

Statechart components (Cont'd)

- Event: A time discrete occurrence of interest
- Guard: Conditions that must be met for a transition to be taken
- Action: Quick operation executed in practically no time establishing a relevant change or result



Execution of actions

Actions can be executed when:

- A state is entered. Specified with **entry:** in the state container
- A transition is made. Specified besides the transition
- A state is left. Specified with **exit:** in the state container.



Specification of transitions

Labelsyntax: eventname [guard] / actionlist

eventname Name of the event triggering the transition

guard Boolean expression that must evaluate to **TRUE** for the transition to be taken

actionlist Dot-comma separated list of actions executed as a result of the transtion taken



Initial and terminal pseudo states

- Are almost, but not quite, states. States with a transient character (i.e. not triggered by event).
- We know the following pseudo states:
 - Initial state:** Indicates the initial or default state. The associated transition to must be made unconditionally (no guard). Drawn as filled dot. ●
 - Terminal state:** Indicates destruction of the object (at the outermost level). Drawn as encircled dot. ⊙
 - Choice state:** Selects the next state dependent on some condition. Used to simplify the diagram by reducing the number of outgoing transitions of a state. ◇

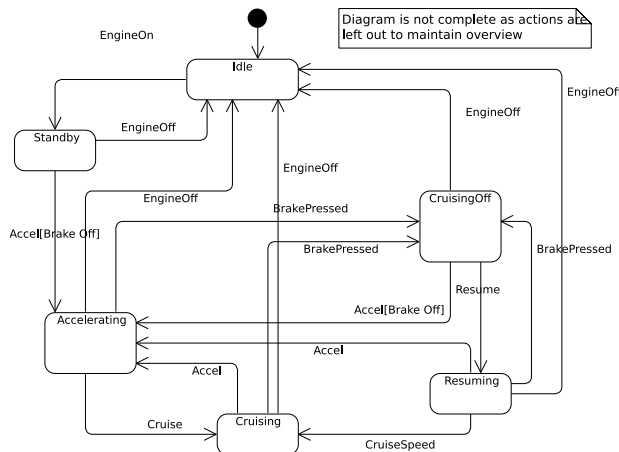


Composed states

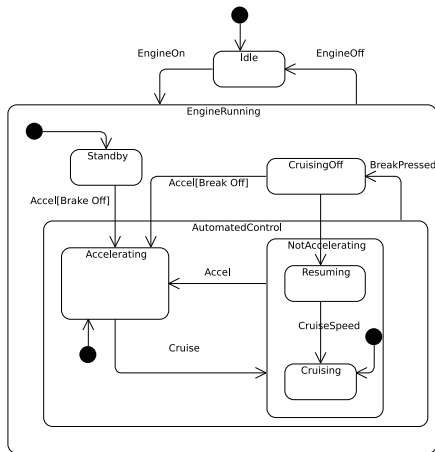
- A Super state or container state may be composed of two or more (sequential) sub states
- If state machine reaches super state S, consisting of two (sequential) substates S11 and S12, then it reaches either sub state S11 **OR** sub state S12.
- If a state is not further composed of other states it is called a leaf state.
- Group transition: A transition connected to the edge of a composed state (or: non leave state).
 - Incoming: The sub state entered is determined by initial (or: history if present) pseudo state
 - Leaving: The last actual sub state is stored in the history connector (if present at all)



flat std example



Hierarchical std example



History pseudo state

- First time, if no history exists, the connected state is initially entered (like the initial pseudo state).
- Later on the last occupied sub state is entered. Two variants:
 - Shallow history: Applied to the newly entered state only
 - Deep history (with star): Applied to the newly entered state and all its child states



State machine operation

- In a state machine the state changes only due to an event occurrence
- If an event occurs in a state where an leaving transition exists with that particular event name next to it, the transition is triggered. A triggered transition is taken only:
 - when no guard is present
 - if a guard is present and the guard conditions are met.
- When the transition is taken a chain of actions is executed: The actions from action lists associated with the states left (inside out), the transition and the states entered (outside in).
- Also when the transition is taken the state is changed unless it is a self transition. In that case the state remains the same, but some action can be taken.



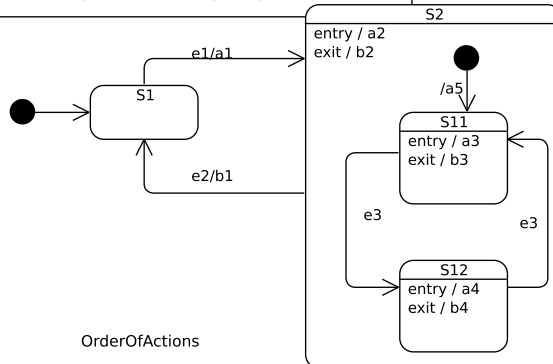
UML state representation

State chart syntax
Transition notation
Composite states

Order of actions

Order of actions:

e1 action chain (from state S2): a1;a2;a5;a3
e2 action chain (from State s1.x): b3 (or: b4); b2; b1



OrderOfActions



UML state representation

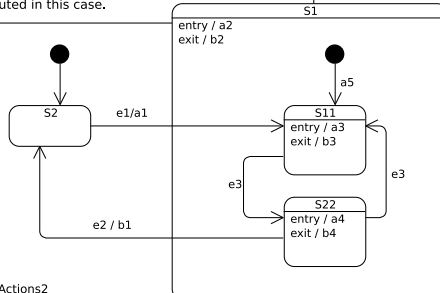
State chart syntax
Transition notation
Composite states

Order of actions with direct to inner state

With a direct transition to an inner state, the initial state is bypassed.

Order of actions:

e1 action chain: a1;a2;a3
e2 action chain: b4;b2
Note that a5 is not carried out because the initial state is not executed in this case.



OrderOfActions2



UML state representation

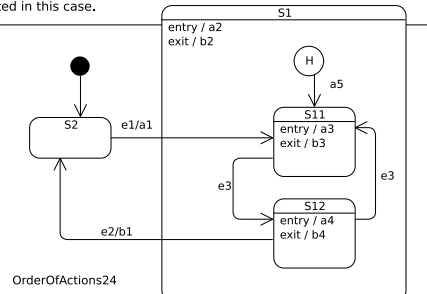
State chart syntax
Transition notation
Composite states

Order of actions with direct to inner state(2)

This also applies to diagrams with history states.

Order of actions:

e1 action chain: a1;a2;a3
e2 action chain: b4;b2
Note that a5 is not carried out because the initial state is not executed in this case.



OrderOfActions24



Quiz: Criticize this diagram.

Recommendations

- **Wrong**
 - Unlabeled transitions (no event)
 - Guard on initial/history action list
 - State with
 - only incoming transitions
 - only transitions leaving
 - more transitions leaving responding to the same event with no or non mutual exclusive guards
 - Overlapping guard conditions
- **Better not**
 - Apply an entry action list and als an initial action list
 - Add an action list to a history state
 - Use the deep history state (it is not clear on lower levels)



More UML Modeling

Besides statecharts we will use:

- Use Case Diagram/descriptions
- Class Diagram
- Sequence Diagram

A consistent and coherent set of diagrams and descriptions together form a model of your system. This model helps you to make sure you understand the problems to be solved and the needs to be fulfilled and to make the step towards the implementation smaller.



Use case diagram/description

- Use case
 - Depicted by an oval. Each use case has an associated description. Described is the interaction of an actor with the system to obtain some result from the system or to establish a change to the system without revealing the systems internal structure.
- Actor: Participant from outside the system
- The <<uses>> Relation (default) <<extend>> Relation <<includes>> Relation



Use case description

A use case is described in a table, showing:

- Name (as in diagram!)
- Summary
- Dependency (extend/include relations)
- Actors
- Preconditions (assumptions)
- Description
- Alternatives (another or exceptional sequence)
- Post-conditions (Result)



Class diagram

- Classes
- Attributes
- Operations (or: methods)
- Relations
 - Associations (bidirectional, unidirectional)
 - Composition and aggregation (whole-part)
 - Inheritance (Generalization/specialization)
 - Realization (Inheritance of abstract classes) Dependency
- Multiplicity (cardinality)



Aggregation and composition

- Whole part relationship
- The **aggregation** relationship expresses a collection (e.g. books in library) where as the **composition** expresses a more rigid relation (e.g. a cow has four legs)
- Major difference is the lifetime coupling
 - Aggregation Parts may be constructed and/or destructed during the lifetime of the whole object
 - Composition Parts are constructed as part of the construction of the whole and destructed as part of destruction of the whole object



Aggregation and composition (cont'd)

- Composition implementation:
 - Part is a member variable of whole
 - Part is instantiated in constructor of whole and discarded in the destructor of the whole object.
- Aggregation implementation:
 - Part may be instantiated or discarded from any method or operation of the whole object as required. However all existing parts are discarded from the destructor of the whole object



Aggregation and composition (Cont'd)

- The cardinality of a composition relation should be fixed since the lifetime is tightly coupled to the whole lifetime. If cardinality is omitted, it defaults to one.
- The cardinality of an aggregation should not be fixed. The number of parts may change during the lifetime of the whole object. If cardinality is omitted it defaults to 0...*



Sequence/collaboration diagram

- Shows dynamic interaction between objects (class instances), ordered in a time sequence
- The sequence is usually initiated by an actor and eventually brings a result back to the actor
- Only the cluster of objects involved is shown and together have to establish a specific task
- Several sequences may be required to show different scenarios (e.g. in case of an error) for a specific task
- The collaboration diagram is equivalent to a sequence diagram, it shows another representation of the same interaction



Diagram relations (some examples)

- An incoming arrow on an class instance in the sequence diagram implies that:
 - That class exists on the class diagram
 - That class on the class diagram shows an operation with the same name and signature
 - That class is associated with the class instance on the other end of the arrow
- A statechart specifies the state behaviour of a class, therefore it must always be clear to what class a statechart belongs

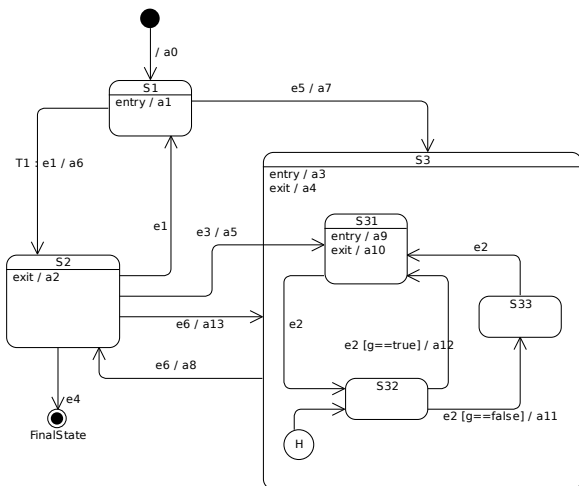


Implementing model diagrams

- If syntactical and semantic rules are clear, diagrams can be implemented
- If rules are more formal and strict, the implementation can be generated in an automated way
- Upper case tools use this to generate code from the model (for a certain execution environment)



Example std



Simple state implementation, cont'd

nested state

```

case S32:
  switch(e){
  case E2:
    if (g()) {
      a12();
      a9();
      state=S31
    } else{
      a11();
      state=S33;
    }
  }
}

```



State impl example 2

Having a method per event reduces the nesting level of the state machine.

This is the same as event then evaluate state.

```

// events have event method
public void e1(){
  switch (state){
  case S1:
    a6();
    state=S2;
    break;
    // all other states do nothing
  case S2:
  case S3:
  case S31:
  case S32:
  case S33:
    break;
  }
}

```



State impl example 2

```

// method e2 handling nested state
public void e2(){
  switch(state){
  case S1: break;
  case S2: break;
  case S31:
    a10();
    state=S32;
    break;
  case S32:
    if (g()){
      a12();
      a9();
      state=S31;
    } else {
      a11();
      state=S33;
    }
    S3H=state;
    break;
  }
}

```



UML state representation

State chart syntax
Transition notation
Composite states

Implementing statecharts

- Presented are two simple examples of statechart implementation strategies, you will see them back during the laboratory work
- True object oriented implementations (e.g. based on the state pattern) will be introduced during the patterns part.

