CSC 2210
Object Oriented Analysis & Design

Dr. Akinul Islam Jony
Associate Professor
Department of Computer Science, FSIT
American International University - Bangladesh (AIUB)

akinul@aiub.edu

Statechart Diagram

>> What is Statechart Diagram?
>> Statechart vs. Sequence Diagram
>> Statechart vs. Activity Diagram
>> Notations of Statechart Diagram
>> Building a Statechart Diagram
>> Defining Entry and Exit Actions
>> Defining Send Events
>> Order of Events
>> Applying Basic Statechart Diagram Notation to Case Study
>> Modeling Transition Events
>> Examples of Statechart Diagram
>> Exercises

What is Statechart Diagram?

>> The Statechart describes the life of an object in terms of the events that trigger changes in the object's *state.*

>> It is used to model the life time of an object.

>> It is also used to model the dynamic nature of the system.

>> It is a simple network of states and events.

>> It identifies both the external events and internal events that can change the object's state.

>> Statechart diagram describes the flow of control from one state to another state.

>> It is also called state diagram, state machines, or state transition diagram.

What is Statechart Diagram?

>> What is States:

-The *state of the object is simply its* current condition. States are defined as a condition in which an object exists, and it changes when some event is triggered.

- That condition is reflected in the values of the attributes that describe that object. There are behaviors in the system that alter those attribute values.

- Object or system can be viewed as moving from state to state.

>> So, objects states are changed by events.

- Internal event
- External event

>> An **event** is an occurrence of a stimulus that can trigger a state transition.

What is Statechart Diagram?

>> Purpose of Statechart Diagram:

- To model dynamic aspect of a system.
- To model lifetime of an object.
- To describe different states of an object during its lifetime.

Statechart vs. Sequence Diagram

>> Sequence diagram

-Sequence diagram models the interactions between objects

-The scope is a single scenario

>> Statechart diagram

- Statechart diagram models the effect that these interactions have on the internal makeup of the object

- The scope is the entire life of an object

Statechart vs. Activity Diagram

>> Activity diagram

- shows flow of control from activity to activity across various objects

- Activity diagrams used to model workflow to model an operation

>> Statechart diagrams

- shows flow of control from state to state within a single object

- used to model the dynamic aspects of a system or to model behavior from the perspective of single entity (such as a class)

Notations of Statechart Diagram

State:

A state is modeled as a rounded rectangle with the state name inside, as in Figure 20-1.

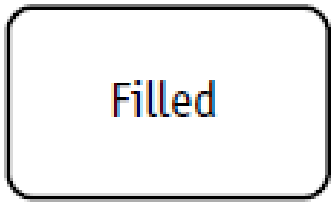


Figure 8-1 State symbol with only name compartment shown (minimum configuration)

Initial State:

A solid dot with an arrow pointing to the first state. The initial state indicates the state in which an object is created or constructed.

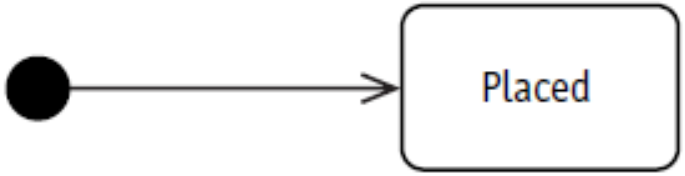


Figure 8-2 The initial state notation

Notations of Statechart Diagram

Event :

- The Statechart event notation is a line style arrow connecting one state to another state.
-The arrow is actually the transition associated with the event. The direction of the arrow shows the direction of the change from one state to another.
- Figure 8-3 shows the event “products available” that causes the transition (the arrow) from the state “Placed” to the state “Filled.”

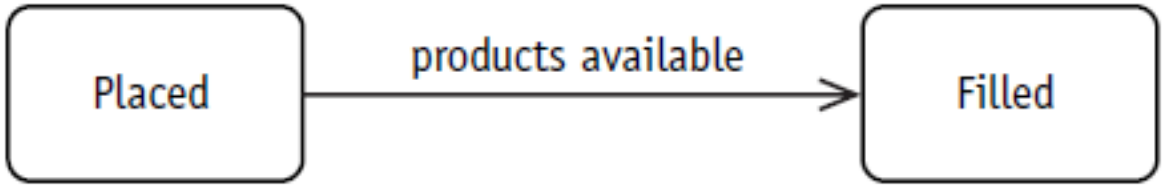


Figure 8-3 The state transition from “Placed” to “Filled”

Notations of Statechart Diagram

Action:

- An action is associated with an event. An action is the behavior that is triggered by the event, and it is the behavior that actually changes the attributes that define the state of the object.
- To model the action, place a forward slash after the event name followed by the name of the action or actions you want performed, as in Figure 8-4 where the "products available" event triggers the fillOrder() action. The act of filling the Order alters its contents and redefines its state.

Figure 8-4 Event/action pair

- An action is an **atomic task**, and as such it cannot be broken into component tasks, nor can it be interrupted. There are no break points within it and, furthermore, stopping it midway would leave the object state undefined.

Notations of Statechart Diagram

Final State:

-An object may reach a final state from which it may not return to an active state. In other words, you would never see an arrow going out of this state.

- The final state may also mean that the object has actually been deleted.

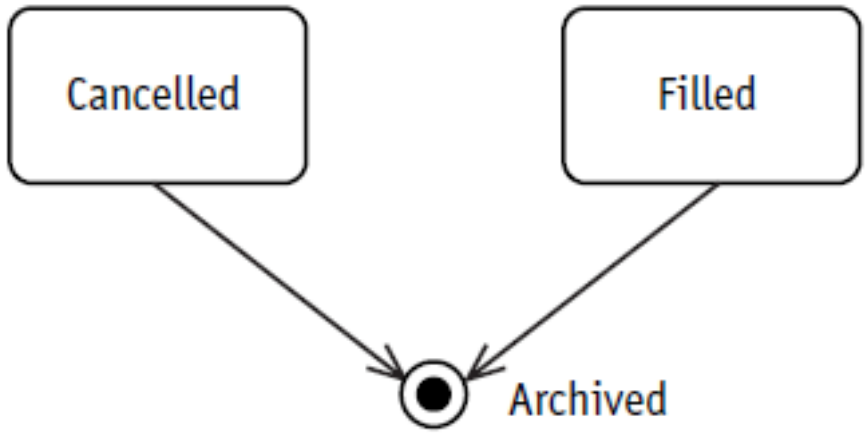


Figure 8-5 The final state notation

Notations of Statechart Diagram

Expanded State:

-The state icon can also be expanded. The purpose of the expanded form is to reveal what the object can do while it is in a given state.

- The notation simply splits the state icon into two compartments:

-- the name compartment and

-- the internal transitions compartment, as shown in Figure 8-6.

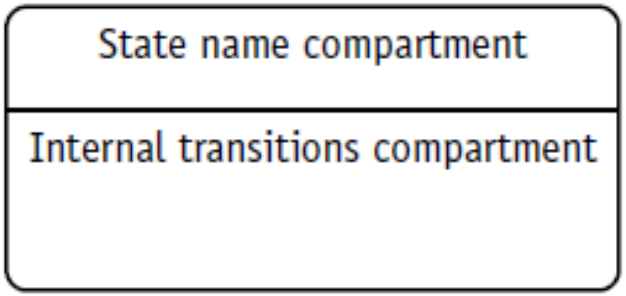


Figure 8-6 The expanded state icon

- The internal transitions compartment contains information about actions and activities specific to that state.

Notations of Statechart Diagram

Expanded State:

-Activities are processes performed within a state. An activity tends not to be atomic, that is, an activity may be a group of tasks. Activities may be interrupted because they do not affect the state of the object.

- Activities just do work. They do not change the state of the object.

-For example, Figure 8-7 models the active state of the Customer object. While in that state, the customer object generates a monthly invoice for the customer's purchasing activity and generates monthly promotions tailored to the Customer.

- To model activities within a state, use the keyword Do: followed by one or more activities.

- Actions alters the state. You must not interrupt actions

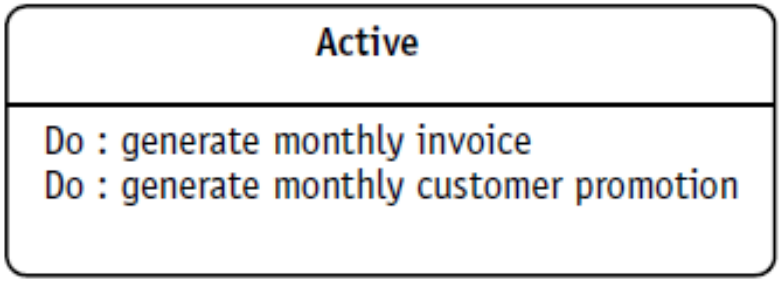


Figure 8-7 The expanded state icon with activities

- These activities will be performed from the time the object enters the state until the object leaves the state or the activity finishes.

Building a Statechart Diagram

Problem Statement:

We track current customer status to help avoid uncollectable receivables and identify customers worthy of preferred treatment. All customers are initially set up as prospects, but when they place their first order, they are considered to be active.

If a customer doesn't pay an invoice on time, he is placed on probation. If he does pay on time and has ordered more than $10,000 in the previous six months, he warrants preferred status. Preferred status may be changed only if the customer is late on two or more payments. Then he returns to active status rather than probation, giving him the benefit of the doubt based on his preferred history.

Building a Statechart Diagram

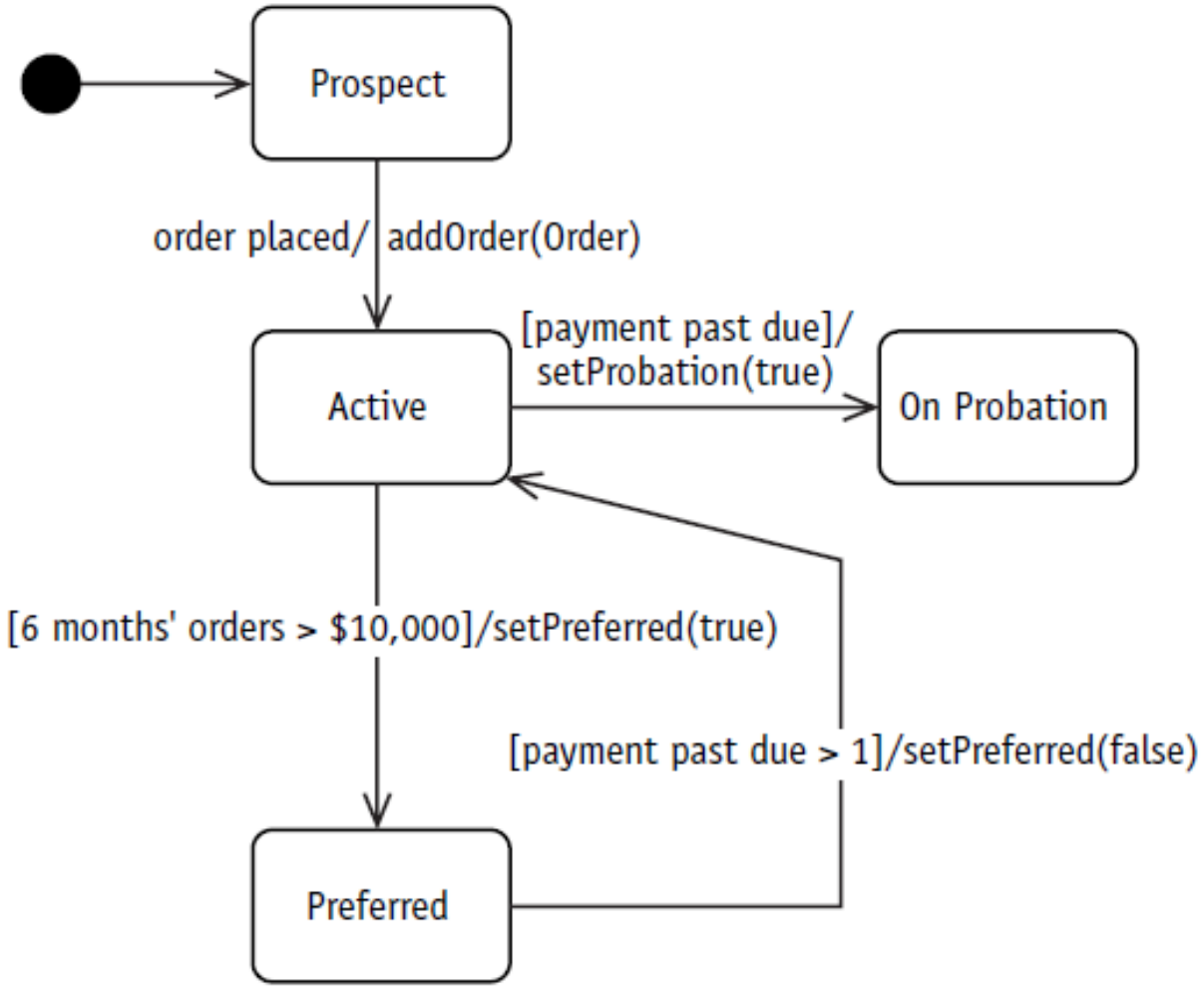


Figure 8-8 Statechart Diagram based on the Problem Statement

Note: This Statechart did not have a final state because within the scope of the problem statement there is no time when a customer object can no longer change. “On Probation” might be a final state because there are no arrows coming out of it, but this happened only because of the limited size of the example.

Building a Statechart Diagram

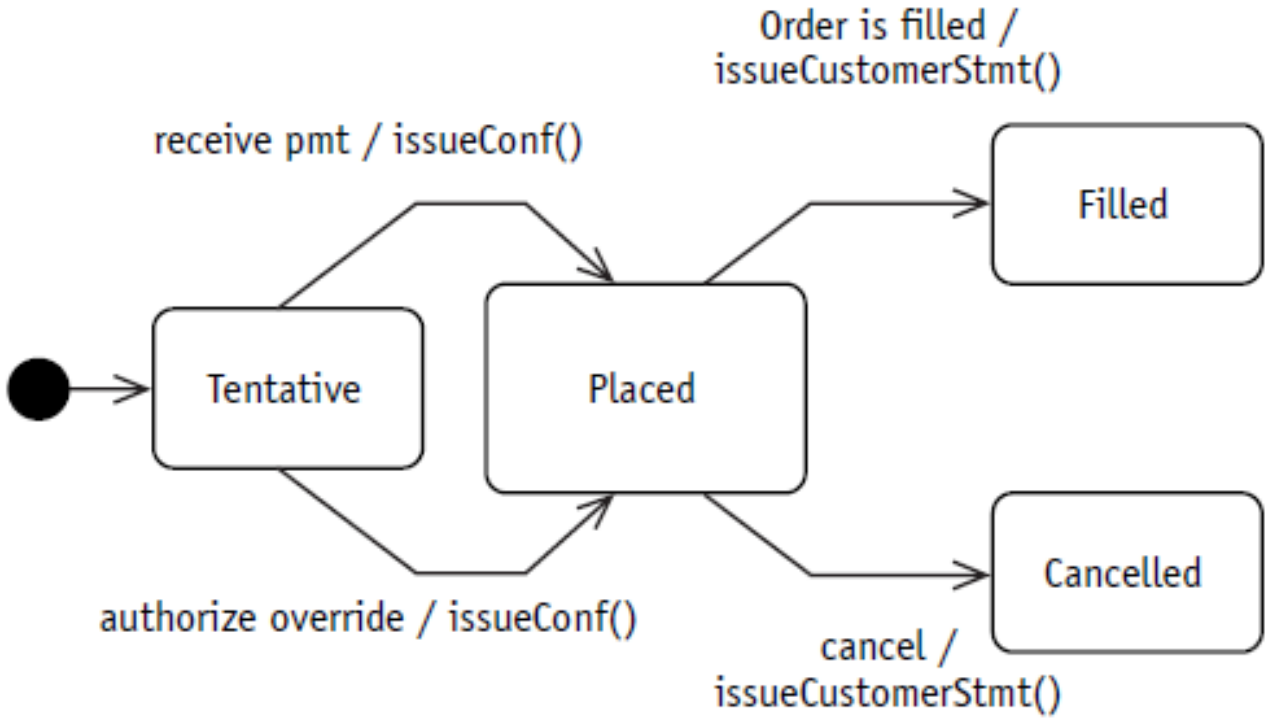
The process of building a Statechart

1. Model the initial state.
2. Identify the event(s) that change the object from the initial state to some other state
3. Name the new state
4. Draw the transition from the first state to the second
   - Label the transition arrow with the event that triggers the transition
5. Identify the action(s) associated with the event and that actually change the object attributes
   - Add the action after the event name and preceded by a forward slash
6. Continue the process with each new event until all events have been accounted for
7. If there is a state from which the object can never leave, convert this state to the final state notation

Defining Entry and Exit Actions

>> Modeling state transitions often results in more than one event that changes the object to the same state. Each of those events may have a corresponding action.

>> The UML offers two simplifications called entry & exit actions



In Figure 8-9, the Statechart for the *Order* object says that you can transition the Order from *Tentative* to *Placed* by either receiving the payment for the order or getting an override authorization. But both events require the same action: issue an order confirmation (*issueConf()*).

Figure 8-9 Redundant actions entering and exiting the *Placed* state of the Order

Defining Entry and Exit Actions

>> Whenever an action must take place with *every* event that transitions to the same state, you can write the action(s) once as an ***entry action***.

- When the redundant action is replaced by the entry action, you can remove it from the individual event arrows
- This simplifies the diagram while preserving the same meaning
- You would read the diagram to say, “Every time you enter this state issue an order confirmation.”

>> In an internal transitions compartment of an expanded state icon, we can model something called *entry actions*.

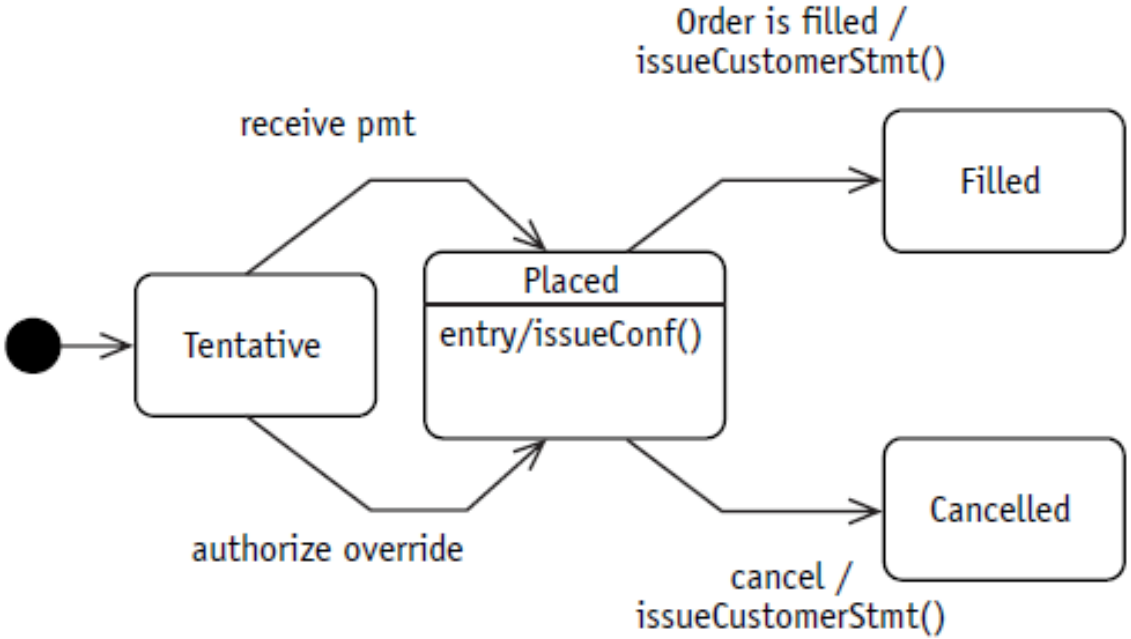


Figure 8-10
Consolidating the **entry actions**

Defining Entry and Exit Actions

>> The same simplification may be used for actions associated with events that leave a state with ***exit actions***

- Modeled in the same manner as entry actions

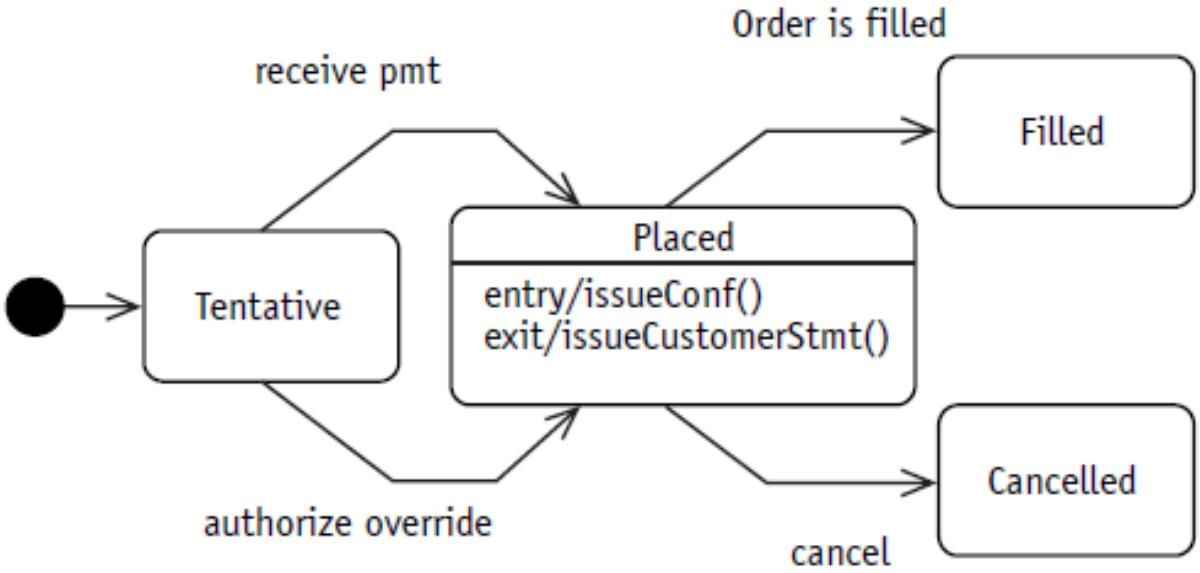


Figure 8-11 shows the *exit/action(s) notation* added to the internal transitions compartment of the *Placed* state and the actions removed from the event arrows (i.e., remove redundancy).

Figure 8-11 Consolidating the **exit actions**

Note: Just remember that they may only be used when the action takes place *every* time you enter (for entry actions) or *every* time you exit (for exit actions) the state

Defining Send Events

>> A *send* event is used when the object in the Statechart diagram needs to communicate with another object

>> An outgoing event must define the receiving object whether it is only one object or a broadcast to many objects

- Simply provide the object name followed by a period before the action expression
- This is often referred to as the *dot notation*

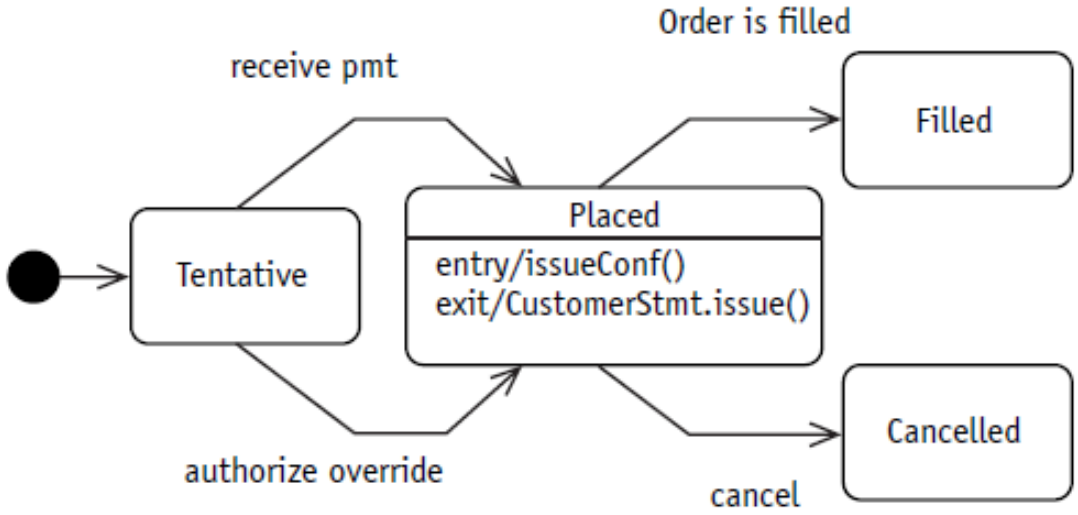


Figure 8-12 Consolidating the **exit actions**

When the Order is cancelled, the Order is supposed to issue a customer statement. But the customer statement is another object that takes care of issuing itself. See the *exit action notation* in Figure 8-12 where the ***issue()*** action is now being sent to the *CustomerStmt* object.

Order of Events

When an event occurs, the order of execution runs like this:

1. If an activity is in progress in the current state, interrupt it (gracefully if possible)
2. Execute the exit action(s)
3. Execute the actions associated with the event that started it all
4. Execute the entry: action(s) of the new state
5. Execute the activity or activities of the new state

Applying Basic Statechart Diagram Notation to Case Study

Inventory control: Problem Statement:

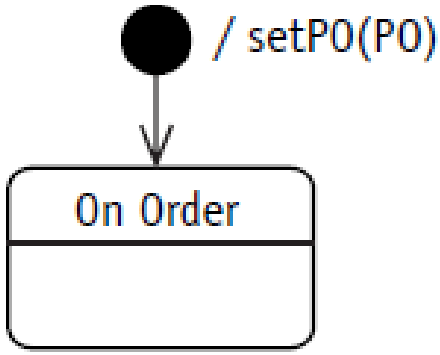
Products are first entered into our system when they are ordered using a purchase order (P.O.). Each product keeps a record of the originating P.O. (*1) When the product is received, it is placed into inventory by recording the location where it is placed. When the product is received, you have to update the P.O. to indicate that you have received the product. (*2)

When a product is sold, the product tracks the order to which it belongs. (*3) When a product is sold, it is also packed for shipping and the associated shipment is recorded. (*4) When the product is shipped, you need to record the shipper and the date it was picked up. (*5) Occasionally, a product is returned. In that case, you put the product back into inventory and record the location. (*6)

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*1)

- The initial state is On Order
- The action is to record the purchase order
- Notice that an action may be associated with the creation of the object



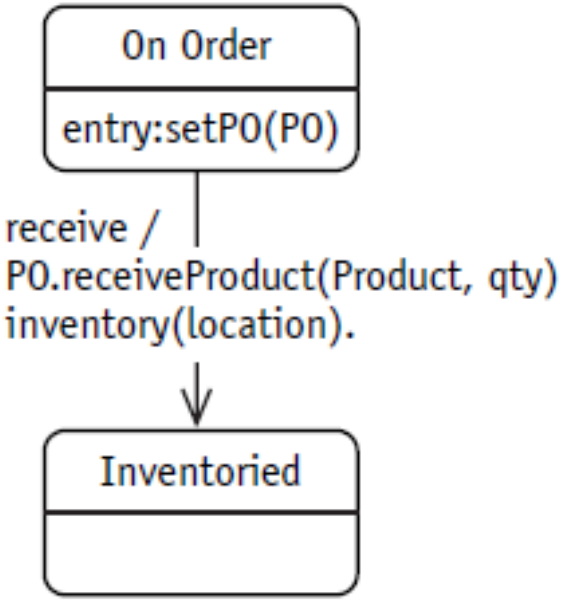
"Products are first entered into our system when they are ordered using a purchase order (P.O.). Each product keeps a record of the originating P.O." (*1)

Figure 8-13 Model the initial state of the product

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*2)

- The transition from On Order to Inventoried is triggered by the *receive* event
- The associated actions are to update the purchase order object with the product and quantity received and update the product with the inventory location



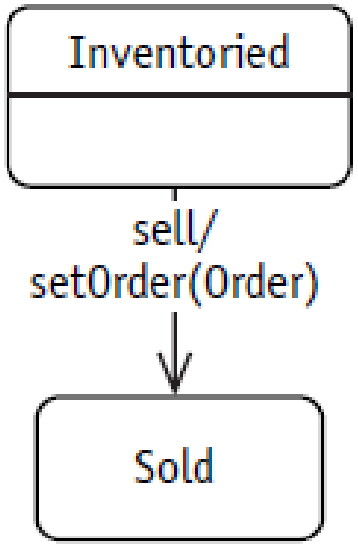
"When the product is received, it is placed into inventory by recording the location where it is placed. When the product is received, you have to update the P.O. to indicate that you have received the product."(*2)

Figure 8-14 Inventory the product and update the P.O.

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*3)

- The *sell* event triggering the transition to the sold state and the action to record the Order that now holds the product



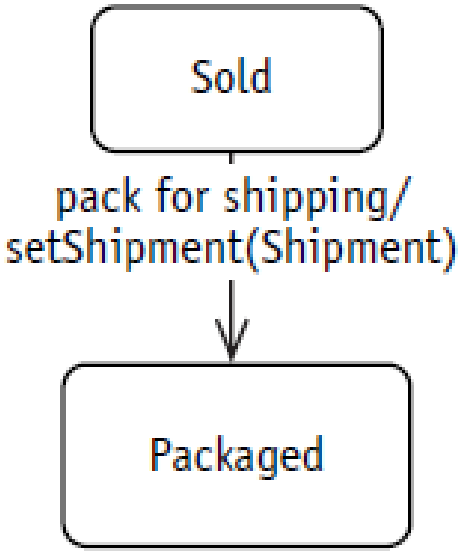
"When a product is sold, the product tracks the order to which it belongs." (*3)

Figure 8-15 Sell the product and record the order.

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*4)

- The event *pack for shipping* triggering the transition from sold to packaged
- The *pack for shipping* event also triggers the action to record the shipment



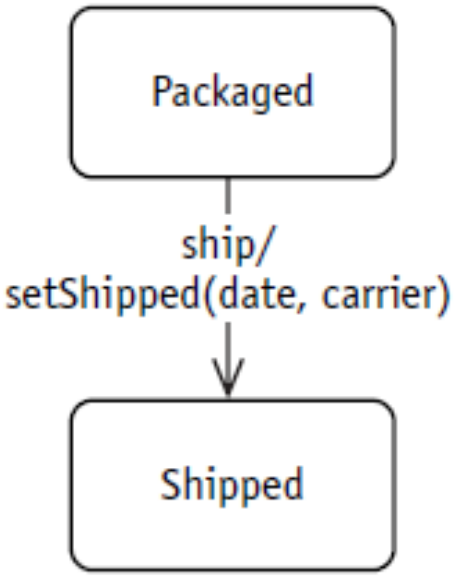
"When a product is sold, it is also packed for shipping and the associated shipment is recorded." (*4)

Figure 8-16 Pack the product for shipping

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*5)

- The *ship* event is the trigger
- The action is *setShipped* with the date and the carrier



"When the product is shipped, you need to record the shipper and the date it was picked up." (*5)

Figure 8-17 Ship the product

Applying Basic Statechart Diagram Notation to Case Study

>> Constructing the Statechart diagram for the product object (*6)

- The *return* event requires us to record the inventory location
- The other incoming event also logs the inventory location

"Occasionally, a product is returned. In that case, you put the product back into inventory and record the location." (*6)

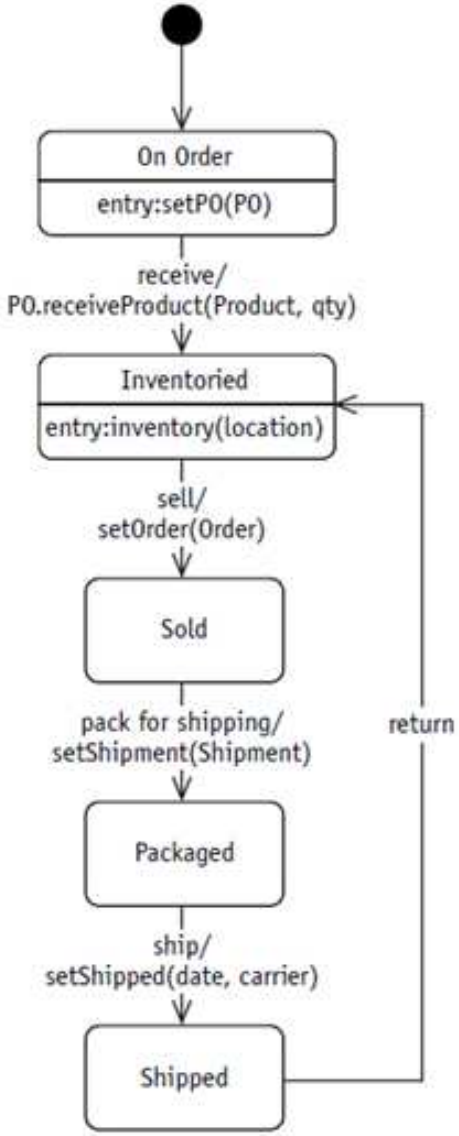


Figure 8-18
The completed Statechart diagram for the Product object

Modeling Transition Events

>> Five different event types

- *call events, time events, change events, send events,* and *guard conditions*

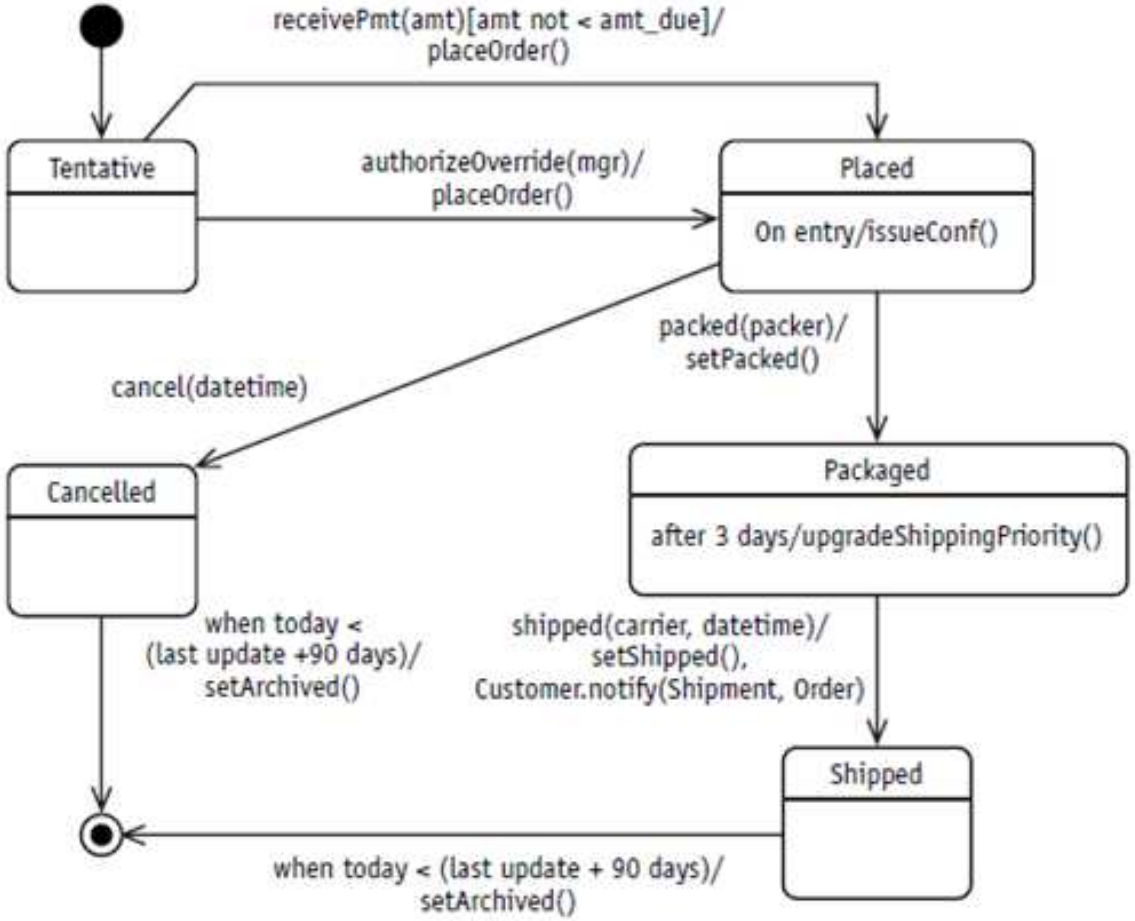


Figure 8-19 illustrates the Statechart diagram for an Order object. The Order is created in the *initial state of Tentative. Two events could cause it to change to the Placed state.* From *Placed* it may either be *cancelled*, or *packed* and made ready for shipping. After it is *packed*, it may be *shipped*. Then, whether it was *cancelled* or *shipped*, the Order is archived after 90 days.

Figure 8-19 The Statechart for a typical Order object

Modeling Transition Events

Call event:

- The most common event type
- It is basically the invocation of an operation on the receiving object
- This type of event is the merge of an event and an event action

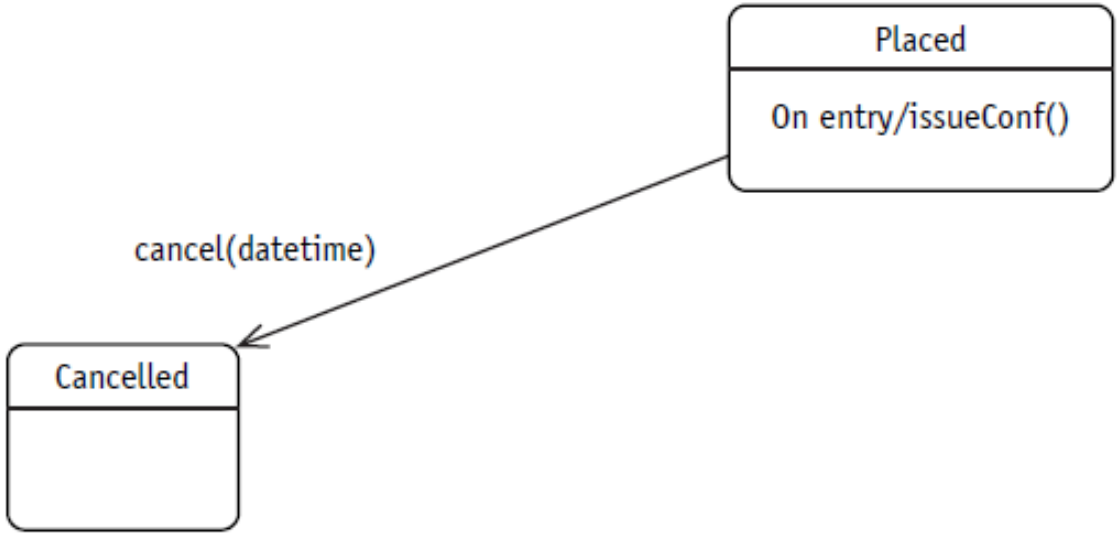


Figure 8-20 shows the transition from *Placed* to *Cancelled*. The transition is triggered by the *cancel(datetime)* event. *"cancel(datetime)"* is actually the operation signature on the Order.

Figure 8-20 Call event "cancel(datetime)"

Modeling Transition Events

Time event:

- A time event evaluates the passage of time as a trigger
- It is implies that the object supports some mechanism to monitor the passage of time
  - -- the mechanism could be a batch program that runs at intervals to update a time attribute
  - -- the mechanism could use a polling type of implementation where the object constantly checks to see what time it is
- Use the keyword *after* to specify the time increment to be evaluated

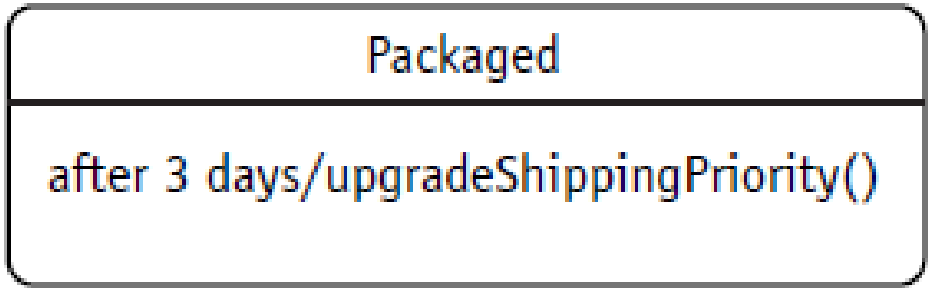


Figure 8-21 Time event "after 3 days"

While the Order is in the *packaged* state, it is waiting to be shipped. But if it has not been shipped within three days, the priority needs to be increased to ensure that it leaves in a timely manner. Figure 8-21 models an internal event within the *packaged* state called "*after 3 days*." This implies that there will be code within the object that watches the passage of time to know when to initiate the action "*upgradeShippingPriority()*."

Modeling Transition Events

Change event:

- A change event tests for a change in the object or a point in time
- Use the keyword *when* with the required test

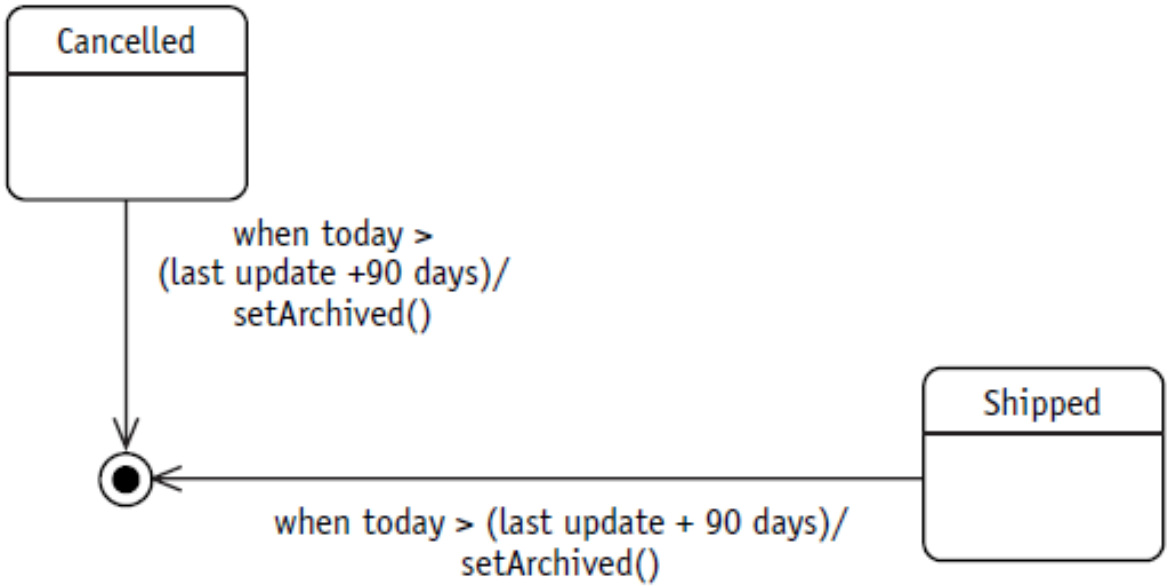


Figure 8-22 represents the change events that cause the Order to be archived. In both cases, the Order is waiting until there has been no activity on the Order for 90 days.

Figure 8-22 Change event "when today > (last update +90 days)"

- Note that this event is only evaluated while the Order is either *Cancelled* or *Shipped*
- Remember that what does not show on a Statechart tells you almost as much as what is shown on the diagram

Modeling Transition Events

Guard Condition event:

- Making events conditional
- A guard condition controls the response to an event
- When an event occurs, the condition is tested
- If the condition tests true, the corresponding transition takes place along with any and all associated actions; otherwise, the event is ignored

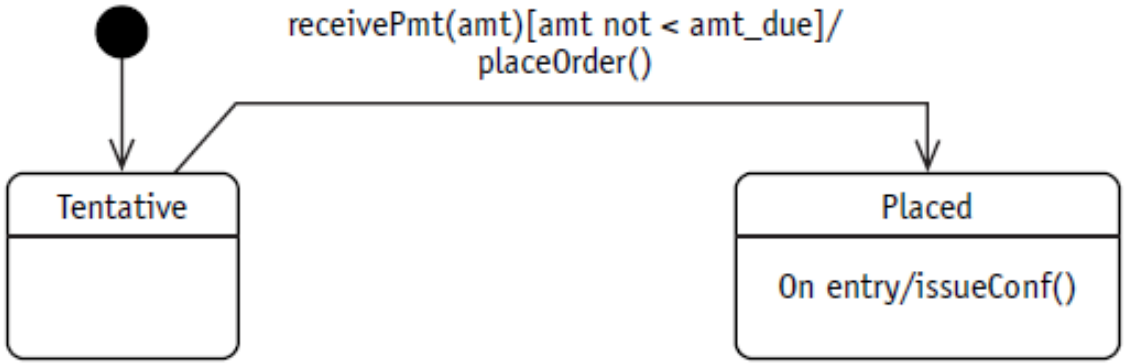


Figure 8-23 Guard condition [amt not <amt_due]

In Figure 8-23, the triggering event is "*receivePmt(amt).*" But receiving the payment will not upgrade the Order to *Placed* unless it was actually enough to pay for the Order, which is checked based on the guard condition. If the event is rejected, the object remains unchanged.

- Guard condition may actually be used by itself as a triggering event.

Modeling Transition Events

Send event:

- A send event models the fact that an object tells another object what to do
- A send event may be a response to a transition event or an internal event

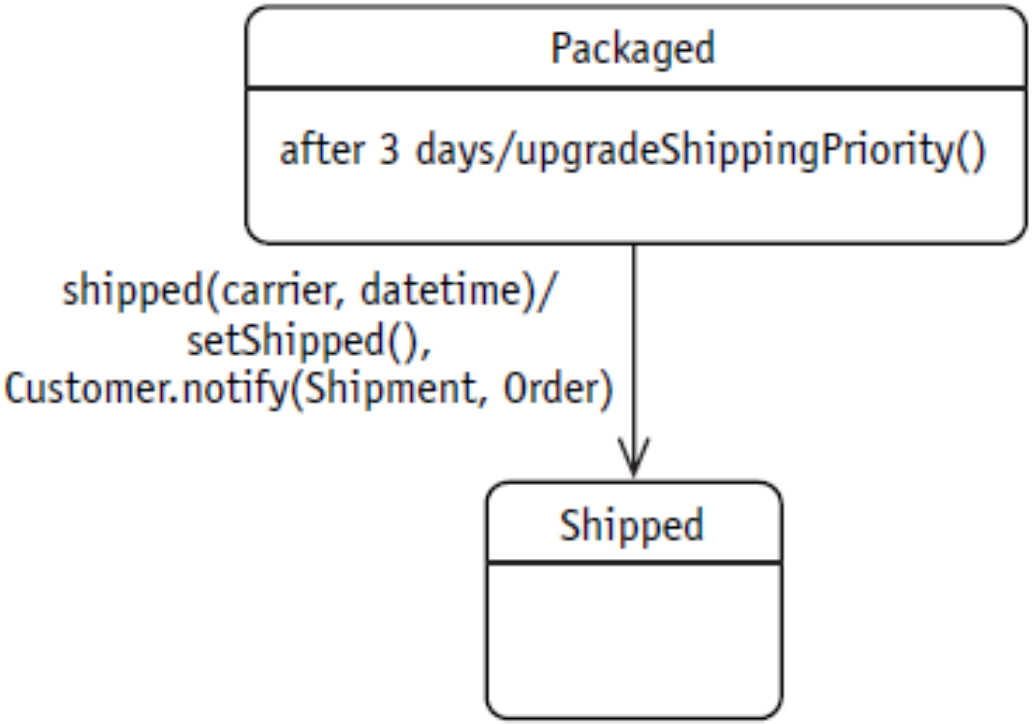


Figure 8-24 models a send event as part of the action required by a transition event. When the *shipped(carrier, datetime)* event occurs, two actions are triggered with the transition: *setShipped()* to change the state; and a message to the **Customer** object to generate a notification to the real customer that her Order has been shipped.

Figure 8-24 Send event *Customer.notify(Shipment, Order)*

Modeling Superstates & Substates

>> Modeling often requires different views of a problem

- High-level views simplify the model
- Low-level views focus on details of a problem
- The UML Statechart diagram supports the concept of nested states, allowing both high- and low-level views of object behavior and states

Modeling Superstates & Substates

Superstate:

-Simply a state that is expanded to show more detail

-The state rounded rectangle icon is expanded and the details are represented as one or more Statecharts within the superstate

-The name of the state is placed at the top

-Represents a high-level view of a complex situation

-Allows you to focus on the bigger, more general problem without getting lost in the details

- The substates are placed within the expanded superstate

Modeling Superstates & Substates

Substate:

-A state within a state, a lower level of detail within a state

- Provides a low-level view of a model element so that you can address specific issues individually and in terms of their interactions and interdependencies

-This detailed view also allows you to highlight concurrent states and focus on how to control the splitting and merging of concurrent states

Modeling Superstates & Substates

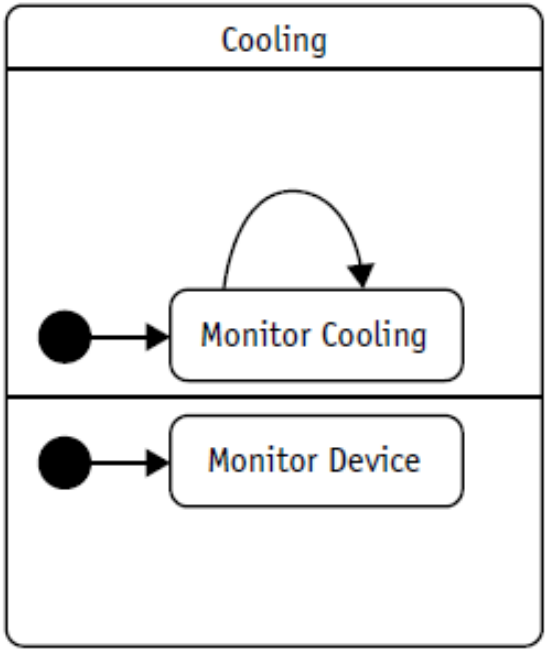


Figure 8-25
Superstate Cooling with two substates, Monitor Cooling and Monitor Device

>> The diagram (Figure 8-25) says that

-When the Thermostat enters the Cooling state, it splits into two concurrent substates, that is, it is now doing two things at the same time

- It is monitoring the progress of the cooling process, and it is monitoring the cooling device for any problems

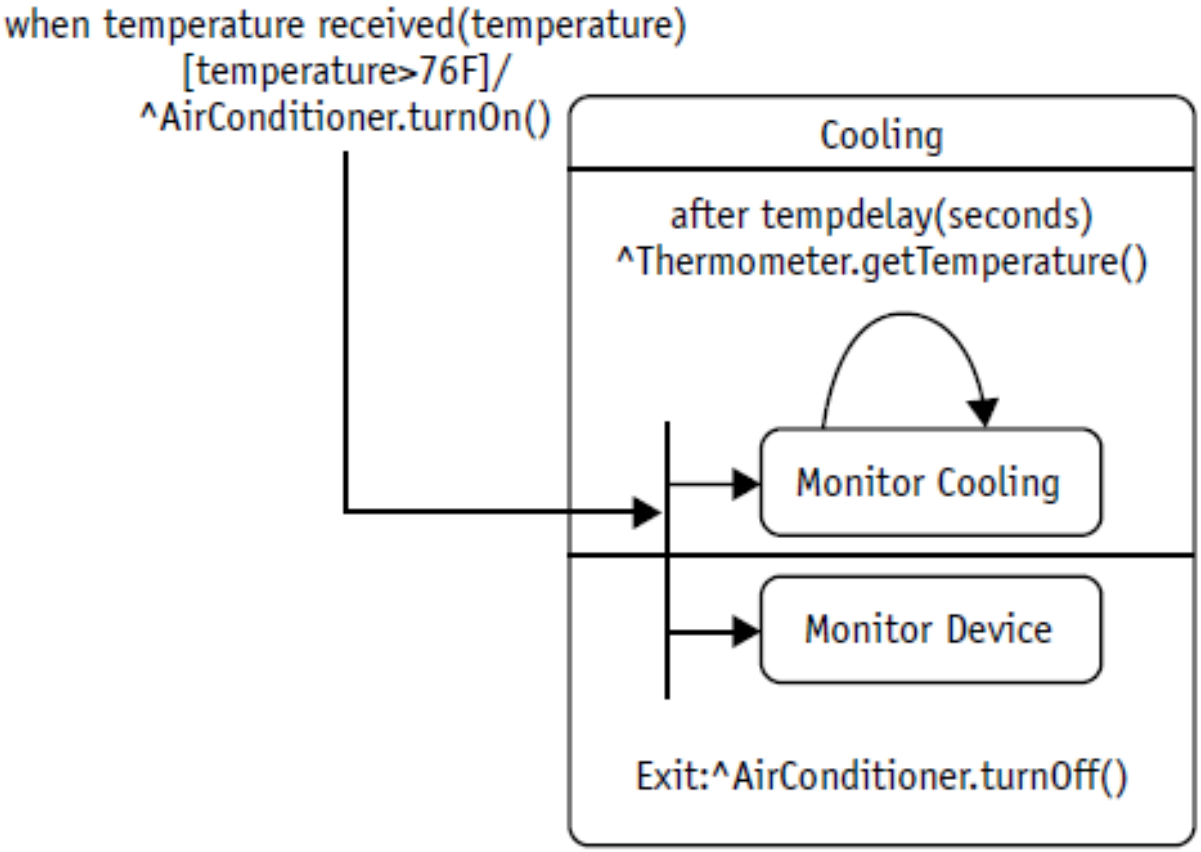
- The two substates start immediately upon entering the Cooling state

Modeling Superstates & Substates

Split of control:

- Split of control means that, based on a single transition, you want to proceed with multiple tasks

-Shown by a single transition divided into multiple arrows pointing to multiple states or substates



-The divide is accomplished with the fork bar you used in the Activity diagram.

- Merge of control can be modeled as multiple transition arrows pointing to a synchronization bar, as you saw earlier in learning about the Activity diagram.

Figure 8-26 Split of control

Modeling Superstates & Substates

Concurrency

- The UML supports concurrency within a state by allowing multiple Statecharts within a state
- Simply split the superstate internal transition compartment into as many separate compartments as needed, one for each sub-Statechart

In the thermostat example, the Thermostat is doing two jobs at the same time: monitoring the cooling device and watching for problems with the device. So the Cooling state internal transition compartment is split in two by a line.

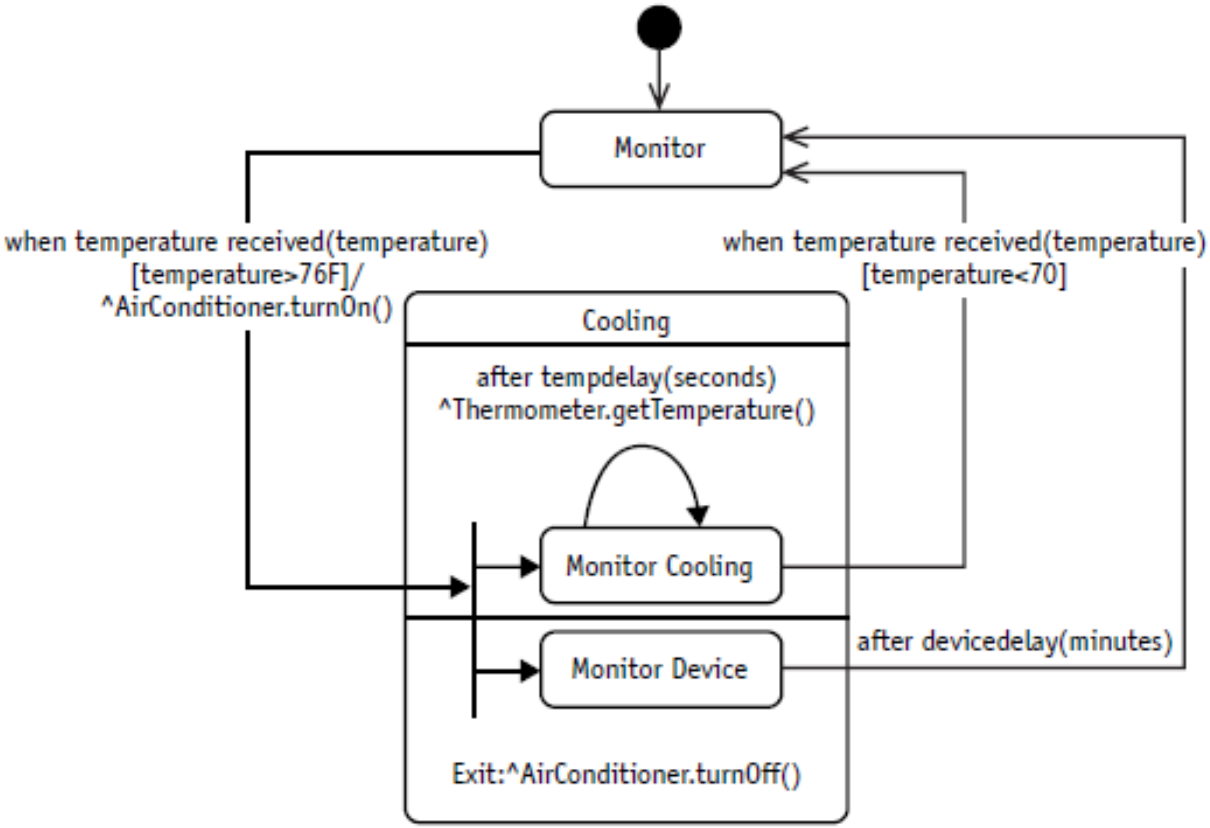
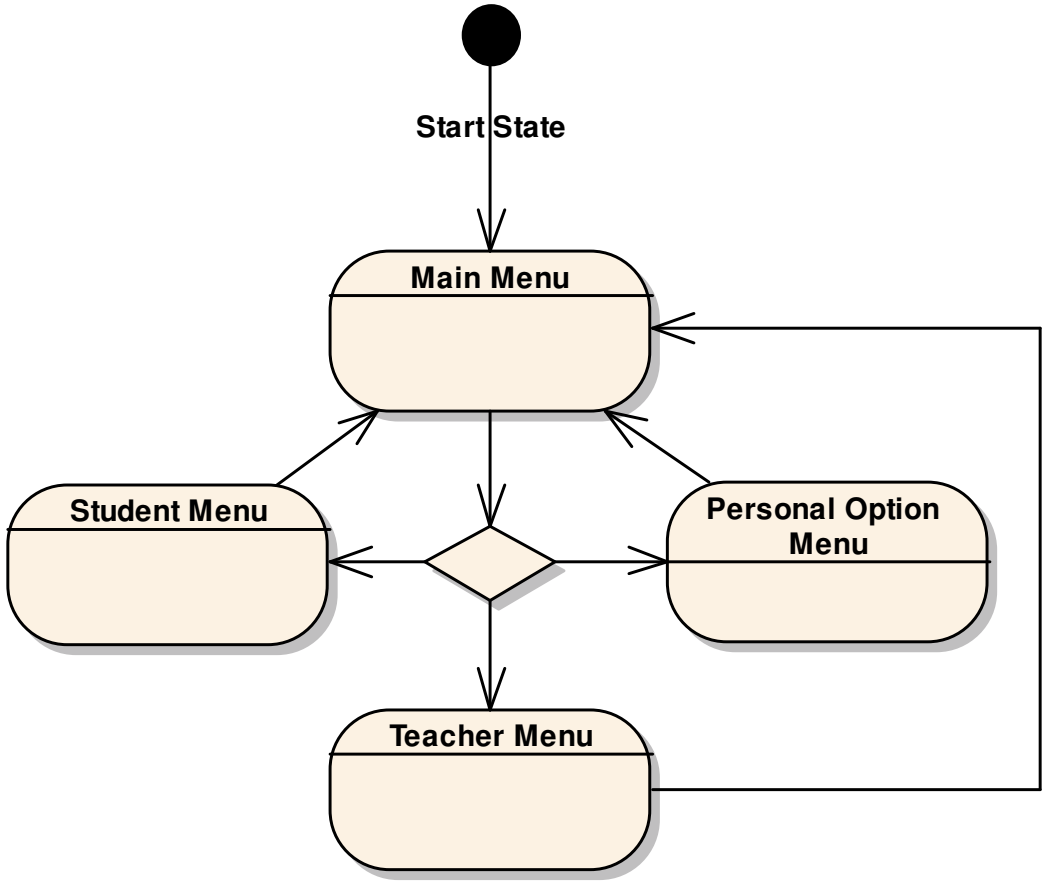


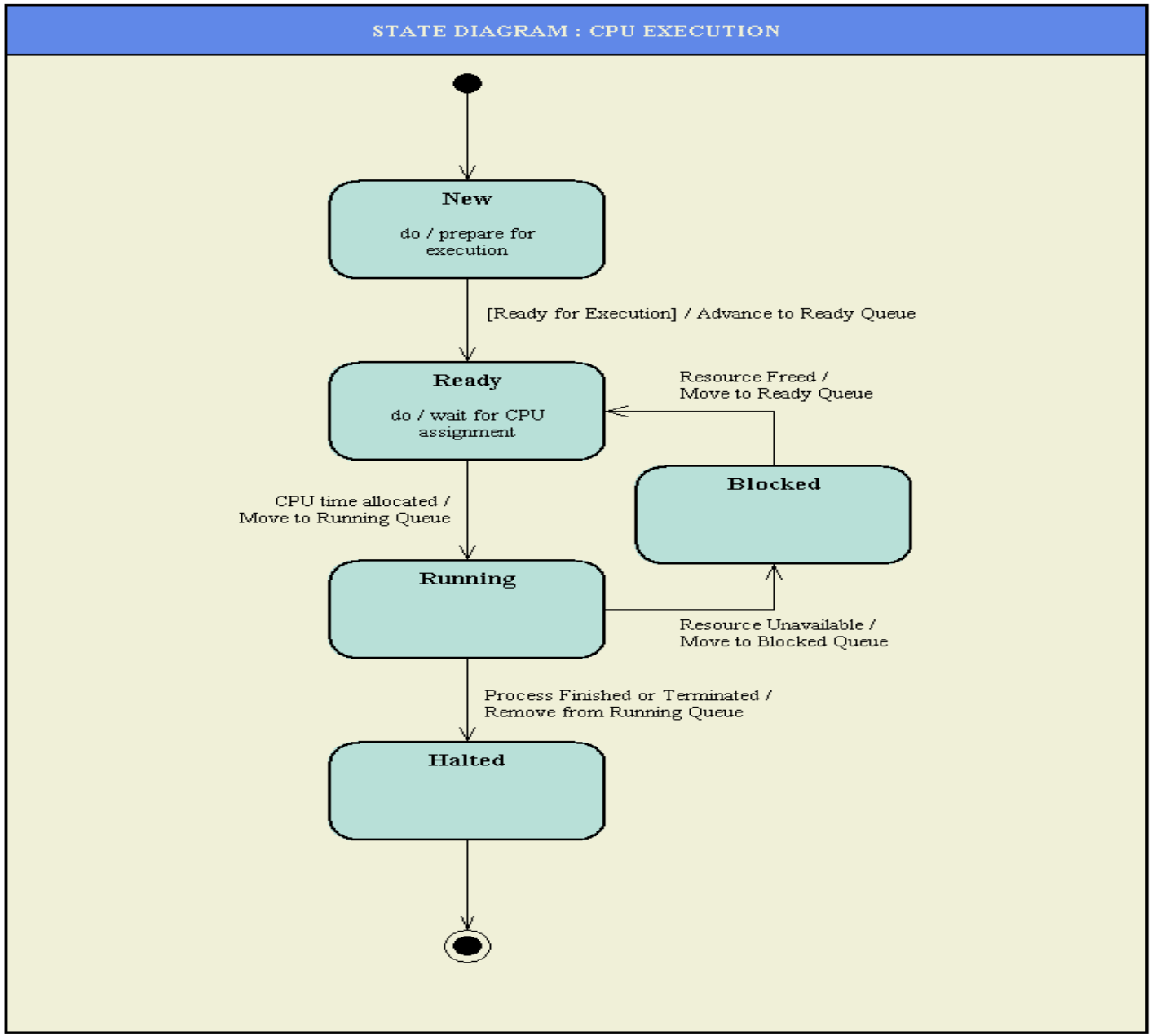
Figure 8-27
Multiple transitions out of a superstate

Examples of Statechart Diagram

The Main Menu state can navigate to any of the other menu states. Each of these can then navigate back to the Main Menu



Examples of Statechart Diagram



Exercises

Case 1:

A course is initially created in offered state. The offered courses are then included by the students in their requests and the requests are placed for registration. Some of the requested courses are then had to be checked against other completed courses by the students as their pre requisites. If the pre-requisites are fulfilled students are added in the courses. When a course reaches the limit of its capacity the course is then closed for further student requests. On the other hand if there are not enough students requesting for a course the course is cancelled.

Exercises

Case 2:

A course is initially created in offered state. The offered courses are then included by the students in their requests and the requests are placed for registration. Some of the requested courses are then had to be checked against other completed courses by the students as their pre requisites. If the pre-requisites are fulfilled students are added in the courses. When a course reaches the limit of its capacity the course is then closed for further student requests. On the other hand if there are not enough students requesting for a course the course is cancelled. Once the student registration is finished a time schedule is attached the course along with a teacher. At the end of the semester the course is evaluated by the authority. If the evaluation is satisfactory it is recommended for the next semester, otherwise it is recommended for up gradation.

Exercises

Case 3:

Products are first entered into our system when a request for purchase for that order is issued. A purchase order is then generated for the product and recorded with that associated product. The purchase order is then sent to the vendor. When the product is received from the vendor, it is examined by the quality control department. Once the product is approved by the quality control it is placed into inventory by recording the location where it is placed. If it is not approved the product is returned to the vendor. When a customer buys a product it is included in an invoice. Sometimes, a product is returned. In that case, you put the product back into inventory and record the location. If a product is lying in the inventory for more than six months it is returned to the vendor, but this time the vendor buys back the product in a reduced price, which means the product is included in an invoice.

Exercises

Case 4:

Products are first entered into our system when they are request for purchase is issued. A purchase order is then generated for the product. When the product is received, it is placed into inventory by recording the location where it is placed. When a customer buys a product it is included in an invoice. When an invoice is issued, the product is packed for shipping and the associated shipment is recorded. When the product is shipped, you need to record the shipper and the date it was picked up. Sometimes, a product is returned. In that case, you put the product back into inventory and record the location. Occasionally products are damaged in the inventory and are sold to the junkyard. If a product is lying in the inventory for more than one year, it is included in the yearly auction.

References

→ **Chapter 25**
The Unified Modeling Language User Guide
SECOND EDITION
By Grady Booch, James Rumbaugh, Ivar Jacobson

→ **Session 20, 21, 22**
UML Weekend Crash Course
Thomas A. Pender