EVENTS AND SIGNALS

Events

- An event is the specification of a significant occurrence that has a location in time and space.
- Any thing that happens is modeled as an event in UML.
- In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition
- four kinds of events – signals, calls, the passing of time, and a change in state.
- Figure 1: Events
- Events may be external or internal and asynchronous or synchronous.
  asynchronous events are events that can happen at arbitrary times eg:- signal, the passing of time, and a change of state. *synchronous events*, represents the invocation of an operation eg:- Calls
  External events are those that pass between the system and its actors. *Internal events* are those that pass among the objects that live inside the system.
  A signal is an event that represents the specification of an asynchronous stimulus communicated between instances.

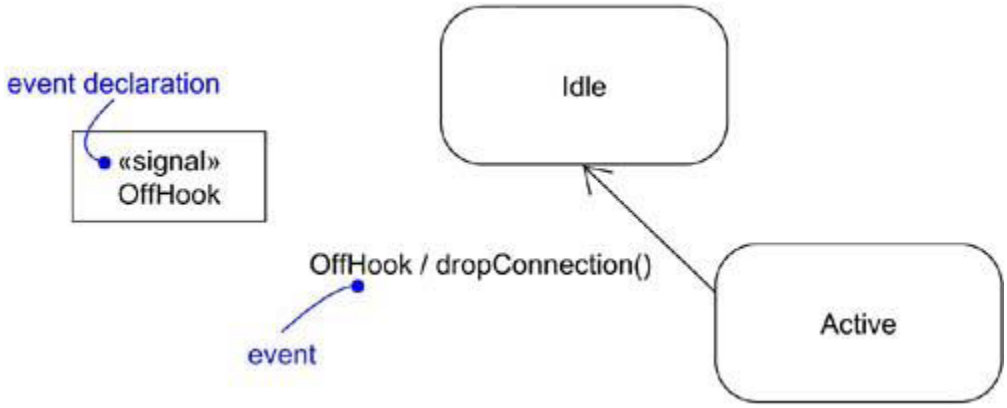

Figure 1: Events

kinds of events

Signal Event

- a signal event represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another. Exceptions are an example of internal signal
- a signal event is an asynchronous event
- signal events may have instances, generalization relationships, attributes and operations. Attributes of a signal serve as its parameters
- A signal event may be sent as the action of a state transition in a state machine or the sending of a message in an interaction
- signals are modeled as stereotyped classes and the relationship between an operation and the events by using a dependency relationship, stereotyped as send
- Figure 2 shows Signal Events

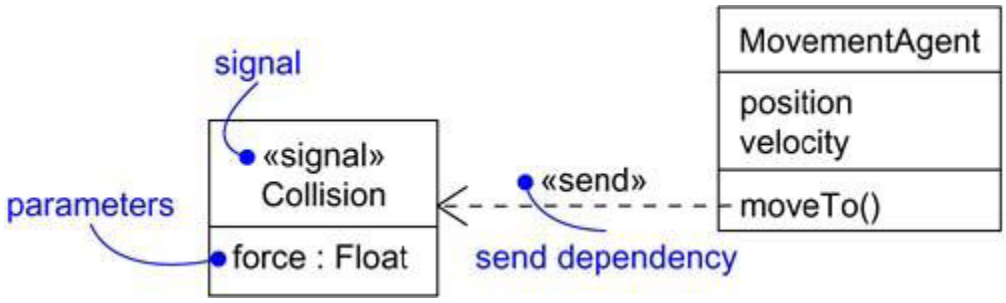

Figure 2: Signals

Call Event

- a call event represents the dispatch of an operation
- a call event is a synchronous event
- Figure 3 shows Call Events

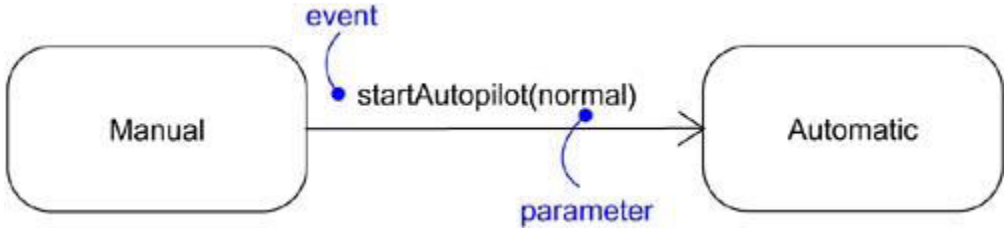

Figure 3: Call Events

Time and Change Events

- A *time event* is an event that represents the passage of time.

- modeled by using the keyword ‘after’ followed by some expression that evaluates to a period of time which can be simple or complex.
- A *change event* is an event that represents a change in state or the satisfaction of some condition
- modeled by using the keyword ‘when’ followed by some Boolean expression
- Figure 4

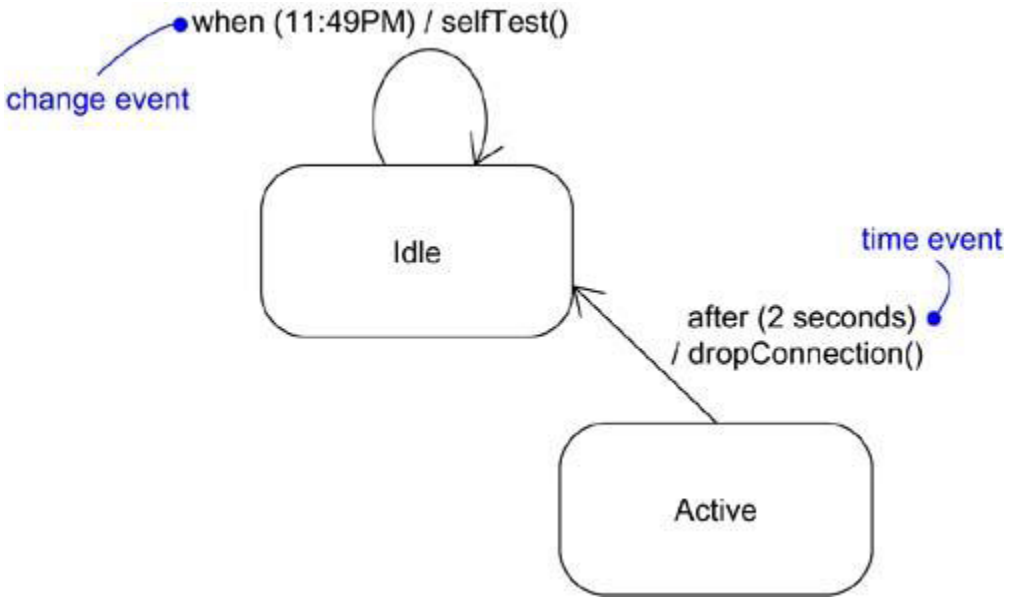

Figure 4: Time and Change Events

Sending and Receiving Events

For synchronous events (Sending or Receiving) like call event, the sender and the receiver are in a rendezvous(the sender dispatches the signal and wait for a response from the receiver) for the duration of the operation. when an object calls an operation, the sender dispatches the operation and then waits for the receiver.

For asynchronous events (Sending or Receiving) like signal event, the sender and receiver do not rendezvous ie,the sender dispatches the signal but does not wait for a response from the receiver. When an object sends a signal, the sender dispatches the signal and then continues along its flow of control, not waiting for any return from the receiver.

call events can be modelled as operations on the class of the object.

named signals can be modelled by naming them in an extra compartment of the class as in Figure 5: Signals and Active Classes

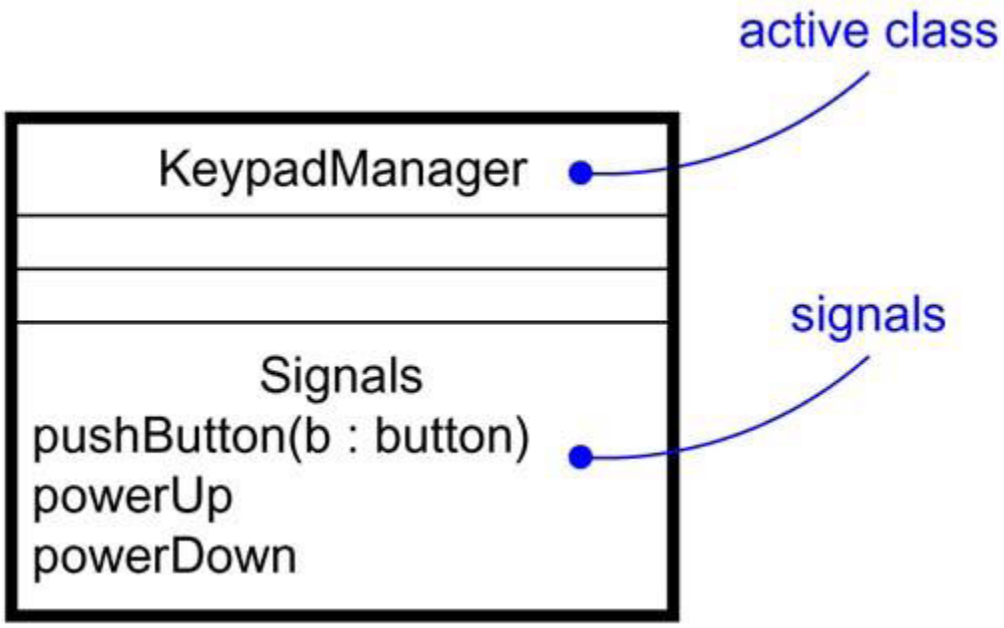

Figure 5: Signals and Active Classes

Modeling family of signals

To model a family of signals,

- Consider all the different kinds of signals to which a given set of active objects may respond.
- Look for the common kinds of signals and place them in a generalization/specialization hierarchy using inheritance. Elevate more general ones and lower more specialized ones.
- Look for the opportunity for polymorphism in the state machines of these active objects. Where you find polymorphism, adjust the hierarchy as necessary by introducing intermediate abstract signals.

  Figure 6 models a family of signals that may be handled by an autonomous robot.

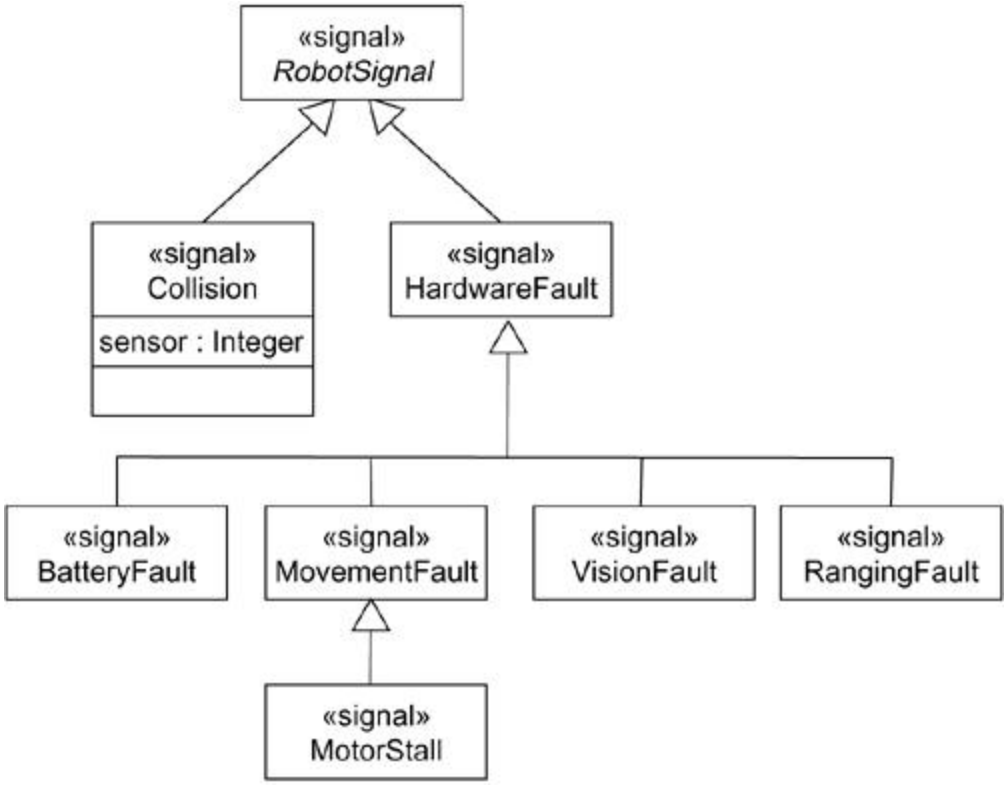

Figure 6: Modeling Families of Signals

Modeling Exceptions

To model exceptions,

- For each class and interface, and for each operation of such elements, consider the exceptional conditions that may be raised.
- Arrange these exceptions in a hierarchy. Elevate general ones, lower specialized ones, and introduce intermediate exceptions, as necessary.
- For each operation, specify the exceptions that it may raise. You can do so explicitly (by showing send dependencies from an operation to its exceptions) or you can put this in the operation's specification.

  Figure 7 models a hierarchy of exceptions

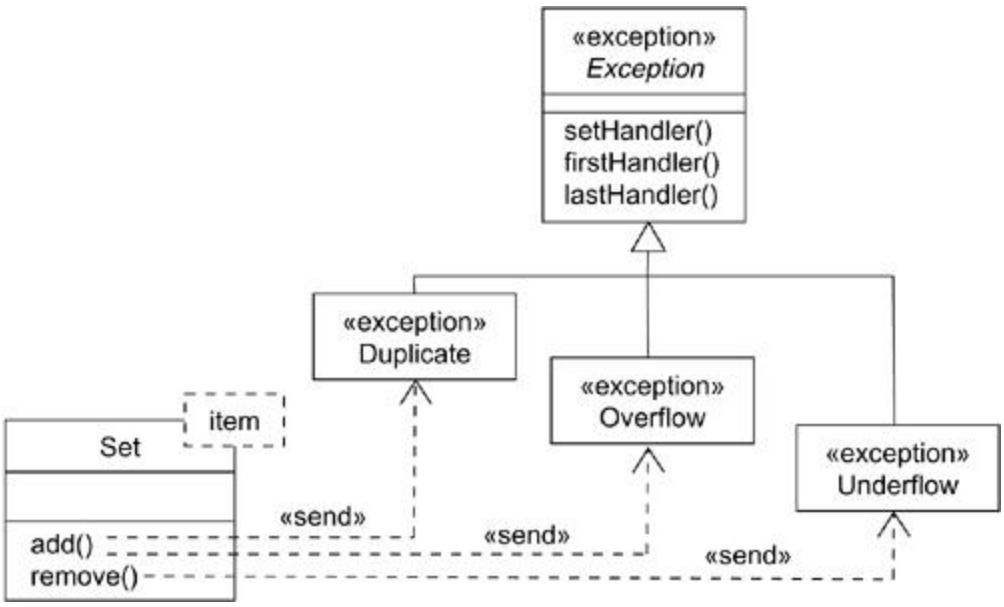

Figure 7: Modeling Exceptions

Source : http://praveenthomasln.wordpress.com/2012/04/06/events-and-signals-s8-cs/