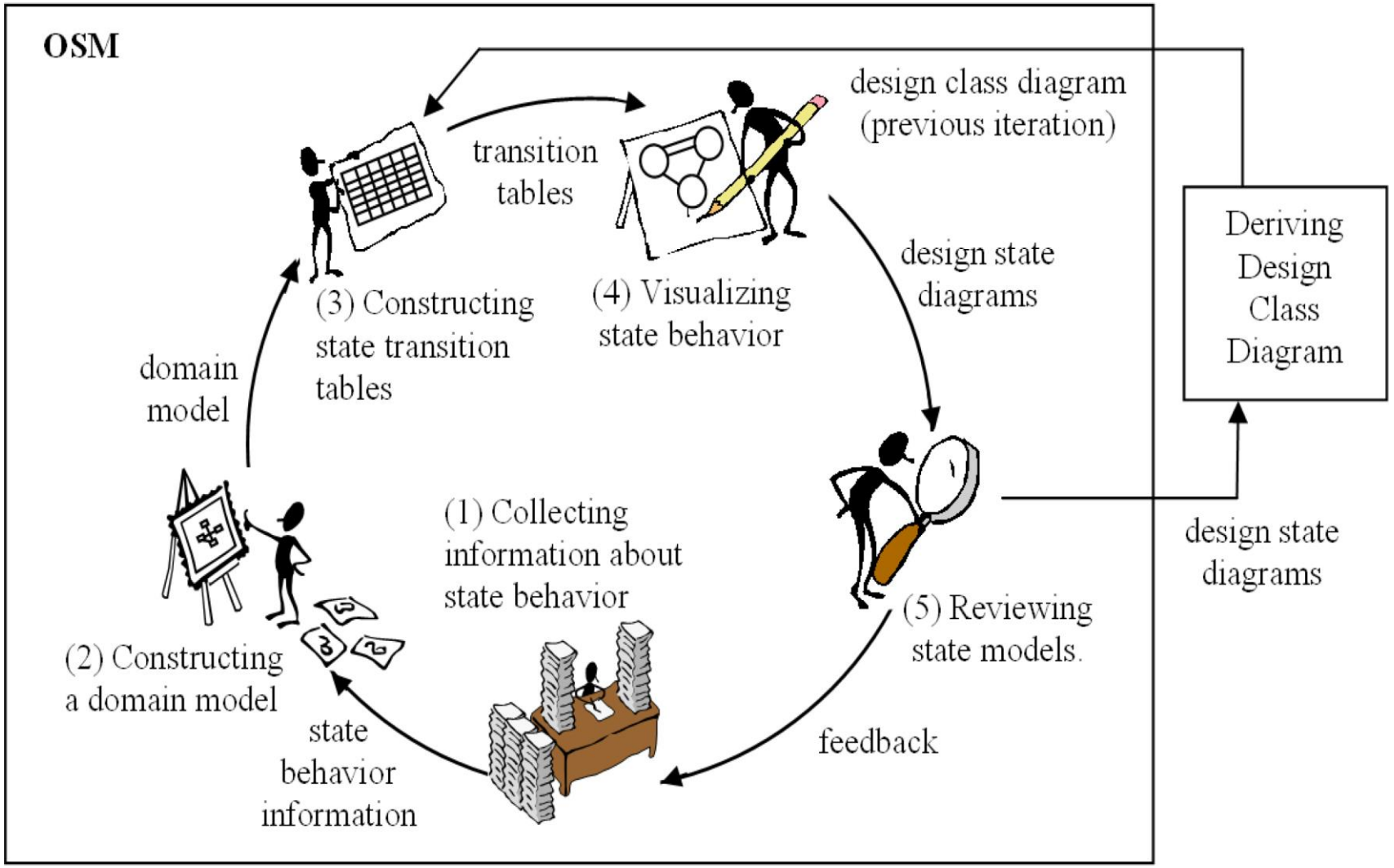
What is the state dependent behavior of a car?

Access the text alternative for these images

Basic Definitions

- An *event* is some happening of interest or a request to a subsystem, object, or component.
- A *state* is a named abstraction of a subsystem/ object condition or situation that is entered or exited due to the occurrence of an event.

Object State Modeling Steps



Access the text alternative for these images

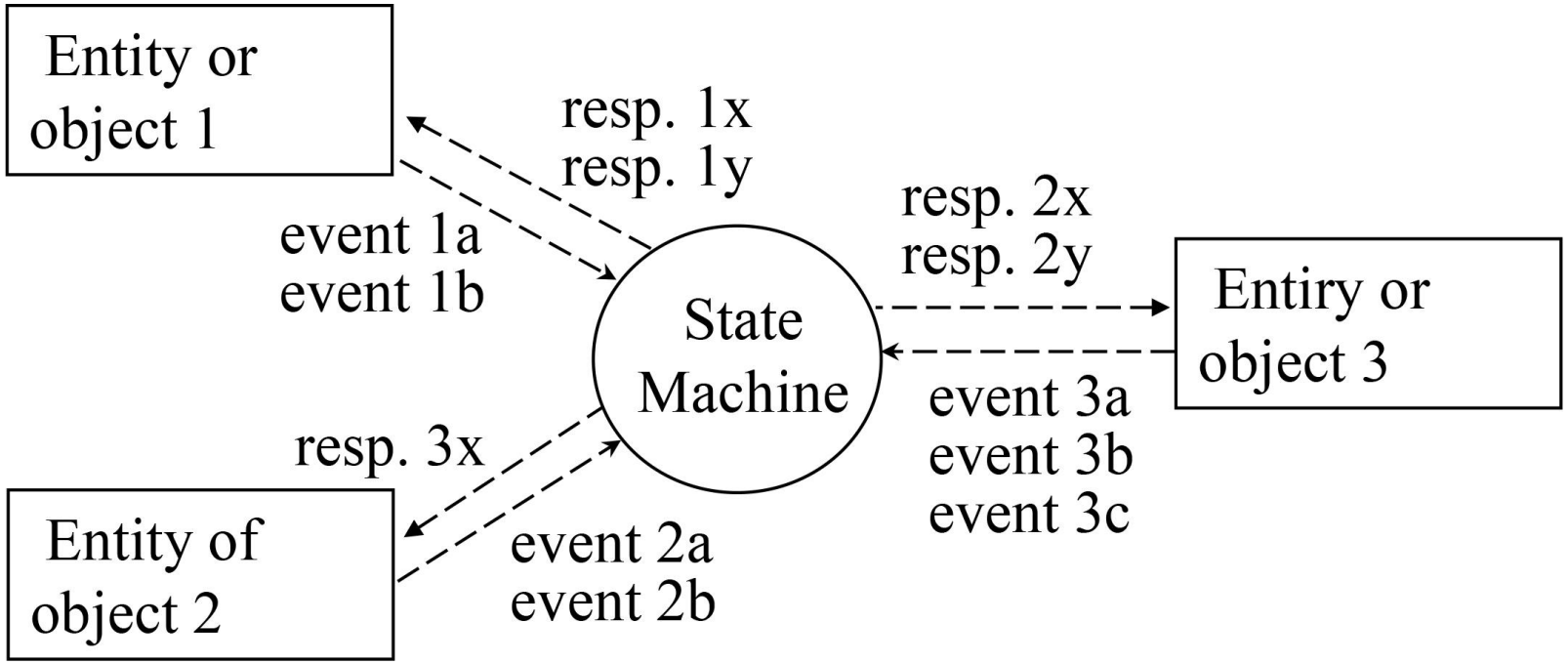
Collecting & Classifying State Behavior Information

What to look for	Example	Classification	Rule
Something interest happened	An online application submitted.	Event	E1
Mode of operation	A cruise control operates in activated/ deactivated modes.	State	S2
Conditions that govern the processing of an Event	Turn on AC if room temperature is high	Guard condition	G1
An act associates w/an event	Push the lever down to set the cruising speed.	Response	R1

More rules are found in the book.

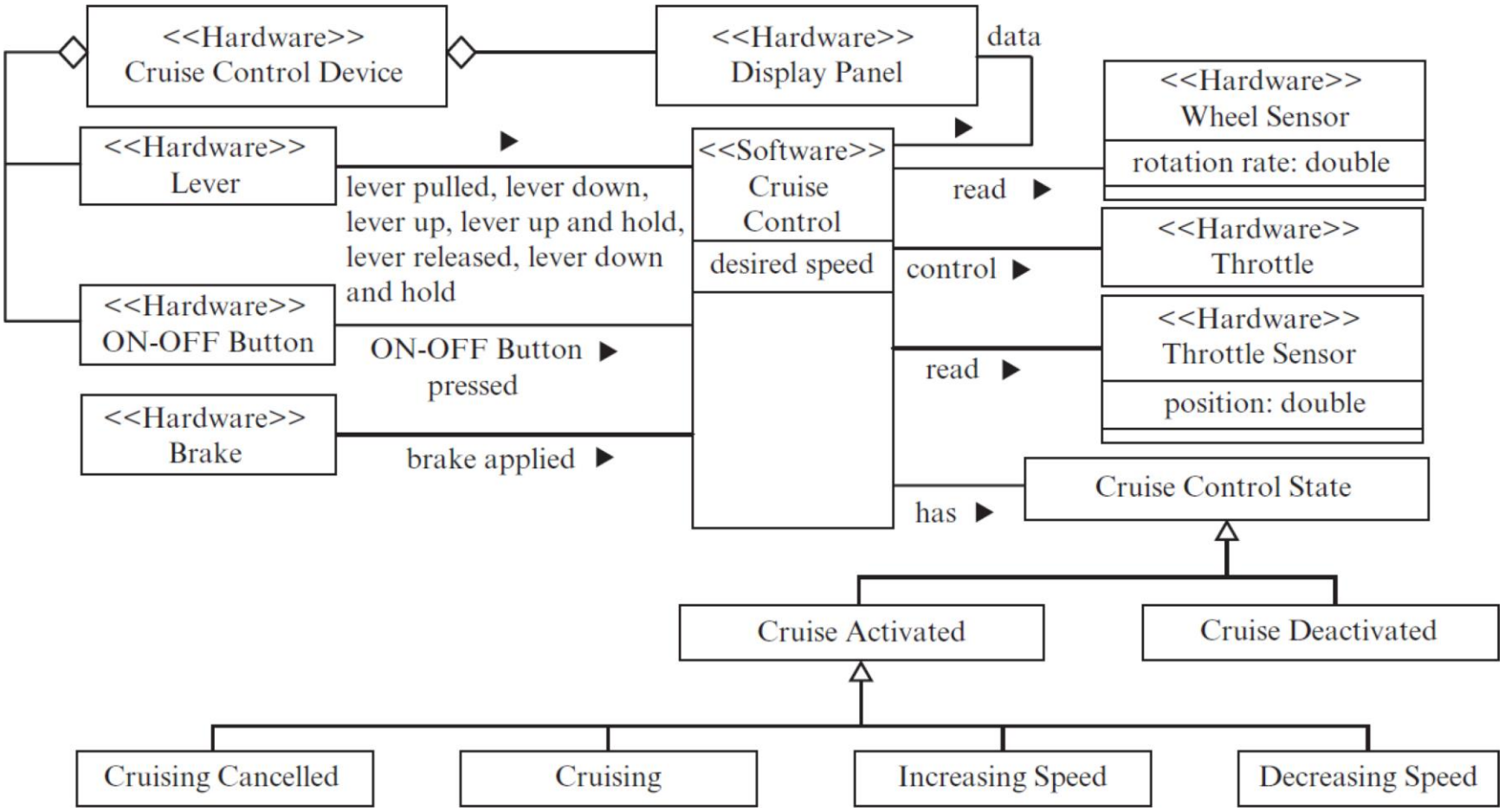
Constructing a Domain Model

- The domain model shows relationships between the state dependent software and its context.



Access the text alternative for these images

Example: Cruise Control Domain Model

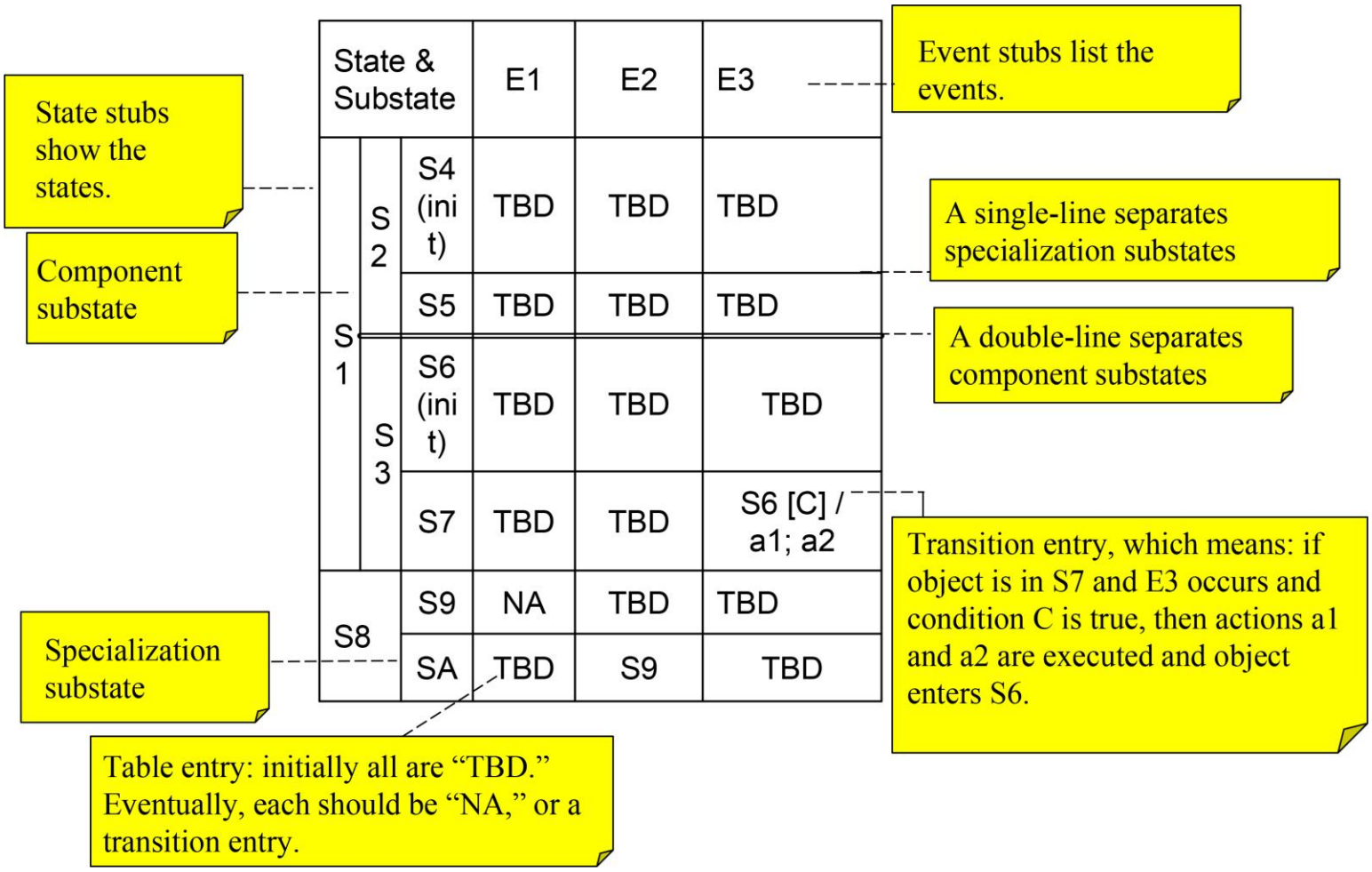


Access the text alternative for these images

Constructing a State Transition Table 1

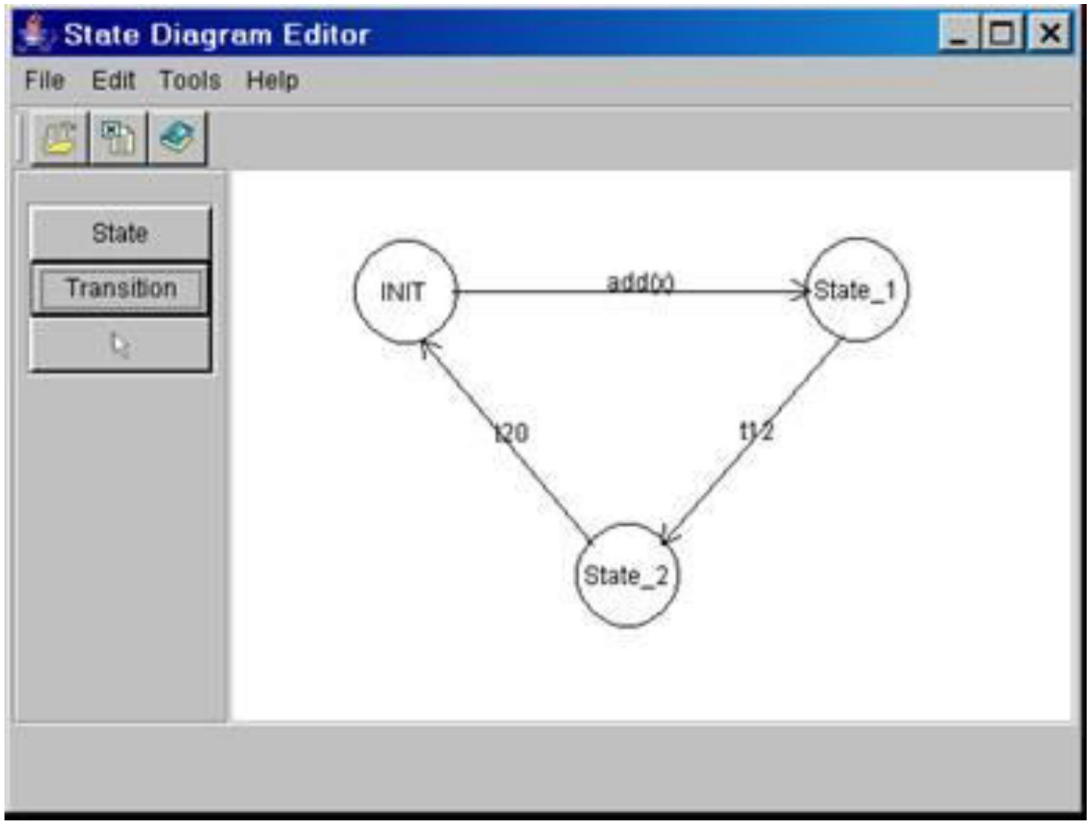
- Constructing the state transition table is optional.
- It is a systematic approach to state behavior modeling.

State Transition Table



Access the text alternative for these images

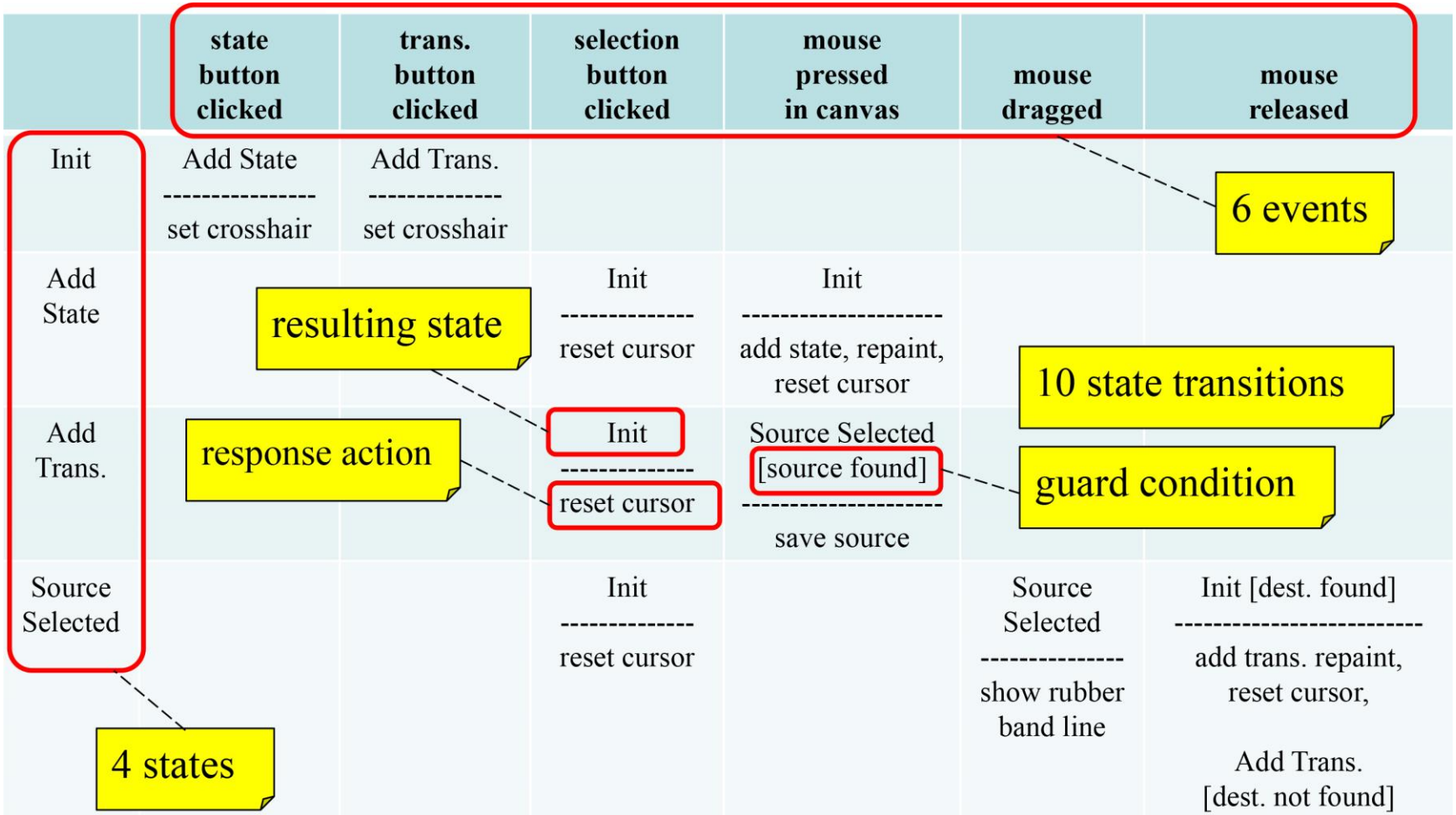
Case Study: State Diagram Editor



Click the state button, and then click any place in the drawing area draws a dummy state. Click the transition button, then press in a state and drag to the destination state and release the mouse draws a dummy transition.

Access the text alternative for these images

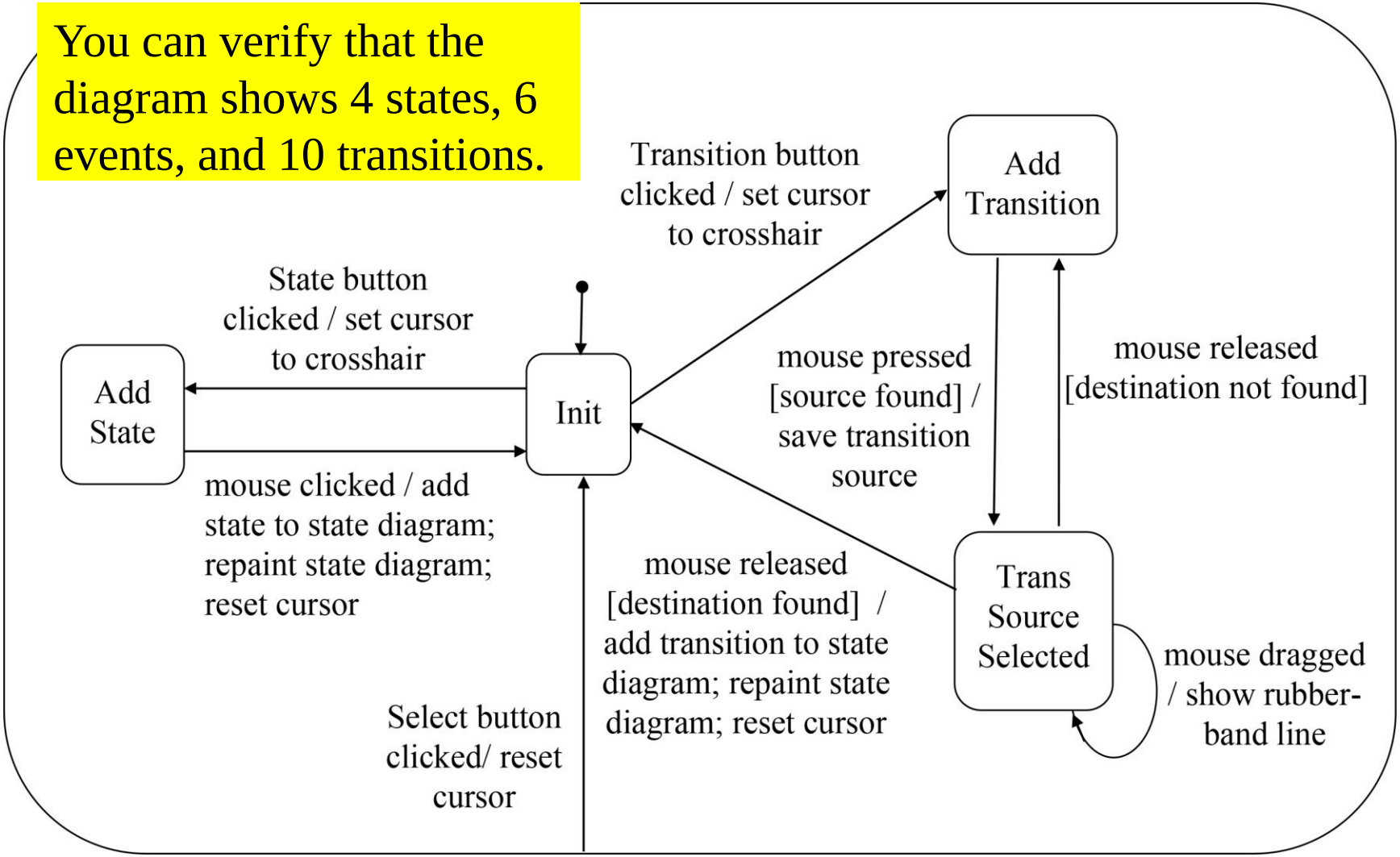
Constructing a State Transition Table 2



Access the text alternative for these images

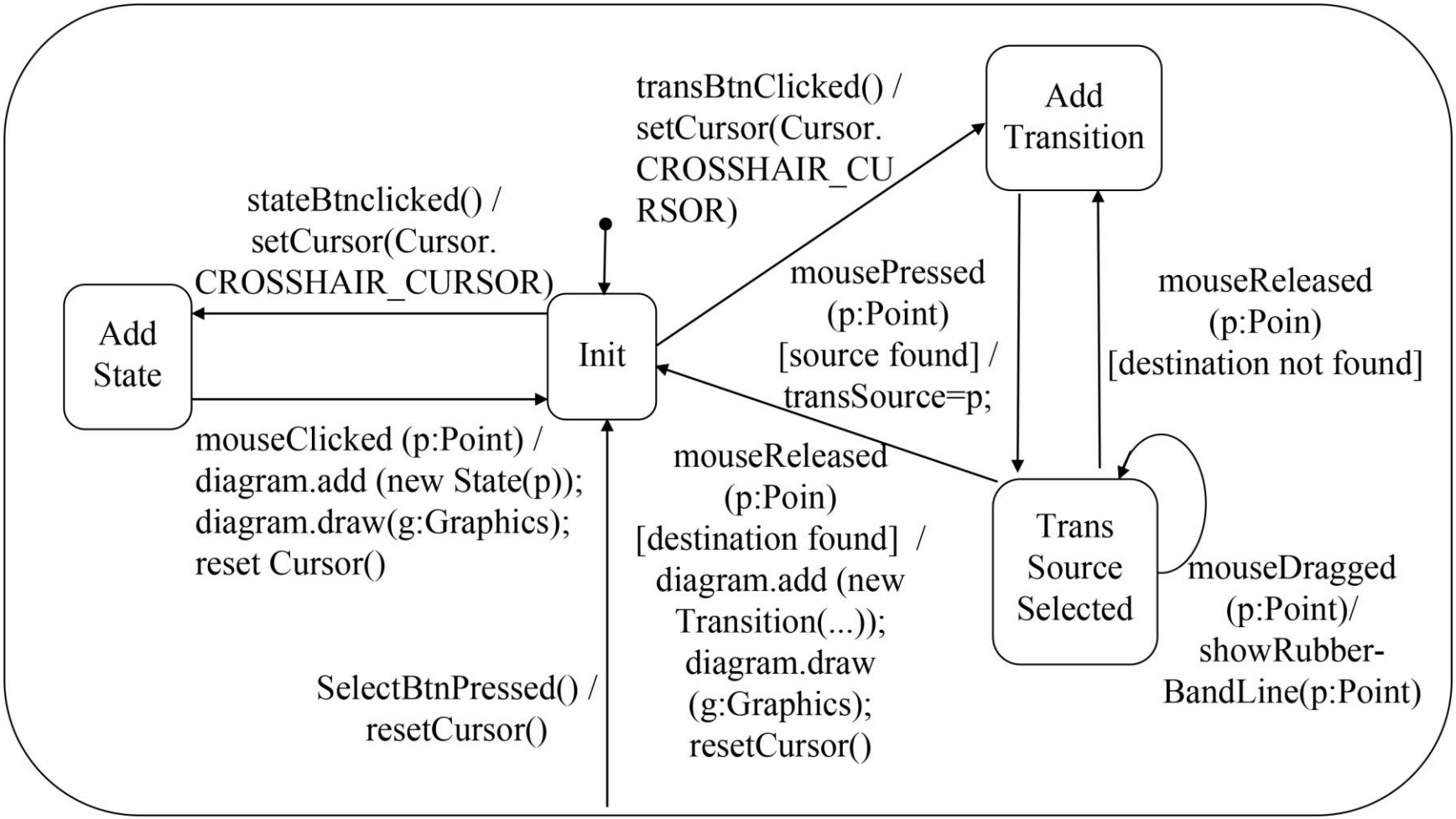
Transition Table to Informal State Diagram

You can verify that the diagram shows 4 states, 6 events, and 10 transitions.



Access the text alternative for these images

Informal SD to Formal/Design State Diagram



Access the text alternative for these images

Usefulness of State Transition Table

The systematic approach ensures internal completeness --- every state-event combination is analyzed.

State diagrams can be generated automatically.

It is easy to detect:

- dead state
- unreachable state
- neglected event
- impossible transitions
- nondeterministic transitions
- redundant transitions
- inconsistent transitions

Implementation: Conventional Approaches

- using nested switch statements
- using a state transition matrix
- using method implementation

Conventional Implementation: Nested Switches

```
switch (STATE) {
  case Init: switch (EVENT) {
                case State button clicked:
                    set cursor to crosshair; STATE=AddState;
                    break;
                case Trans button clicked:
                        set cursor to crosshair;
                        STATE=AddTransition;
                     break;
               }
  case AddState: switch (EVENT) { ... }
  case AddTransition: switch (EVENT) { ... }
  case ...
}
```

Conventional: State Transition Matrix

Event / State	State button clicked	mouse clicked	Trans button clicked	mouse pressed	mouse dragged	mouse released	Select button pressed
Init	set cursor to crosshair/ Add State		set cursor to crosshair/ Add Transition				
Add State		add state to state diagram; repaint state diagram; reset cursor/ Init					reset cursor/ Init
Add Transition				[source found] save transition source/ Trans Source Selected			reset cursor/ Init
Trans Source Selected					show rubber-band line/ Trans Source Selected	add transition to state diagram; repaint state diagram; reset cursor/ Init	reset cursor/ Init

Access the text alternative for these images

Using Method Implementation

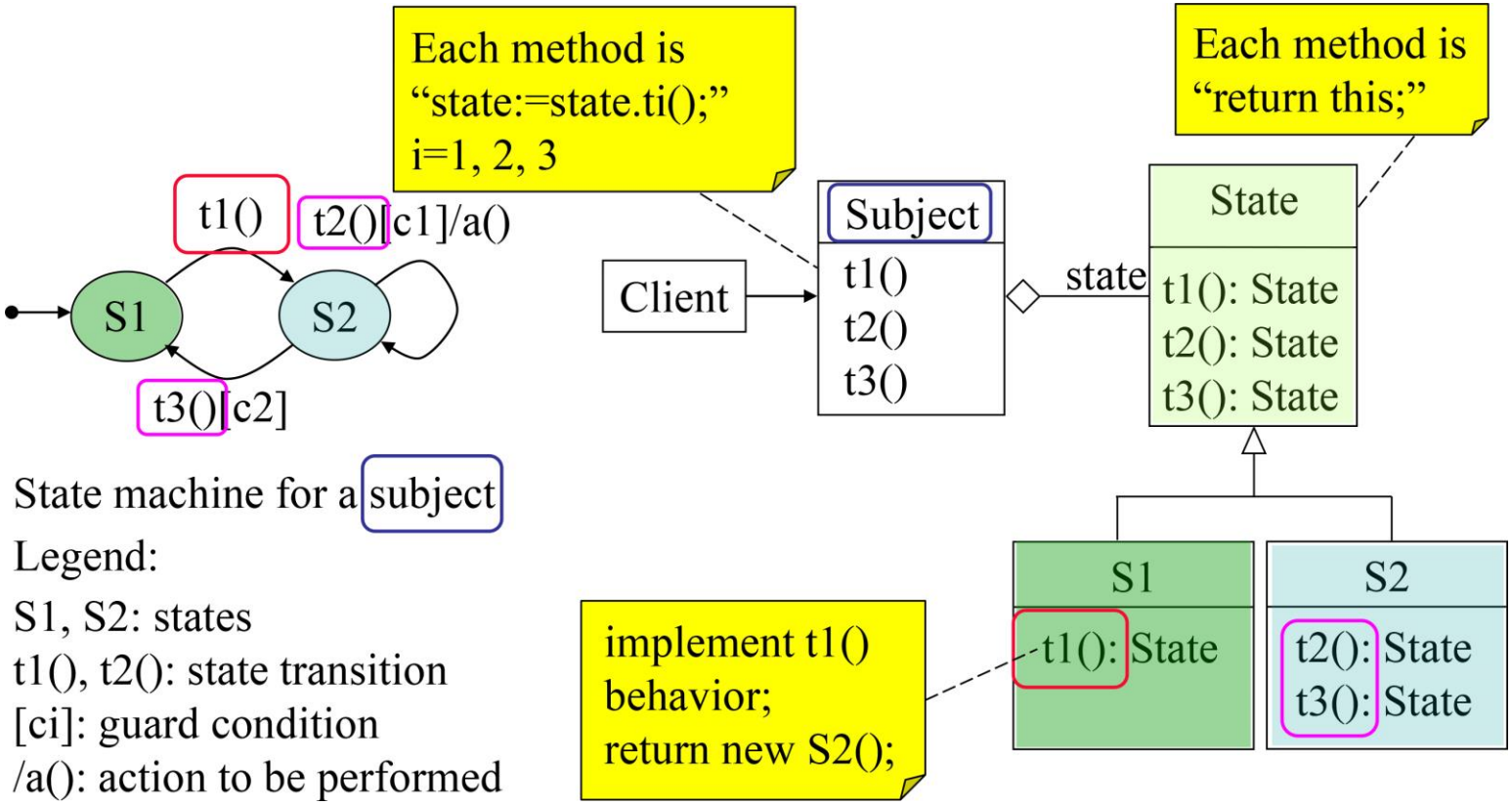
- State behavior of a class is implemented by member functions that denote an event in the state diagram.
- The current state of the object is stored in an attribute of the class.
- A member function evaluates the state variable and the guard condition, executes the appropriate response actions, and updates the state variable.

Problems with Conventional Approaches

- High cyclomatic complexity (number of states multiplied by number of transitions).
- Nested switch statements make the code difficult to comprehend, test, and maintain.
- Difficult to add states (need to change every event case).
- Difficult to add events (need to change every state case).

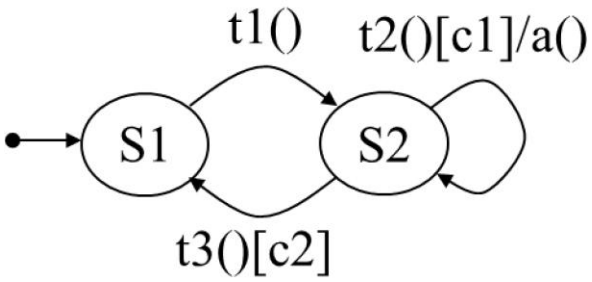
Applying State Pattern 1

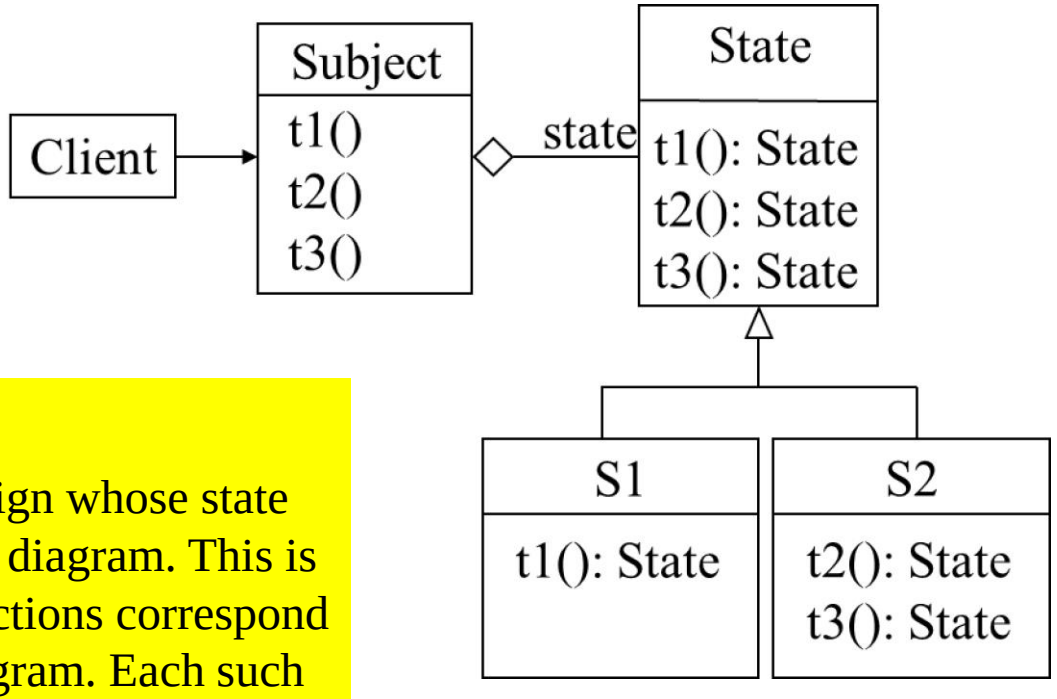
- The state pattern is a low complexity and easy to extend approach to implement a state machine.



Access the text alternative for these images

Applying State Pattern 2



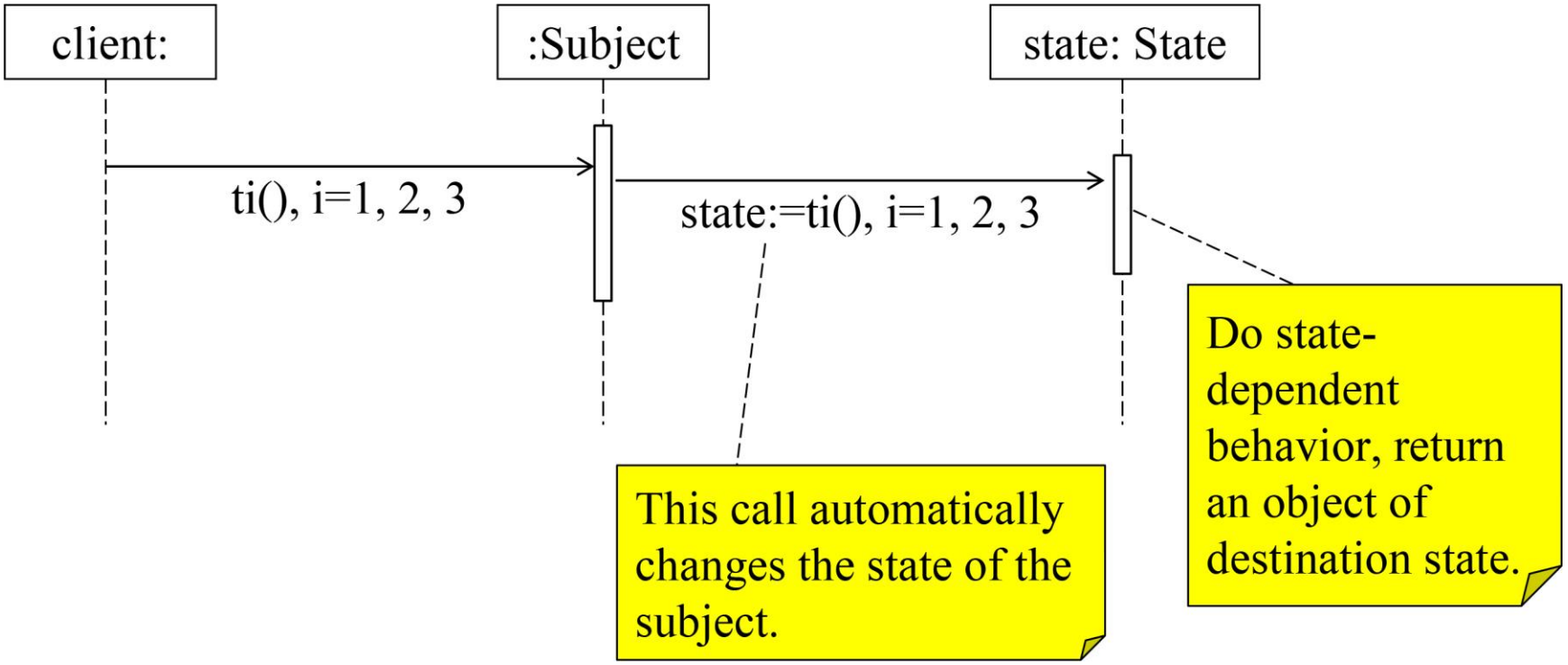


From state diagram to state pattern:

1. Identify the class in existing design whose state behavior is modeled by the state diagram. This is the Subject class, which has functions correspond to the transitions in the state diagram. Each such function f(...) is simply "state:=state.f(...);"
2. Define a State parent class with functions corresponding to transitions in state diagram. Each such function simply "return this."
3. For each state in state diagram, define a State subclass, which overrides the functions that correspond to the transitions from the state. Each such function returns an object of the destination state.

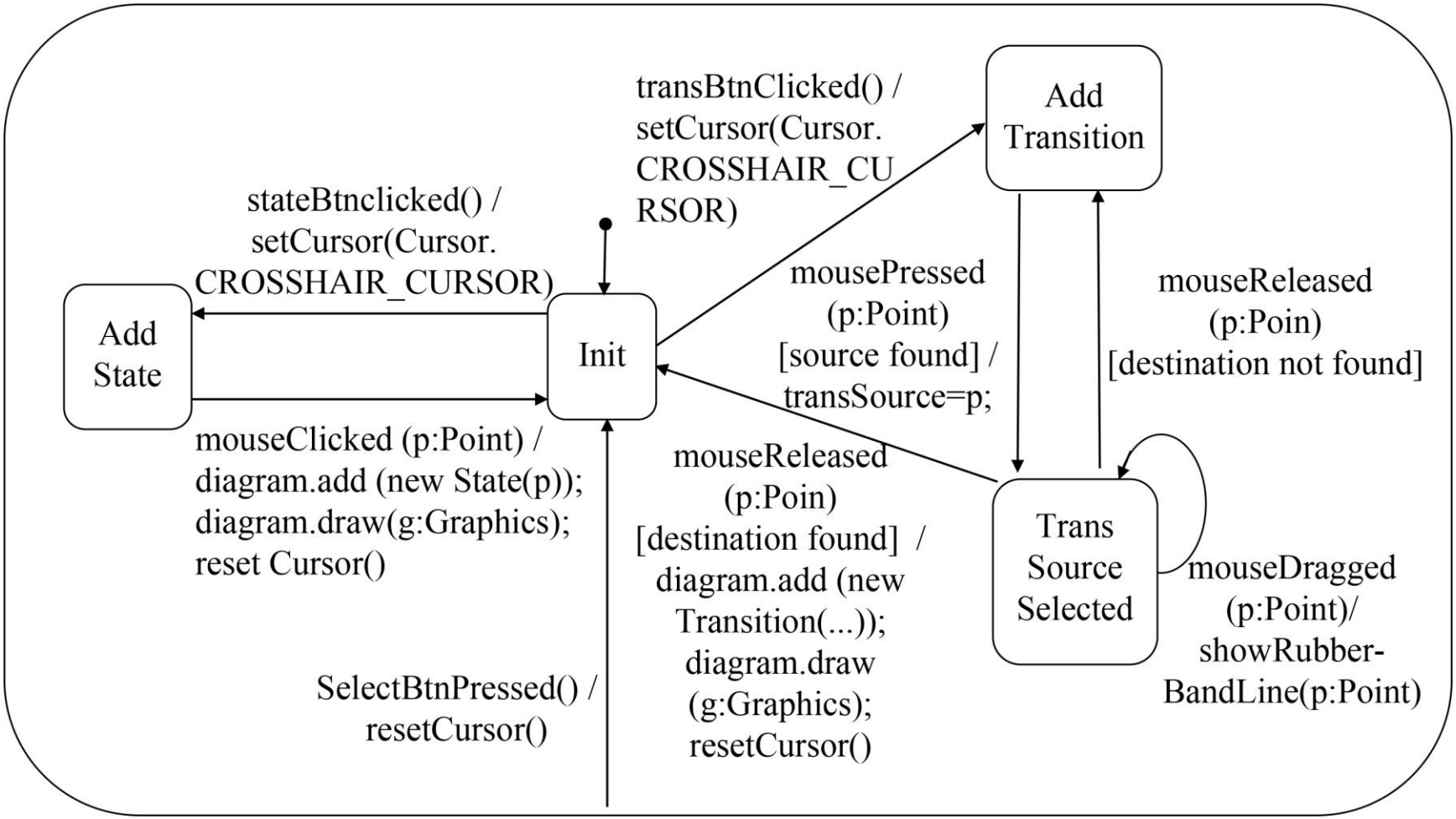
Access the text alternative for these images

Object Interaction in State Pattern



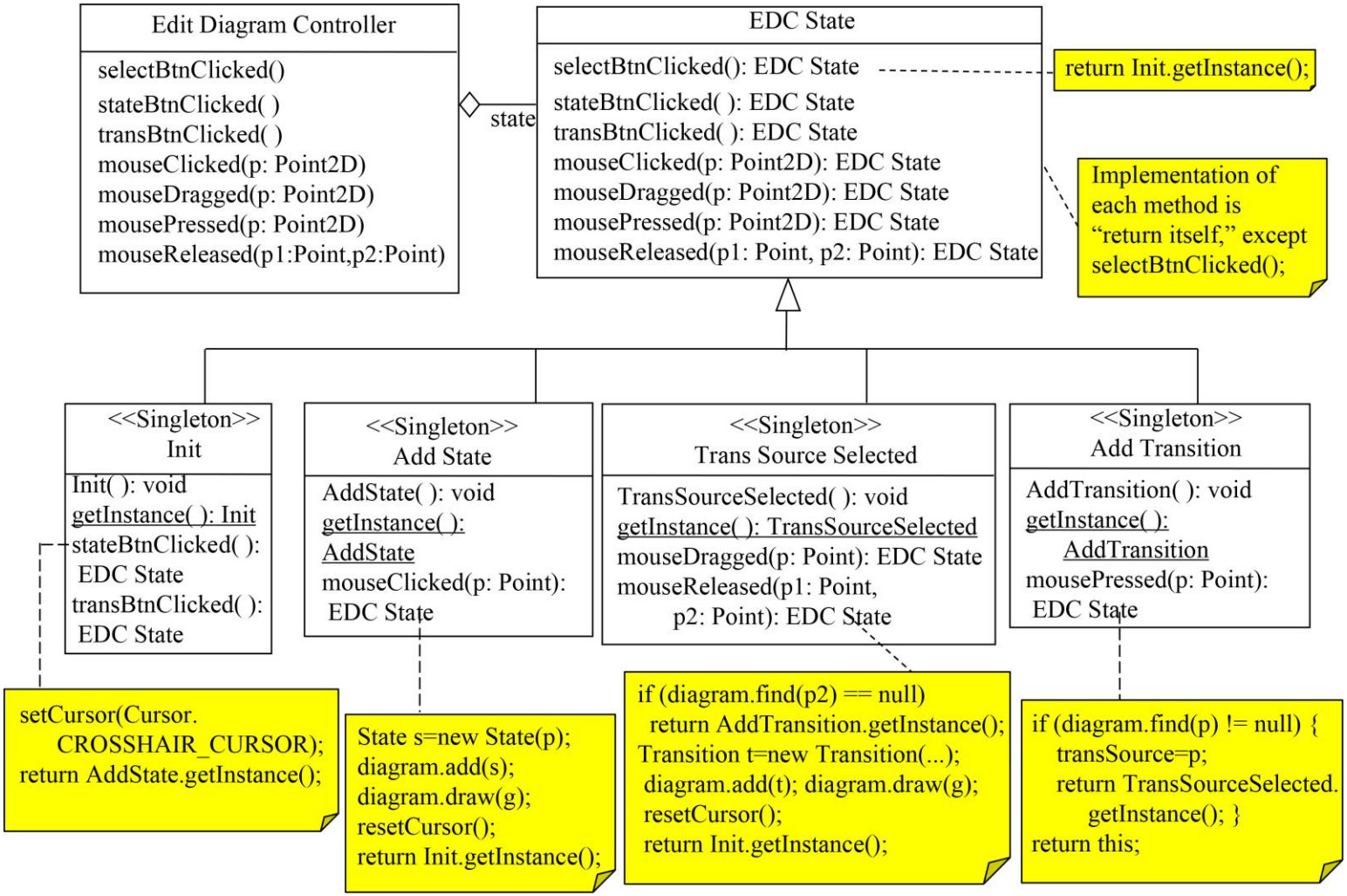
Access the text alternative for these images

Design State Diagram for the Editor 1



Access the text alternative for these images

Design State Diagram for the Editor 2



Access the text alternative for these images

Advance UML State Diagram

The editor state diagram is a flat state diagram, called Mealy-type state diagram. Its states do not contain other states or state diagrams.

A state of a UML state diagram may be a state diagram. Such state diagrams were invented by David Harrel (1988). It is characterized as:

- hierarchical
- concurrent, and
- communicating

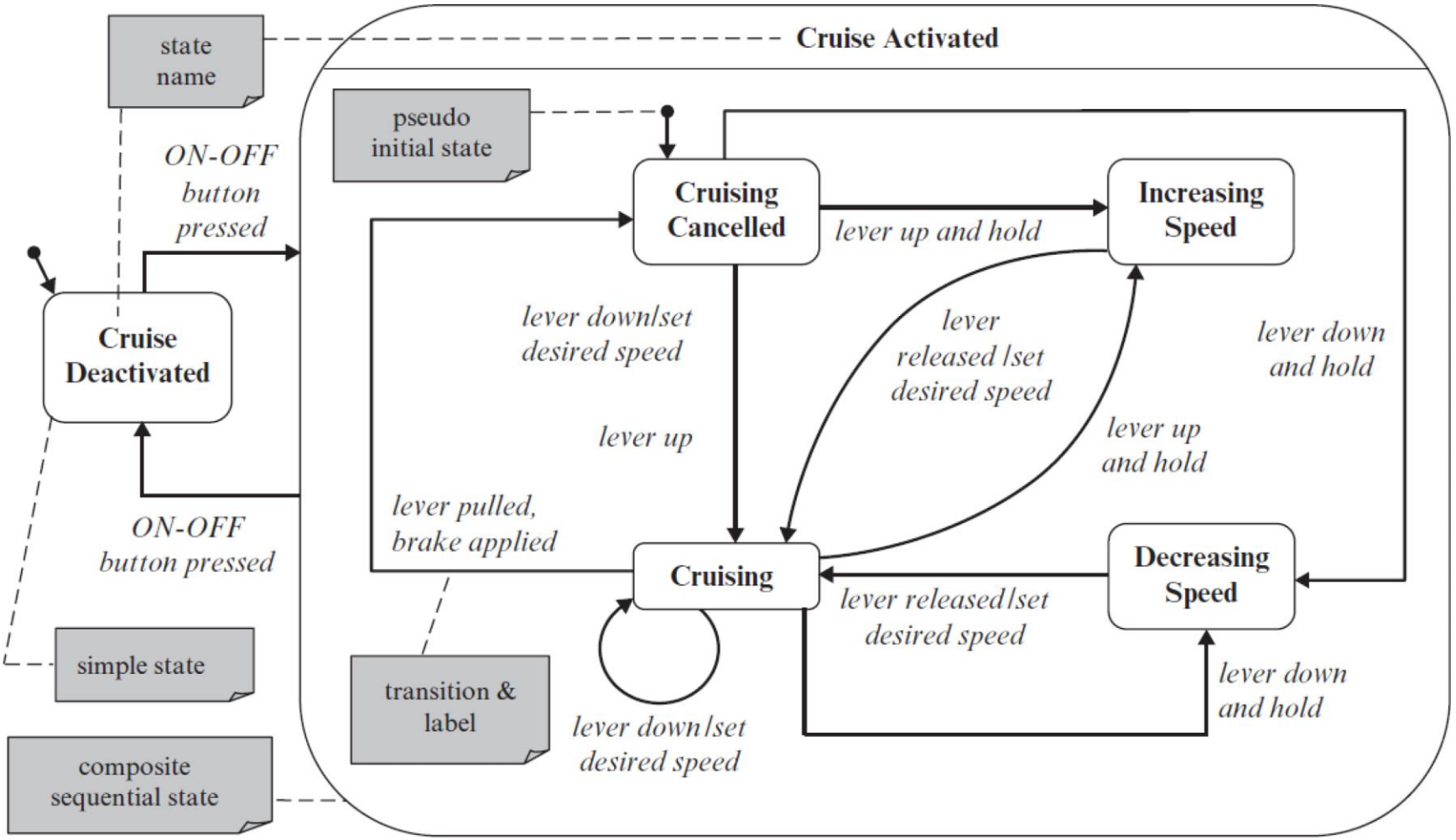
Cruise Control State Transition Table

State & Substate \ Event		Lever down	Lever up and hold	Lever down and hold	Lever released	Lever pulled	Brake applied	Lever up	ON-OFF button pressed
Cruise deactivated (init)		NA	NA	NA	NA	NA	NA	NA	Cruise activated
Cruise Activated	Cruising canceled (init)	Cruising/ set desired speed	Increasing speed	Decreasing speed	NA	NA	NA	Cruising	Cruise deactivated
	Cruising	Cruising/ set desired speed	Increasing speed	Decreasing speed	NA	Cruising canceled	Cruising canceled	NA	
	Increasing speed	NA	NA	NA	Cruising/set desired speed	NA	NA	NA	
	Decreasing speed	NA	NA	NA	Cruising/set desired speed	NA	NA	NA	

Note: The Cruise Activated state is itself a state diagram.

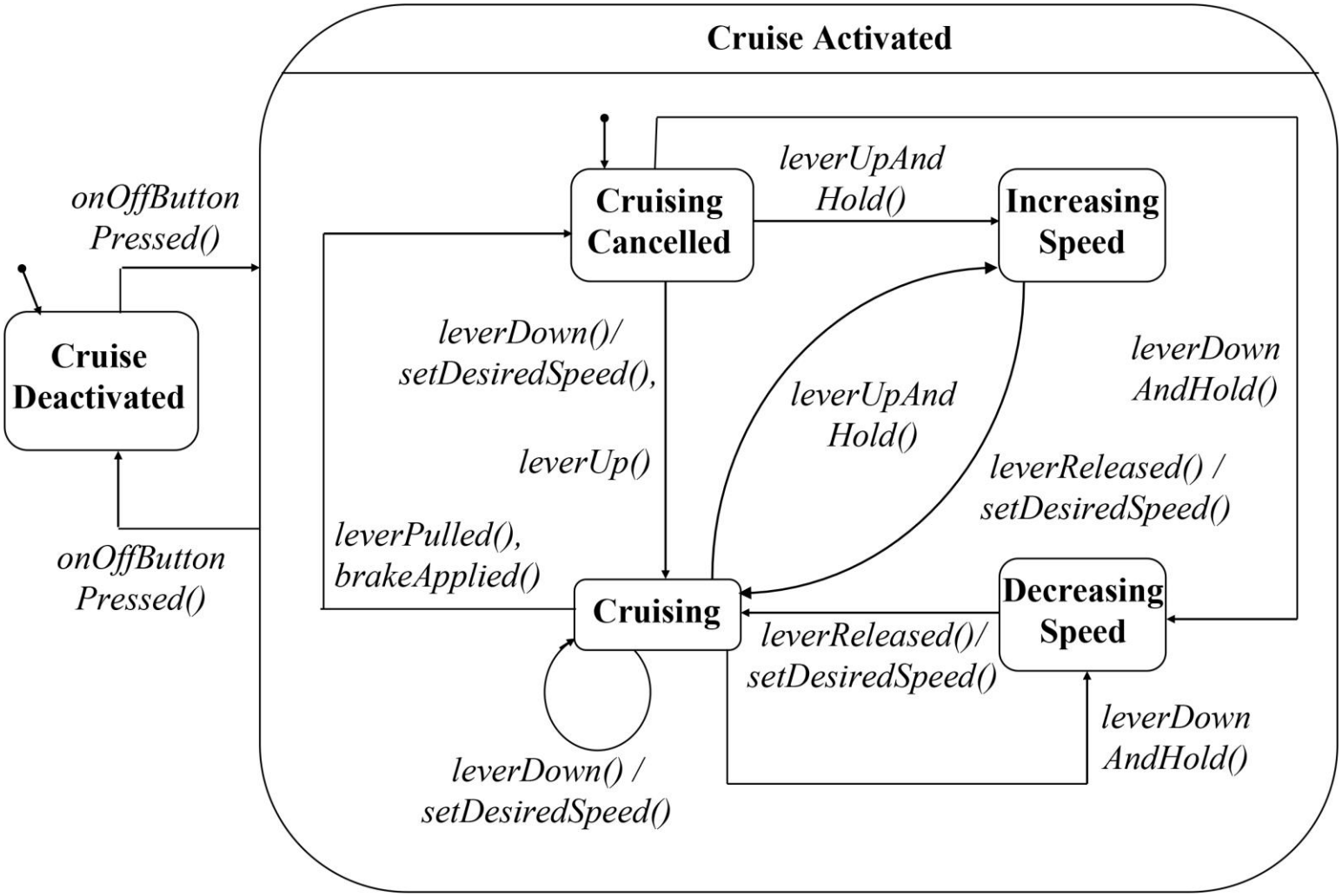
Access the text alternative for these images

Converting to State Diagram



Access the text alternative for these images

Converting Texts to Function Calls



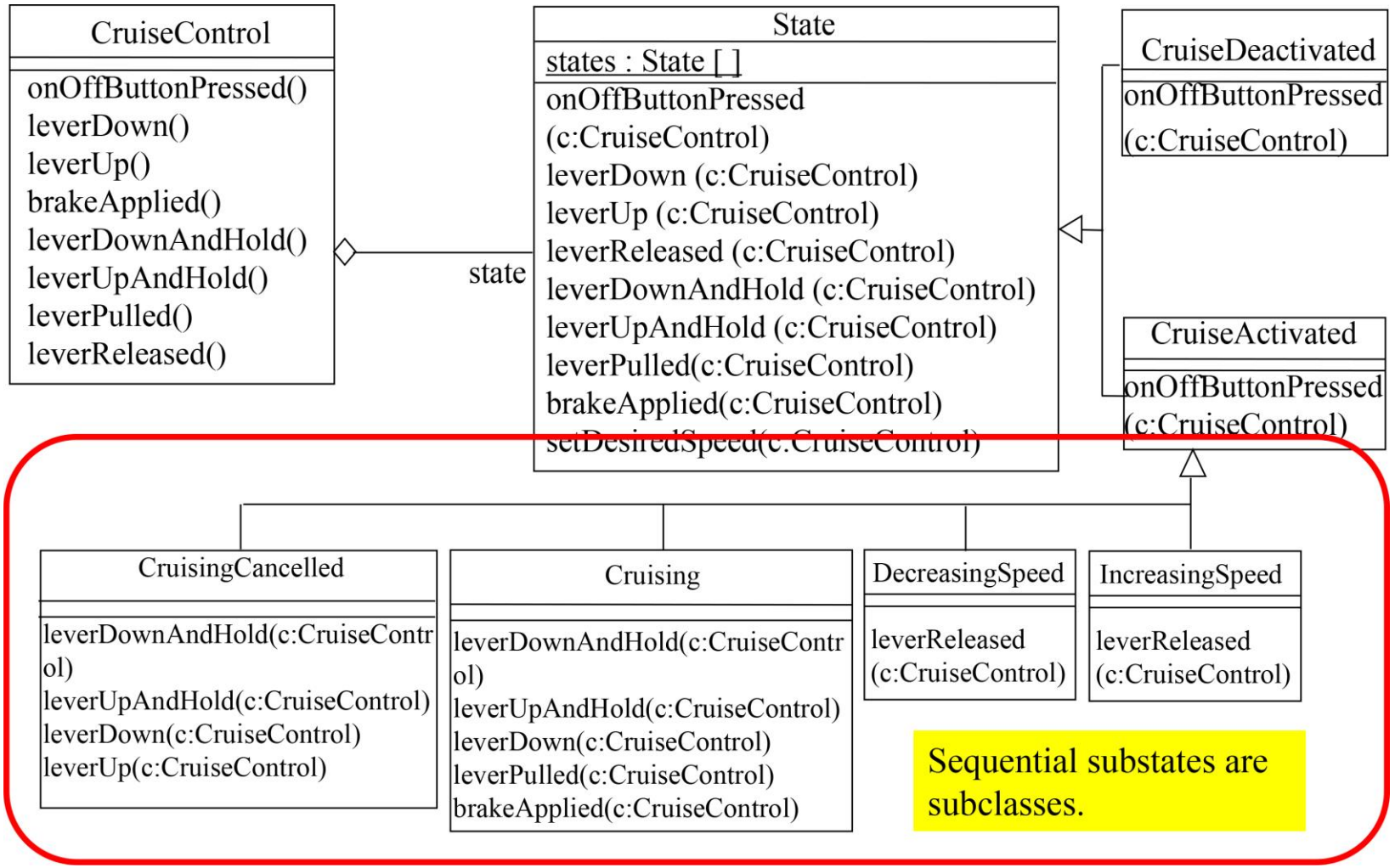
Access the text alternative for these images

Update Design Class Diagram

- Add methods labeling the transitions to the subject class:

CruiseControl
onOffButtonPressed()
leverDown()
leverUp()
brakeApplied()
leverDownAndHold()
leverUpAndHold()
leverPulled()
leverReleased()
setDesiredSpeed()

Applying State Pattern to Cruise Control



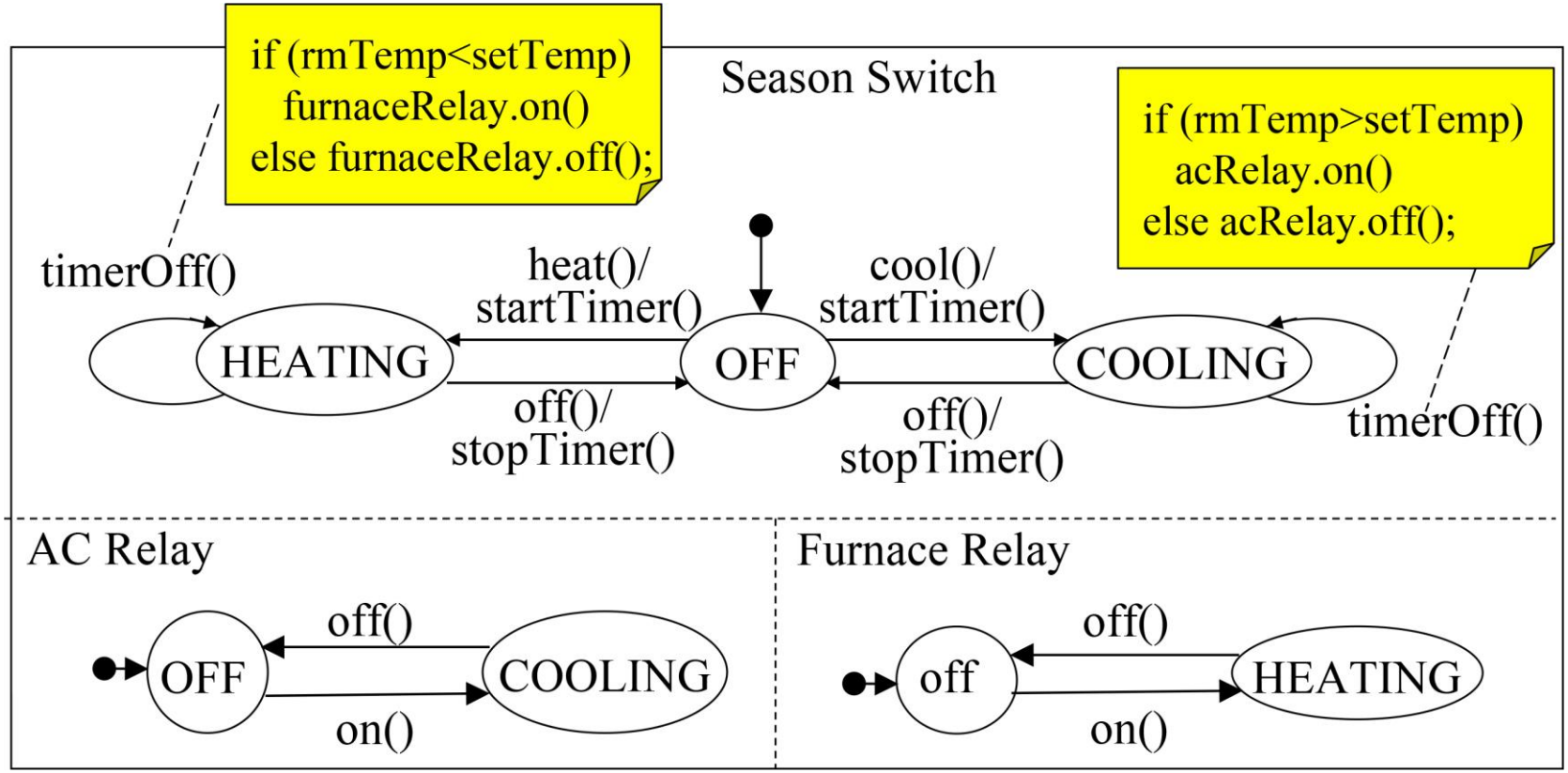
Access the text alternative for these images

Benefits of State Pattern

- Significantly reduces the cyclomatic complexity of the state handling code.
- Easy to add states --- simply add state subclasses and implement the relevant operations.
- Easy to add transitions --- simply add operations to the State class and implement the operations in relevant State subclasses.
- Easy to understand, implement, test and maintain.

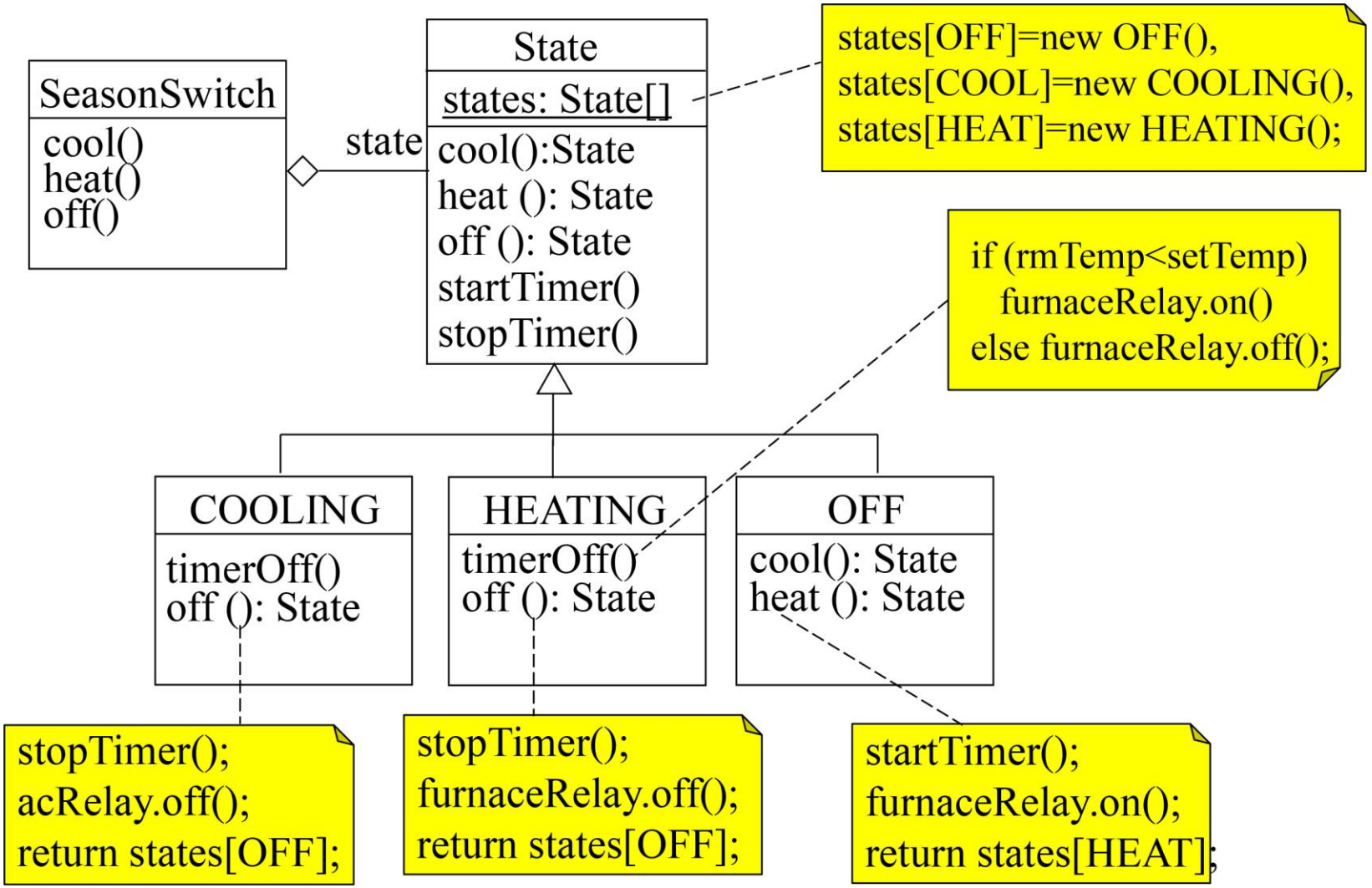
Class Exercise

- Apply the state pattern to the season switch state diagram of a thermostat.



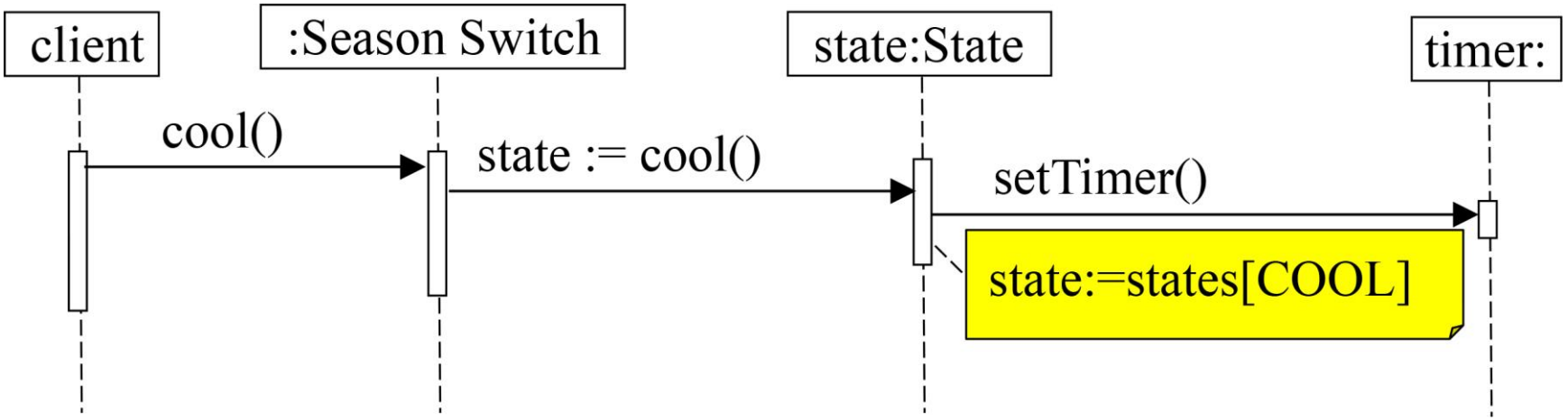
Access the text alternative for these images

The Season Switch State Pattern



Access the text alternative for these images

Object Interaction in Season Switch State Pattern



Access the text alternative for these images

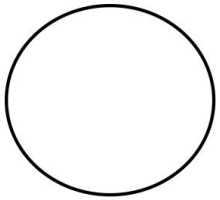
Transformational Schema for Real Time Systems 1

- Ward and Mellor extended DFD with control flows and control processes.
- Control processes are modeled by Mealy type state machines.
- Control processes control the ordinary data transformational processes.
- Control flows represent events or triggers to control processes and responses of control processes to transformational processes.

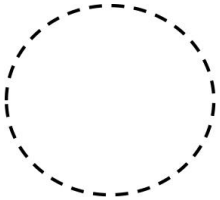
Transformational Schema for Real Time Systems 2

- Real time data flows, which must be processed quickly enough to prevent losing the data.
- Data flows and control flows may be related using logical connectors.
- Timing may be specified for state transitions and data transformation processes.

Real Time Systems Design 1



transformational processes, representing computations or information processing activities



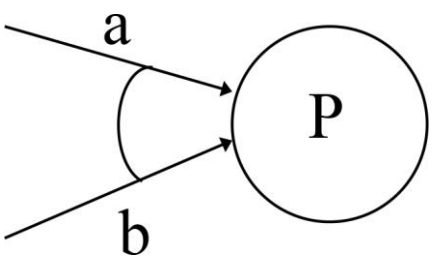
control processes, representing system's state dependent behavior, which is modeled by a Mealy type state machine

continuous data flow, which must be processed in real time

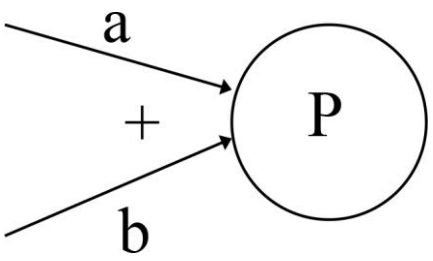
ordinary or discrete data flow

event flow or control flow that triggers a transition of the state machine of a control process, or a command from a control process to a transformational process

Real Time Systems Design 2



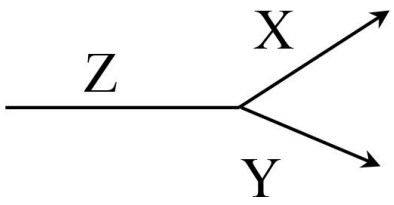
indicates that both data flow a and data flow b are required to begin executing process P



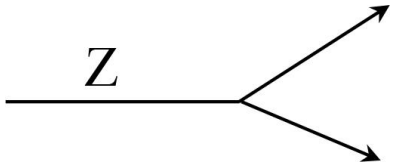
indicates that either data flow a or data flow b is required to begin executing process P

These logical connector can be applied to both data flow and control flow and transformational process and control process.

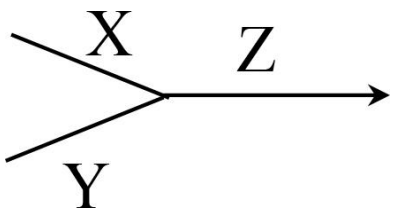
Real Time Systems Design 3



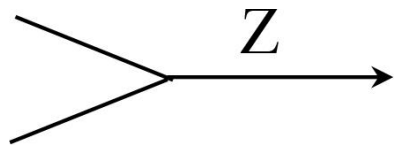
Two subsets of Z are used by two different successor processes.



All of Z is used by two different successor processes.

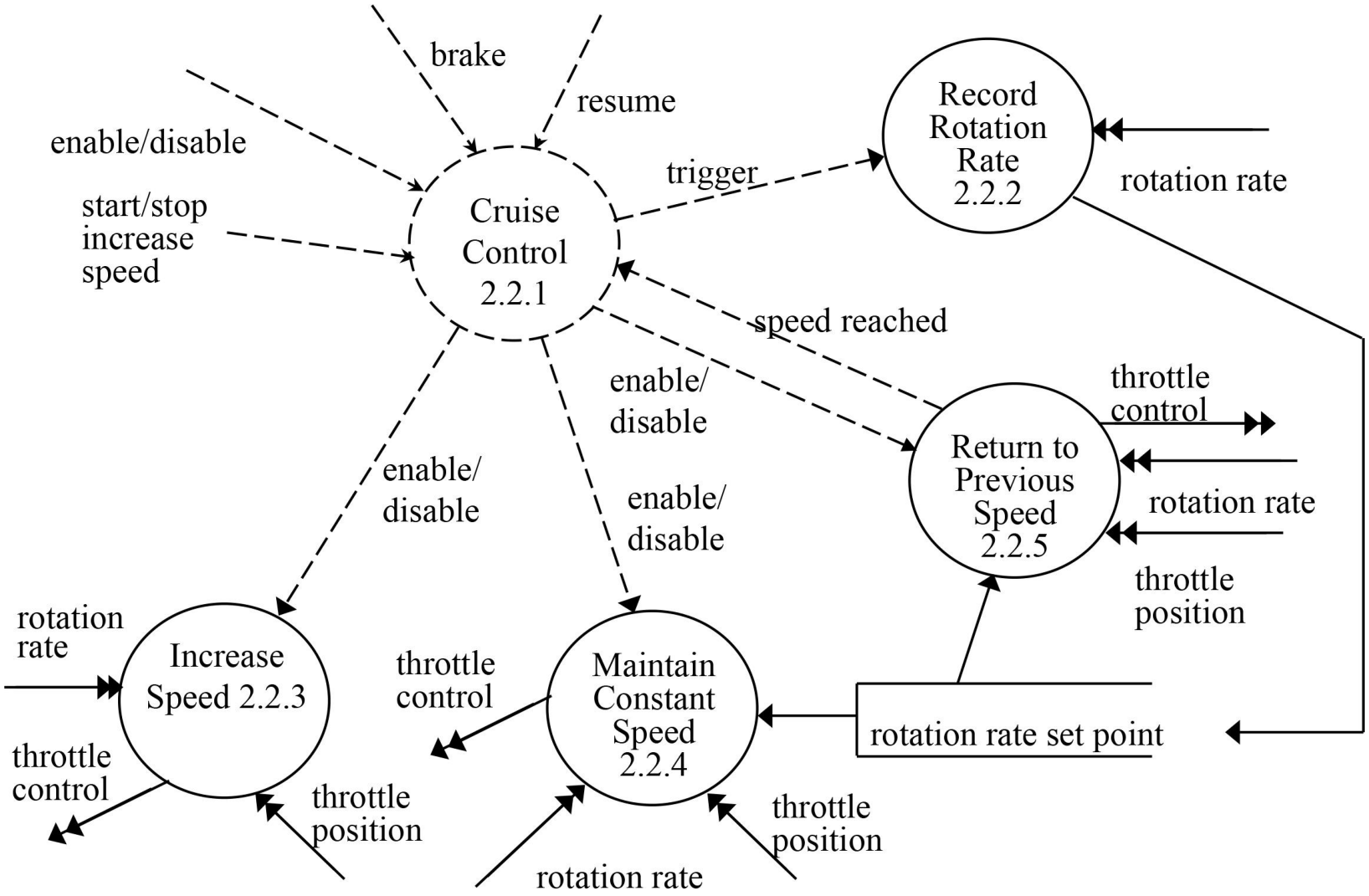


Z is composed of Two subsets provided by two different predecessor processes.



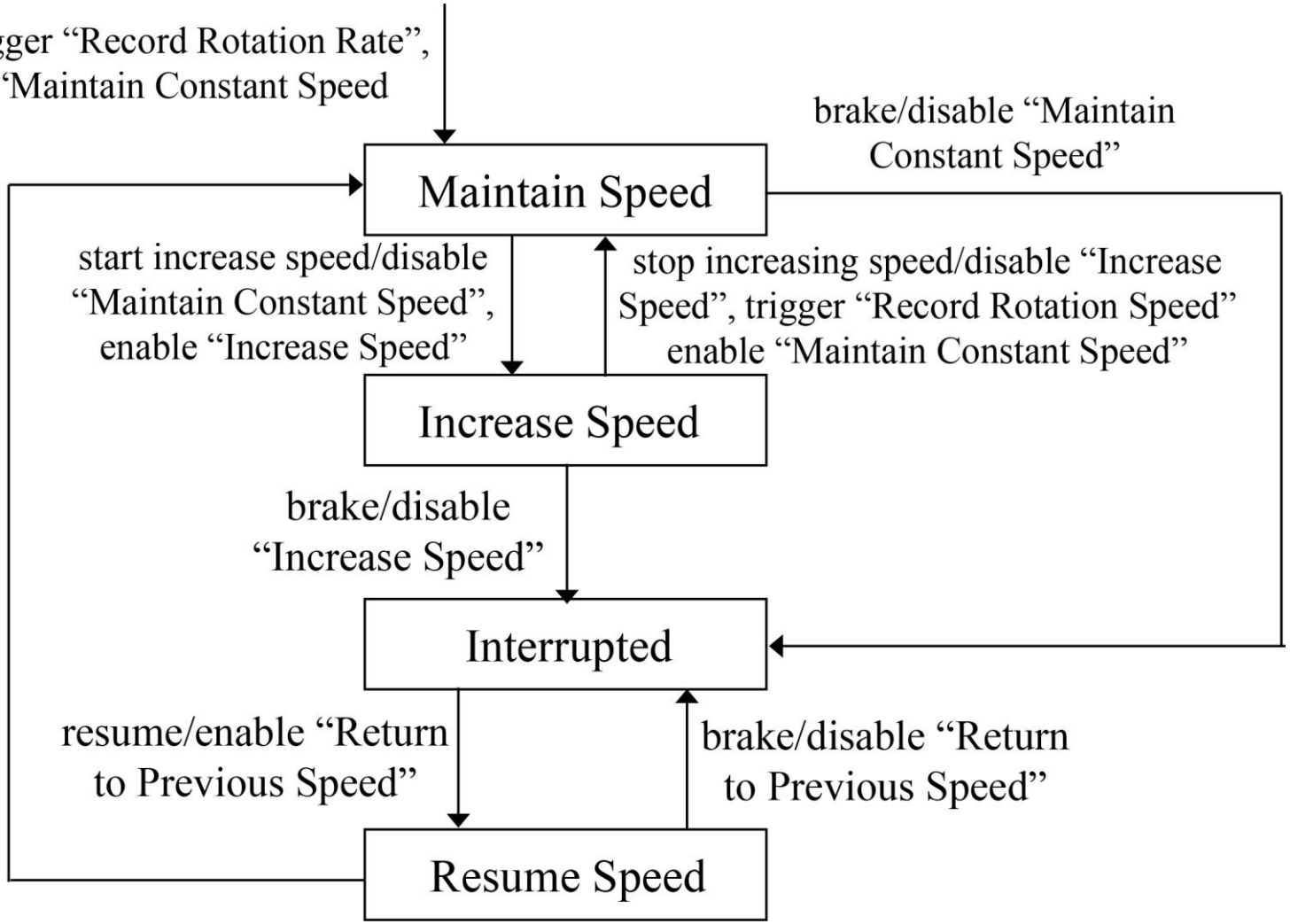
All of Z is provided by either one of two predecessor processes.

Cruise Control Example



Access the text alternative for these images

Cruise Control State Machine

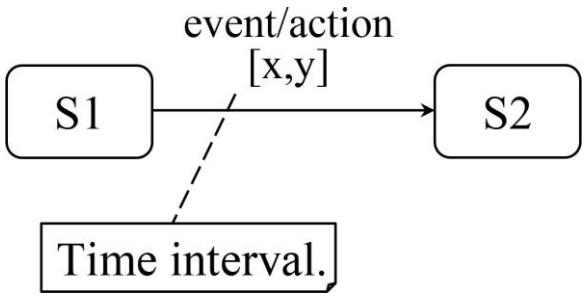


Access the text alternative for these images

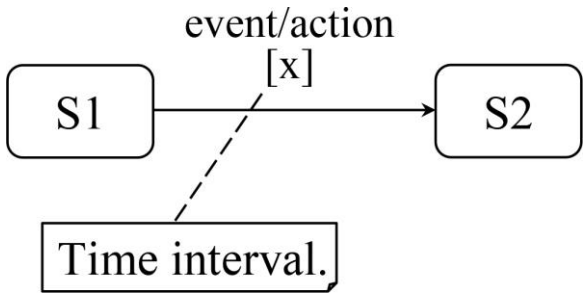
Timed State Machine

- Time intervals can be used to label the state transitions.
- The time intervals define the timing lower bounds and upper bounds allowed for processing the event and executing the list of actions.
- The time interval can be decomposed to define the allowed times for processing the event, and executing each of the actions in the action list.
- Similarly, time can be defined for state entrance action, state exit action, and state activity.

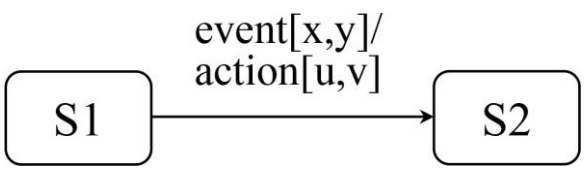
State Diagram for Real Time Systems



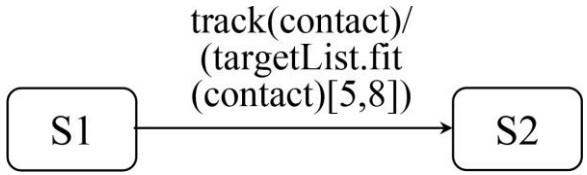
The time allowed for processing the event and executing the action is x to y time units.



The time allowed for processing the event and executing the action is x time units.



The time allowed for processing the event is x to y time units and executing the action is u to v time units.



The time allowed for targetList object to fit the contact with a tracked target is 5 to 8 time units.

Applying Agile Principles

- Work closely with the customer and users to identify and model the state behavior.
- Capture the state behavior at a high level, lightweight, and visual.
- Value working software over comprehensive documentation—do barely enough modeling.

Because learning changes everything.®

www.mheducation.com