

CHAPTER 11

Statechart Diagrams

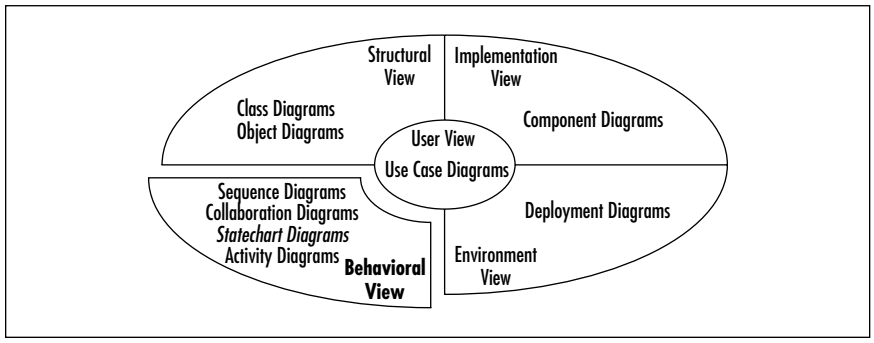


Figure 11-1: Behavioral Model View—Statechart Diagrams

Statechart (or state) diagrams (Figure 11-1) lie within the behavioral view of a system and render the states and responses of a class. These diagrams describe the behavior of a class in response to external stimuli. Statechart diagrams describe the life cycle of an object. These diagrams contain states and transitions. They may be constructed using the following technique:

1. Identify a class participating in behavior whose life cycle is to be specified.
2. Model states.
3. Model events.
4. Model transitions.
5. Refine and elaborate as required.

Statechart diagrams

- Are also known as state diagrams.
- Represent *state machines,* which are graphs of states and transitions that describe the responses of an object of a given class to outside stimuli. States

represent the condition of an object. Transitions specify how these conditions are related. Objects

  - Have one current state. The state of a composite object is the union of the states of all its parts.
  - Have an initial state when they are created.
  - May change states due to events.
  - May perform actions or activities within states.
  - May perform actions or activities when changing states.
  - Have final states before they are destroyed.

- May be associated with classes or methods. They describe the response of an object of a given class to outside stimuli.
- May have the various expressions (guard conditions, transitions, events, action clauses, send clauses, and actions) they utilize be expressed using pseudocode or another language.

States

States (Figure 11-2) are classes that define status conditions an object may satisfy during its existence. They are used to model the situations during the life of an entity in which it satisfies some condition, performs some activity, or waits for some occurrence. An entity remains in a state for a finite and noninstantaneous time.

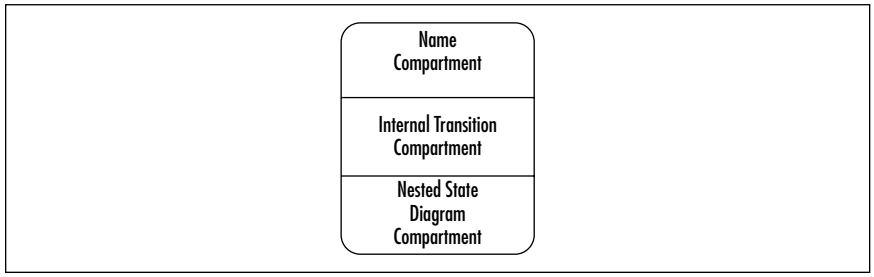


Figure 11-2: State

States

- Are denoted as rectangles with rounded corners.
- May be active or inactive. States become active when entered as a result of a transition firing. States become inactive when exited as a result of a transition firing.
- May be destinations of incoming transitions. These states function as a destination vertex for a transition.
- May be sources of outgoing transitions. These states function as a source vertex for a transition.

- May have a name compartment. This contains a state's name. States without names are distinct.
- May have an internal transition compartment named "internal transitions." These transitions specify actions or activities performed in response to events received while an object is in a particular state; however, they are performed without the object changing states. These transitions do not invoke the entry or exit actions of the state.
- May have one or more nested state machines. These are statechart diagrams within states, and can be shown as multiple statechart diagrams tiled in a single state. Each nested state machine corresponds to one ongoing activity within a state, and each state without nested state machines corresponds to one action. Actions are atomic executable statements that involve the invocation of an operation. Activities are nonatomic sequences of actions that have some duration. When nested state machines are used, the enclosing state is a composite state containing nested states. The nested states are substates of the composite state (or superstate). The nested states may be referenced by pathnames. The names are similar to class pathnames, but correspond to composite or containing state names.
- May have other compartments, which may be used for other model properties (business rules, responsibilities, variations, events, exceptions, etc.). These are not normally shown.
- May have their text compartments (name, internal transitions, and other compartments, not including the nested state diagram compartment) shrunk horizontally within the nested state compartment's graphic region (see Figures 11-3 and 11-4). This is only a convenience for appearance.

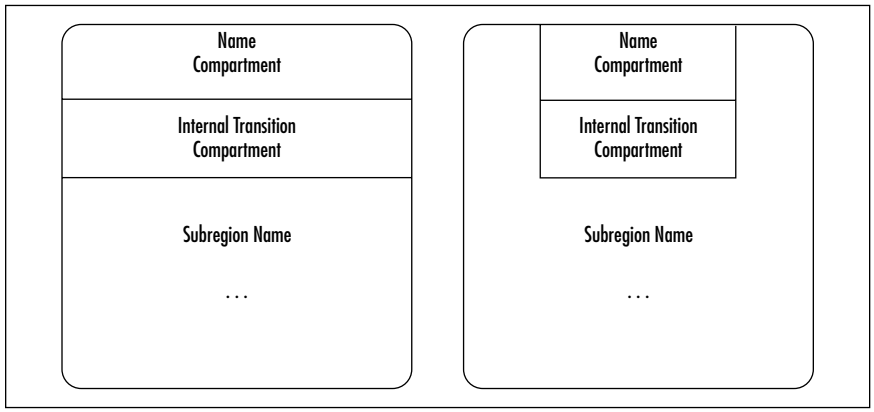


Figure 11-3: Disjoint Substates

- May be composite states having one nested state diagram (Figure 11-3). This corresponds to one activity within a state.
  - The subregion must include an initial state and a final state. A transition to the enclosing state represents a transition to the initial state. A transition to the final state within the subregion represents the completion of activity in the enclosing state. Completion of the outermost state of an

object corresponds to its death. An object that transitions to its outermost final state ceases to exist.

– The state is decomposed using an OR relationship into mutually exclusive disjoint substates. That is, there is only one subregion and there may be only one current or active substate.

– The subregion may have an optional name and must contain a nested state diagram.

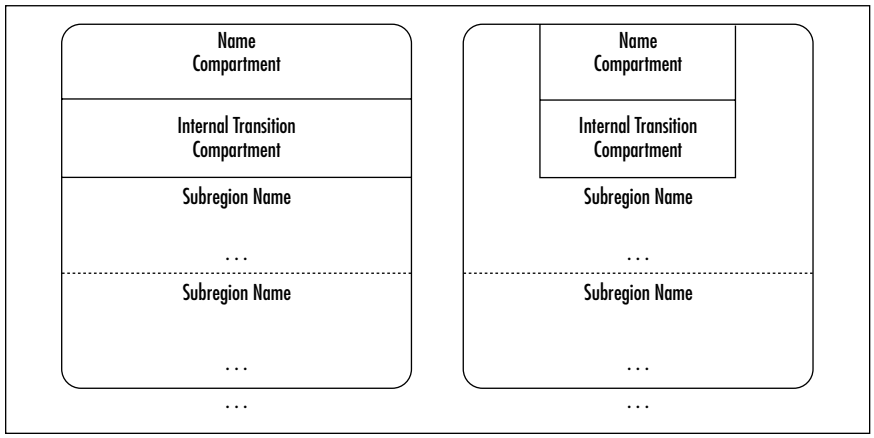


Figure 11-4: Concurrent Substates

- May be composite states having multiple state machines, each in a subregion (indicated by dashed lines) of the nested state diagram compartment (Figure 11-4). This corresponds to multiple activities within a state.
  – Each subregion must include an initial state and a final state. A transition to the enclosing state represents a transition to all initial states in all concurrent subregions. A transition to the final state within a subregion represents the completion of activity in the enclosing region. Completion of activity in all concurrent regions represents completion of activity by the enclosing state. Completion of the outermost state of an object corresponds to its death. An object that transitions to its outermost final state ceases to exist.
  – The state is decomposed using an AND relationship into mutually exclusive concurrent subregions with disjoint substates. That is, there are multiple subregions and each subregion may have its own current or active state. Each subregion has a current substate.
  – Subregions may have an optional name and must contain a nested state diagram with disjoint states. That is, subregions may not share states.
- May be simple states. These are states that do not have any substates.
- May be top-level states. These are the states owned by the state machine. Nested states are owned by their parent composite states. Top states
  – Are composite states. Each state machine has one composite state that contains all its states.

  - Must not be owned by any other state.
  - Must not be the source or target of a transition.
- May be pseudostates. Pseudostates
  - Are states that are only used as notational devices.
  - Represent transient points in transition paths between states.
  - Must not be the current state. They implicitly transition to an actual or non-pseudostate.

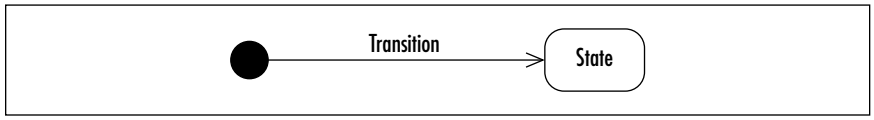


Figure 11-5: Initial (Pseudo) State

- May be initial pseudostates (Figure 11-5). Initial states
  - Represent any transition to the enclosing state. That is, any transition to the enclosing state would make the nested initial state the current state.
  - Are depicted as a small solid circle.
  - Must not have any incoming transitions.
  - Must have only one outgoing transition.
  - May be labeled with a transition string. This is the event that creates the object.
  - May be unlabeled. This represents any transition to the enclosing state.

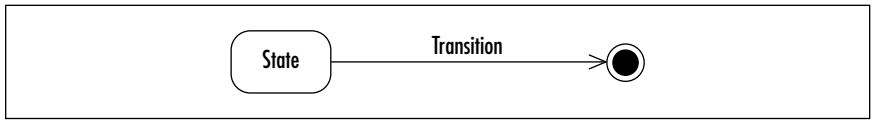


Figure 11-6: Final (Pseudo) State

- May be final pseudostates (Figure 11-6). Final states
  - Represent the completion of activity in an enclosing state.
  - Are depicted as a circle surrounding a small solid circle.
  - May have any number of incoming transitions.
  - Must not have any outgoing transitions.
  - May be labeled with a transition string.
- May be history pseudostates (Figure 11-7). History states (or indicators)
  - Represent the previous state within the state region that directly contains them. When a transition to the history indicator fires, the object resumes the state it last had within the region, and any entry actions are performed.
  - Are depicted as a small circle containing an “H.”

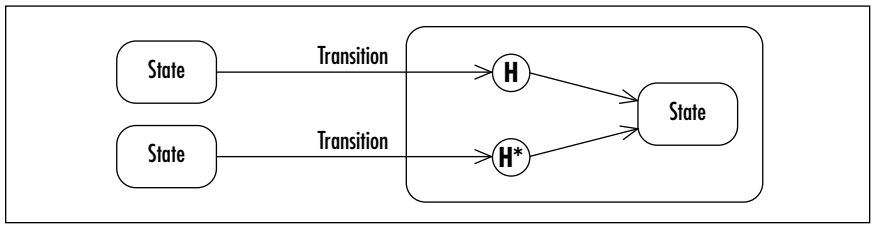


Figure 11-7: History (Pseudo) States

  - Apply to the state region that directly contains them.
  - Must not be used in the top state of a state machine.
  - May have any number of incoming transitions from outside states.
  - May have at most one outgoing unlabeled transition. This is the default "previous state" if the region has never been entered.
  - May be deep history indicators, specified by a small circle containing an "H*." This indicates that the object reuses the state it last had at any depth within the complex state region rather than being restricted to a state at the same level as the history indicator.
- May have generalization relationships with other states. Generalization relationships between states may have one of the following interpretations:
  - Using subtyping, specialized states have the same outgoing transitions, may add outgoing transitions, have a different set of incoming transitions, and may add actions.
  - Using subclassing, specialized states have some of the same incoming transitions (may add or drop some), add outgoing transitions, and may add actions.
  - Generally, specialized states may have different outgoing and incoming transitions and different actions. They add and delete transitions and actions.

Transitions

Transitions (Figure 11-8) are associations between states. They are used to model relationships between the different states of an entity. An entity in one state will perform actions and possibly enter another state when an event occurs and a condition is satisfied (if specified); this occurrence is known as the *firing* of a transition. Transitions

- Are denoted as solid arrows between states and are labeled with a transition string.
- Relate source state vertices and target state vertices.
- May be internal state transitions. These are actions or activities performed in response to events received while an object is in a particular state; however, they are performed without the object changing states. Such transitions do not invoke the entry or exit actions of the state.

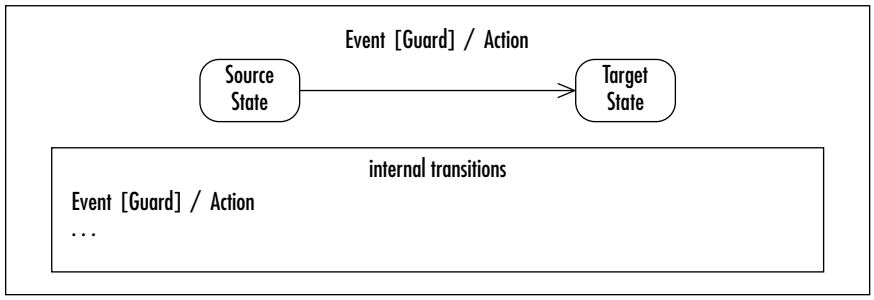


Figure 11-8: Transitions

- May be self-transitions. These are transitions in which the source state and target state are the same state. Such transitions do invoke the entry and exit actions of the state.
- May have an event expression that indicates the event that causes or triggers the transition to fire. The parameters of the event are available to the action on the transition and actions in the subsequent state.

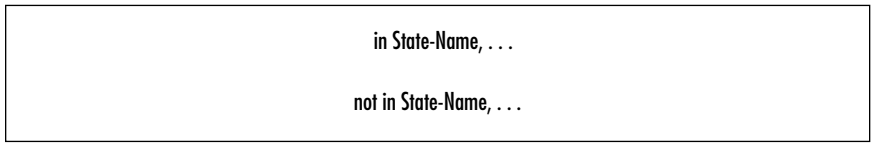


Figure 11-9: Guard Conditions

- May have square brackets containing a guard condition. Guard conditions
  - Are Boolean conditions that must be satisfied in order to enable the transition to fire.
  - May refer to the parameters of the triggering event and to attributes and links of the object that owns the state machine.
  - May involve tests of concurrent states of the current state machine. This includes expressions such as “in State-Name” to test whether the machine is in a state, and expressions such as “not in State-Name” to test whether the machine is not in a state (Figure 11-9).
- May have a forward slash followed by an action expression or action sequence that indicates the actions or effects that result when the transition fires.
- May be completion transitions (or automatic transitions). Every state may have one transition without an event trigger. The transition is automatically triggered upon the completion of the actions within the state.
- May conflict with one another. This occurs when multiple transitions are simultaneously enabled. Priorities are used to resolve the conflict. Transitions of substates have higher priority than transitions from any containing states. Otherwise, priorities must be explicitly specified. These priorities are useful for real-time applications.

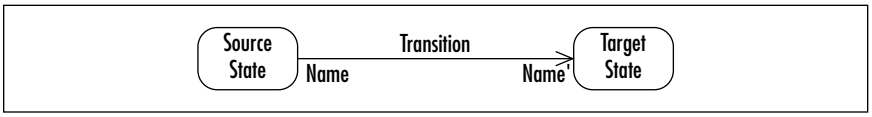


Figure 11-10: Sending and Receiving Names

- May have sending times and receiving times (Figure 11-10). These are formal names that may be used in constraint expressions.
  - The transition is given a name that represents the time at which it fires or is triggered.
  - The transition name with a prime sign appended to it represents the time at which the target state is reached via the transition.
  - These formal names are useful for real-time applications.
- May be compound transitions. These are transitions that consist of a cluster of simple transitions. The simple transitions are clustered using branching and complex transitions (fork and join transitions).

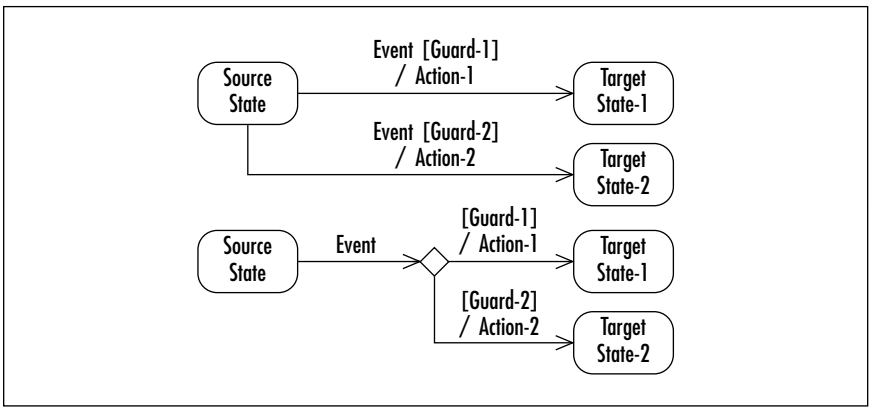


Figure 11-11: Branching (Decisions)

- May be part of a branching, or decision, construct (Figures 11-11 and 11-12). A decision utilizes guard conditions to indicate different possible transitions. Decisions
  - Are shown by labeling multiple outgoing transitions of a state with different guard conditions.
  - May use the diamond shape with one or more incoming transitions and two or more outgoing transitions. The incoming transitions to the diamond contain an event trigger. The outgoing transitions from the diamond must have distinct guard conditions with no event triggers. All possible outcomes of the decision should appear on one of the transitions, such that only one path may be selected as a result of the decision.
  - May be chained as part of complex transitions. However, only the first incoming transition may contain an event expression. All outgoing transitions may have guard conditions and action expressions.

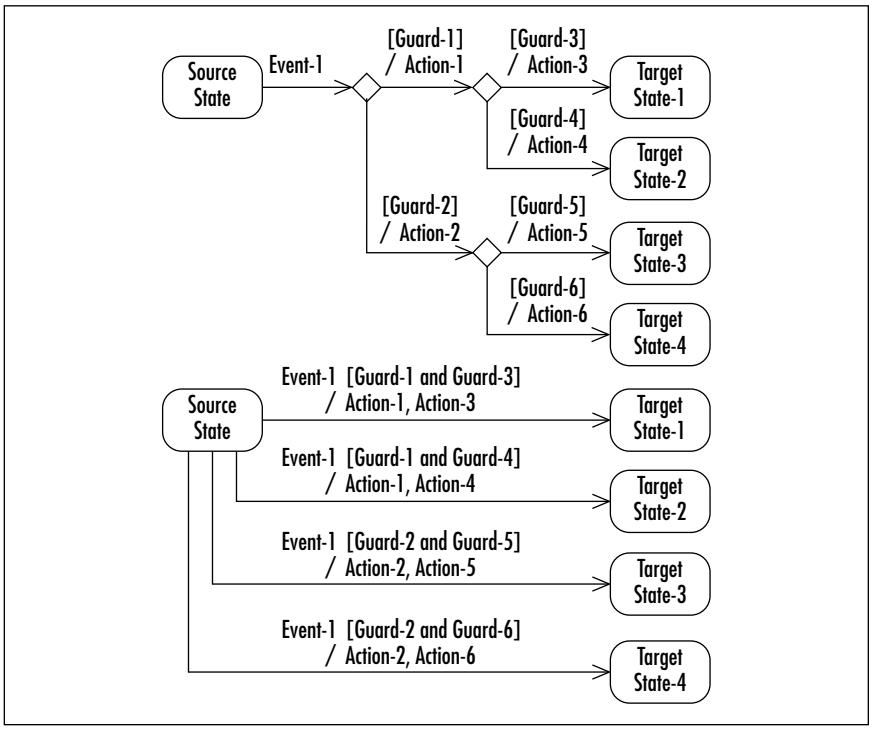


Figure 11-12: Chained and Unchained Branching (Decisions)

– May be unchained. This results in an individual transition for each path through the decision construct. The guard condition on the transition is the AND of all the conditions along the path.

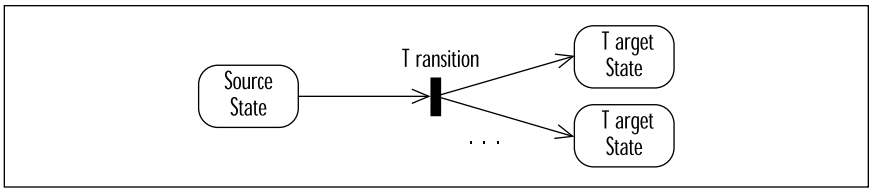


Figure 11-13: Splitting of Control

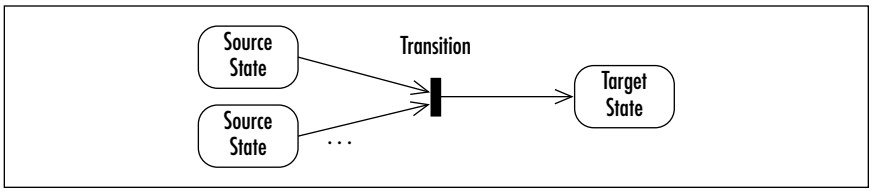


Figure 11-14: Synchronization of Control

- May be complex transitions (Figures 11-13 and 11-14). Complex transitions
  - Are transitions that involve multiple source states and multiple target states.
  - Are depicted as a short heavy bar.
  - Are enabled when all the source states are occupied. After the transition fires, all its destination states are occupied or activated concurrently.
  - May have multiple target states, indicating a splitting of control into concurrent threads without concurrent substates. The short heavy bar represents the forking of threads. This is a fork (pseudo) state. Fork states must have at least two outgoing transitions and one transition expression for the fork, and must originate from a state.
  - May have multiple source states, indicating a synchronization of control of concurrent threads. The short heavy bar represents the synchronization of threads. This is a join (pseudo) state. Join states must have at least two incoming transitions and one transition expression for the join, and must target a state.
  - May have a transition string shown near the bar. Individual arrows do not have their own transition strings.
- May point to complex states. This kind of transition is equivalent to transitions to a complex state's enclosed initial state or to initial states of each concurrent subregion.
- May originate from complex states. These transitions apply to each of the states within a complex state at any depth. That is, transitions are inherited by nested states. They may be masked or hidden (overridden) by the presence of nested transitions with the same trigger event. If a complex state's transition fires, any nested states are forcibly terminated and perform their exit actions before the transition action occurs and the new state is established.
- May be drawn directly to states within a complex state region at any nesting depth. All entry actions are performed for any states that are entered on such a transition. If a fork bar is used, the resulting concurrent states are established, and any other concurrent subregions start with their default initial states.
- May be drawn directly from states within a complex state region at any nesting depth to outside states. All exit actions are performed for any states that are exited on such a transition. If a join bar is used, specific states in other regions are irrelevant to triggering the transition.
- May be stubbed transitions (Figure 11-15). Stubbed transitions
  - Are transitions drawn to the most specific accessible enclosing state of a suppressed state. The contents of the suppressed state are not fully depicted.
  - Are depicted using stubs, which are small vertical lines drawn inside the boundary of the enclosing state.
  - May be drawn to stubs if they do not go to an unlabeled initial state. Stubs are not used for transitions to initial states.

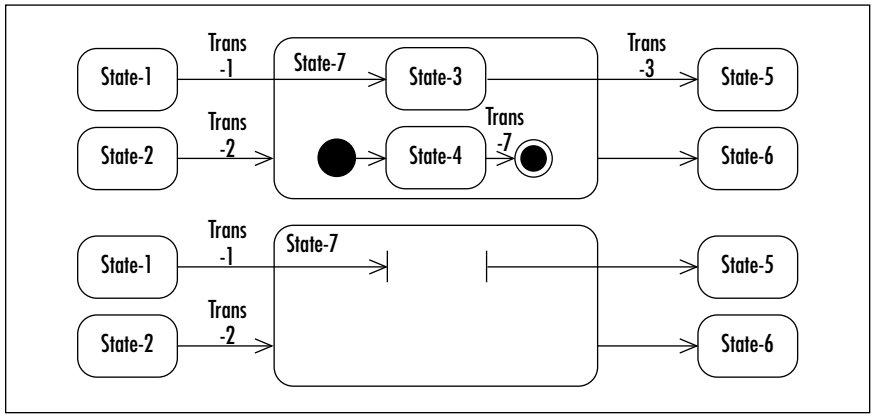


Figure 11-15: Stubbed Transitions

  - May be drawn from stubs if they do not come from an unlabeled final state. Stubs are not used for transitions from final states.
  - May have events depicted when leading into a state. These transitions lead either to a state or to an internal substate.
  - Must not have events shown when leading from a suppressed state to an external state. The transition information belongs to its source state; if the source state is suppressed, so are the details of outgoing transitions. Final state transitions are depicted as unlabeled transitions from the border of a complex state. This represents the implicit event "action complete" (or "completion of activity" event) for the corresponding state.
- May have generalization relationships with other transitions. Generalization relationships between transitions may have one of the following interpretations:
  - Using subtyping, specialized transitions may go to new target states and may add negative guard conditions. The guard conditions negate existing guard conditions.
  - Using subclassing, specialized transitions may go to new target states. They must have the same source states, and may have different guard conditions.
  - Generally, specialized transitions may have different source states and target states, and different guard conditions. They add and delete transitions and guard conditions.

Events

Events (Figure 11-16) are classes that define occurrences that may trigger an object to change states. They are used to model significant occurrences or stimuli that affect an entity. Events

- Are denoted as strings.

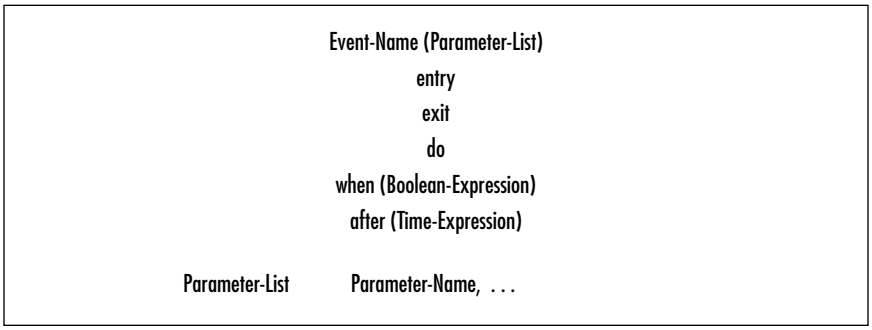


Figure 11-16: Events

- Must have a name or identifier string that represents the name of the event. The event name must reference an operation or signal in the class receiving the event.
- May have parentheses containing a comma-separated parameter list that indicates the formal parameters passed to the event.
- May be used with internal state transitions. For internal state transitions, the following events are predefined and cannot be used for user-defined events:
  - Entry events indicate an entry action and are specified using the "entry" keyword. This is an action sequence that is executed when the state is entered. The action is atomic, may not be avoided, and precedes any internal activities or other transitions. Entry events may not have any parameters or guard conditions because they are implicitly rather than explicitly invoked. However, the entry event at the top level of a state machine for a class may have a parameter list that corresponds to the arguments an object of that class receives when it is created. This event does not occur for internal state transitions since there is no state change. This event does occur for self-transitions since there is a state change.
  - Exit events indicate an exit action and are specified using the "exit" keyword. This is an action sequence that is executed when the state is exited. The action is atomic, may not be avoided, and follows any internal activities, but precedes any outgoing transitions. Exit events may not have any arguments or guard conditions because they are implicitly rather than explicitly invoked. This event does not occur for internal state transitions since there is no state change. This event does occur for self-transitions since there is a state change.
  - Do events indicate an invocation of a nested state machine and are specified using the "do" keyword. The nested state machine must have an initial and final state. The nested state machine begins with its initial state after any entry action. When the nested state machine reaches its final state, any exit action occurs. The current state may then fire a transition based on the implicit completion of activity. Activities associated with do events must reference nested state machines rather than operations of the containing object.
- Must appear at most once in a single state.

- May be signal events. Signal events
  - Represent the reception of a request to invoke an operation. The request is issued using a signal action.
  - Are caused by the asynchronous invocation of an operation. The receiver class must declare the operation to be a signal using the "signal" stereotype keyword.
  - Have a name and a parameter list. They are declared as operations on class diagrams.
  - May have generalization relationships with other signal events. This indicates that an occurrence of the subevent triggers any transition dependent on the event or any of its ancestors.
  - May be declared on class diagrams as classes stereotyped using the "signal" keyword. These are signal names that may be used to trigger transitions. These classes must have their parameters shown in the attribute compartment. They must not have any operations. They must not have any relationships to other classes.

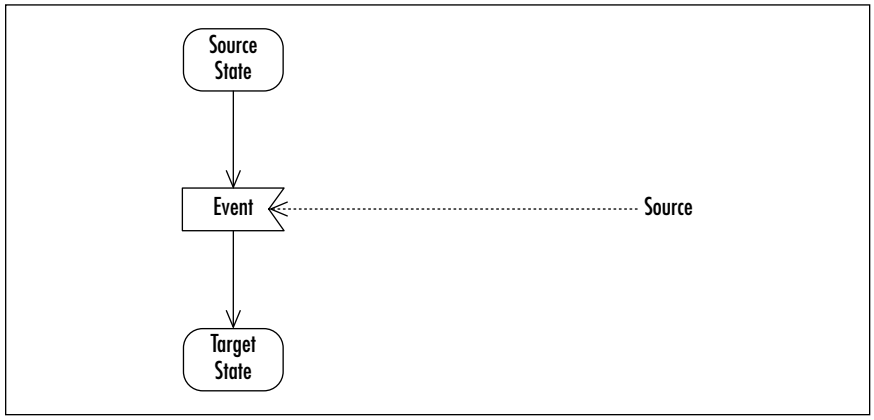


Figure 11-17: Signal Receipt Stereotype Icon

  - May use the signal receipt stereotype icon (Figure 11-17). This is a concave pentagon (or rectangle with a notch) representing the receipt of a signal. The event is indicated inside the symbol. An unlabeled transition is drawn from the previous state to the pentagon. An unlabeled transition is drawn from the pentagon to the next state. A dashed arrow may be drawn from the sender object to the notch on the pentagon. This symbol replaces the event label on a transition.
- May be call events. Call events
  - Occur when an operation is involved. The caller requests the operation using a call action.
  - Are caused by the synchronous or asynchronous invocation of an operation. The caller determines whether to wait for the operation to complete. If the caller waits for the operation to complete, the operation is invoked

synchronously. If the caller does not wait for the operation to complete, the operation is invoked asynchronously.

  - Have a name and a parameter list. They are declared as operations on class diagrams.

- May be change events. Change events
  - Represent a notification that a condition has become true.
  - Are specified using the "when" keyword.
  - Must have a Boolean expression enclosed in parentheses designating the condition that must become true in order for the transition to fire.
  - Are different from guard conditions. Guard conditions are evaluated when the event fires. Change events fire when a condition becomes true; they are continuously checked when an object changes.
- May be time events. Time events
  - Represent a notification that a period of time has elapsed since entering a state or that a particular date/time has been reached.
  - Are specified using the "after" keyword.
  - Must have a time expression enclosed in parentheses designating the time that must elapse or the date/time that must be reached in order for the transition to fire.
- May occur without triggering any transition. The event is simply ignored or lost if it is not deferred (see the section entitled "Actions").
- May trigger more than one transition within the same region. Only one transition will fire, and the choice may be nondeterministic if a firing priority is not specified.
- May be applied to an initial transition if the event is stereotyped using the "create" keyword.

Actions

Actions (Figure 11-18) are classes that define executable statements or computational procedures. Actions are atomic (non-interruptible) computational procedures, typically involving invocation of an operation. Activities are nonatomic (interruptible) and consist of a sequence of steps or actions that have some duration. Activities may be expressed using nested statechart diagrams. Actions and activities are used to model the responses of an entity. Actions

- May be used to defer an event. This uses the "defer" keyword. Normally, an event that is ignored or not handled immediately is lost. Using the deferred action, a state may specify a set of events that are deferred or postponed if they occur while the state is active performing some activity. Deferred events are saved and recur when the object transitions to another state. All other events not included in the set of deferred events are lost. However, after the state completes its activity, the deferred events may trigger a transition or may remain deferred until the next state is reached. In the new state, the events

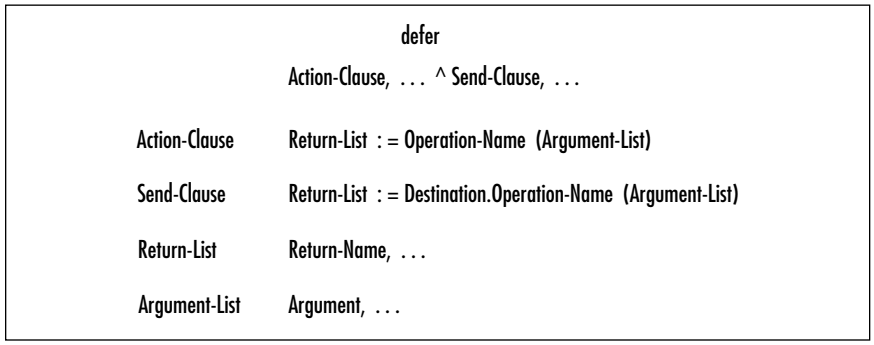


Figure 11-18: Action

may be accepted, deferred again, or lost. Events deferred within a composite state are also deferred for the nested states of the composite state.

- May have an action expression consisting of a comma-separated list of action clauses. Action clauses
  - Are call actions. They result in the invocation of an operation.
  - Are executed when the transition fires.
  - Are executed from right to left.
  - May have a return list consisting of a comma-separated list of names that designate the values returned by the operation. These values may be used as actual parameters in subsequent operations. The return names must match the order, number, and types specified in the formal return list of the operation. When the return list is omitted, the assignment operator (:=) is omitted.
  - Must have a name or identifier string that represents the name of the operation. The operation must be in the class receiving the message.
  - May have parentheses containing an argument list consisting of a comma-separated list of actual parameters passed to a method. The actual parameters must match the order, number, and types specified in the formal parameter list of the operation.
  - May refer to the parameters of the triggering event and to the operations, attributes, and links of the object that owns the state machine.
  - Must be atomic operations. That is, they may execute without interruption and must completely execute before any other actions or transitions are considered.
  - May be used with internal entry and exit state transitions. The action may refer to the attributes and links of the owning object, and to parameters of incoming transitions if the parameters appear on all incoming transitions.

  - May be used with internal do state transitions. In this case, the action must reference a nested state machine. The operation name corresponds to the name of the nested state machine.

- May have a "^" symbol followed by a comma-separated list of send clauses. Send clauses
  - Are send actions. They result in the asynchronous sending of a signal.
  - Are executed when the transition fires.
  - Are executed from right to left after the action clauses.
  - May have a return list consisting of a comma-separated list of names that designate the values returned by the operation. These values may be used as actual parameters in subsequent operations. The return names must match the order, number, and types specified in the formal return list of the operation. When the return list is omitted, the assignment operator (:=) is omitted.
  - May have a destination string that indicates the object or class that will receive the message.
  - Must have a name or identifier string that represents the name of an operation or signal. The operation must be in the class receiving the message.
  - May have parentheses containing an argument list consisting of a comma-separated list of actual parameters passed to a method. The actual parameters must match the order, number, and types specified in the formal parameter list of the operation.
  - May refer to the parameters of the triggering event and to the attributes and links of the object that owns the state machine.
  - Must be atomic operations. That is, they may execute without interruption and must completely execute before any other actions or transitions are considered.

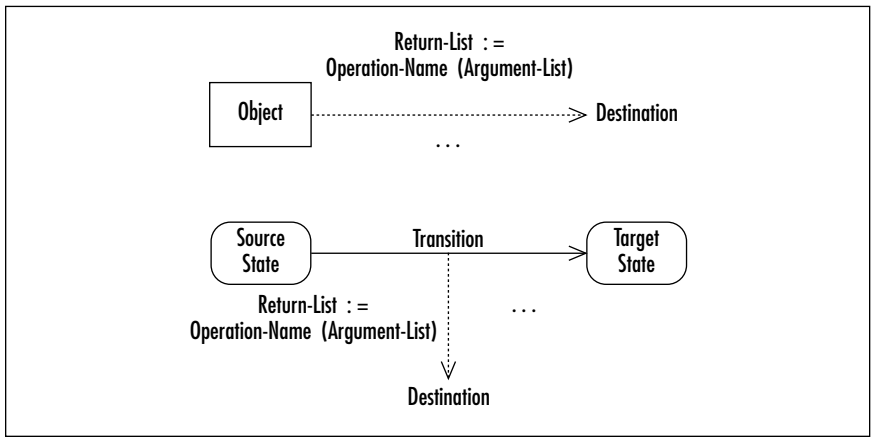


Figure 11-19: Send Clauses

– May be depicted graphically (Figure 11-19). The sender may be a transition or an object, and the receiver may be a transition, an object, or a class. This requires each state diagram to be nested physically within an object symbol. There may be one object enclosing the main state diagram. This is the object that owns the diagram. The message is shown as a dashed arrow from the sender to the receiver labeled with the send clause, but the destination expression is omitted from the send clause since it is indicated by the dashed arrow. If the sender is a transition, the message is sent when the transition fires. If the sender is an object, the message is sent by the object at some point in its life. The actual point is unspecified. If the receiver is a transition, the transition must be the only transition in the object involving the event or be the only transition that would be triggered in response to the message. This notation may not be used if the transition triggered depends on the state of the receiving object and not just the state of the sender. If the receiver is an object, the message may trigger a transition on the corresponding event. There may be many transitions involving the event. This notation may not be used when the target object is computed dynamically. If the receiver is a class, the message may invoke a class scope operation. This notation may be used in other diagrams to show the sending of a message between objects and classes.

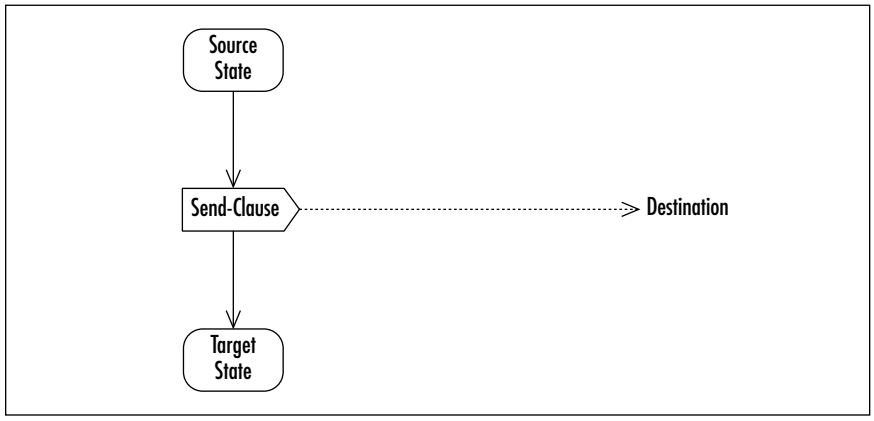


Figure 11-20: Signal-Sending Stereotype Icon

– May use the signal-sending stereotype icon (Figure 11-20). This is a convex pentagon (or rectangle with a point) representing the sending of a signal. The send clause is indicated inside the symbol, but the destination expression is omitted from the send clause since it is indicated by the dashed arrow. An unlabeled transition is drawn from the previous state to the pentagon. An unlabeled transition is drawn from the pentagon to the next state. A dashed arrow may be drawn from the point of the pentagon to the receiver of the signal. This symbol replaces the send clause on a transition.

- May be create actions. Create actions create an instance of a class. They must not have any target object.
- May be destroy actions. Destroy actions destroy an instance of a class. They must have a target object.
- May be return actions. Return actions return a value or set of values to the caller.
- May be terminate actions. Terminate actions cause the self-destruction of objects. They must not have any arguments.
- May be local invocation actions. Local invocation actions invoke a local operation without generating a call event or signal event.
- May be uninterpreted actions. Uninterpreted actions are actions that are not predefined.
- May be exception actions. Exception actions raise a signal in the case of an error during execution. The sender aborts execution, and execution resumes with the receiver of the exception. The receiver is determined implicitly by the interaction sequence during execution and is not explicitly specified.