Delay Alternative

```
select
  accept Read( ... ) do
  ...
  end;
or
  accept Write( ... ) do
  ...
  end;
or
  delay 10*Minutes;
  -- time out statements
end select;
```

Selected if no other alternatives within indicated time

Autonomous Periodic Tasks (one loop for each task)

```
task body Periodic_Task is
  Next_Time : Calendar.Time := Task_Start_Time;
begin
  delay (Task_Start_Time - Calendar.Clock);
  loop
    -- do work
    Next_Time := Next_Time + Task_Period;
    delay (Next_Time - Calendar.Clock);
  end loop;
end Periodic_Task;
```

Not the same delay as in the select statement.

Problem with Previous Approach

- Delay jitter:
  - A task can be preempted between reading clock and starting its delay, making the delay end later than planned.
  - Cumulative effect is that deadlines can be missed.
  - A better approach is to use a central dispatcher task.

Dispatcher Model

```
task body Periodic_Dispatcher is
begin
  loop
    accept Clock_Interrupt;
    loop
      -- Determine which task to activate next
      select
        Selected_Task.Activate;
      else
        -- Handle missed deadline
      end select;
    end loop;
  end loop;
end Periodic_Dispatcher
```

Highest Priority

```
task body Periodic_Task is
begin
  loop
      accept Activate;
    end loop;
end Periodic_Task
```

Better Model: Use builtin *delay_until*

```
task body Periodic_Task is
  Next_Time : Calendar.Time := Task_Start_Time;
begin
  delay_until (Task_Start_Time);
  loop
    -- do work
    Next_Time := Next_Time + Task_Period;
    begin
      delay_until (Next_Time);
    exception
      when Calendar.Time_Error => -- handle missed deadline;
    end;
  end loop;
end Periodic_Task;
```

Priority Issues

- To get Ada to use the Priority Ceiling Protocol:

  pragma Locking_Policy(Ceiling_Locking);

- Recall that PCP guarantees:
  - Execution of a high-priority task can be delayed by at most one lower priority task per call.

Sources

- A lot of these examples are from:
  ftp://ftp.aw.com/aw.computer.science/Barnes4e/code.txt
  which in turn are from: Barnes, J. G. P. *Programming in Ada*. (4th edition) Addison-Wesley, 1994.
- See also:
  - Ben-Ari, M. Principles of Concurrent and Distributed Programming. Prentice-Hall International, 1990.
  - Alan Burns and Andy Wellings. Concurrency in Ada. Cambridge University Press, 1995.
  - Kjell Nielsen and Ken Shumate. Designing large real-time systems with Ada. Multiscience Press, Inc. 1988.
  - Mark W. Borger, Mark H. Klein, Robert A. Veltre. Real-Time Software Engineering in Ada: Observations and Guidelines. Tech. Rept. CMU/SEI-89-TR-22, 1989. http://www.sei.cmu.edu/publications/documents/89.reports/89.tr.022.html

Also discusses RMA, PCP

Real-Time Models

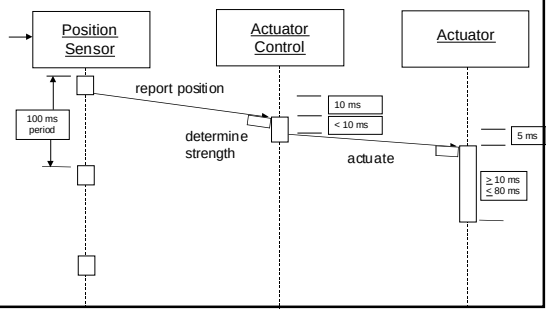
Real-Time Models

- Sequence Diagrams
- State Charts & Variants
- Time Petri Nets
- Temporal Constraint Networks
- Logic Models

Sequence Diagram (UML)

- Objects are shown as vertical lines
- Messages cross between these lines.
- The time sequence is shown by vertical position, flowing down the diagram.
- Time is not generally to scale.
- Bars indicate periods of object activity.

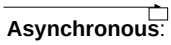
Sequence Diagram for Robot Grip



Added Facets of Sequence and Collaboration Diagrams

- Objects can send messages to *themselves* (e.g. one method implemented using another, including recursion).
- Pre-conditions can exist on message arcs: [... pre-condition ...]
- Object destruction indicated by large X

UML Message Types

-
  Simple: No specific detail of control passing
- 
  Asynchronous: No explicit reply before sender resumes.
-
  Synchronous: Nested flow, as in typical method calls; caller waits for reply before resuming
-
  Synchronous (“immediate” reply)

Problems with Sequence Diagrams

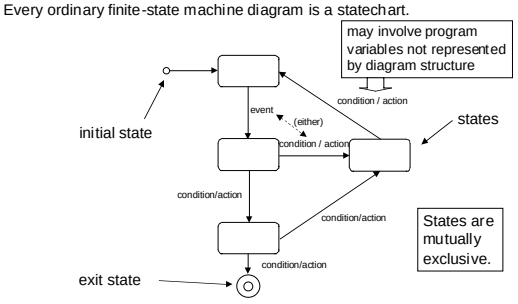
- Only represent one scenario, rather than all possible scenarios
- Informal, therefore subject to misinterpretation

State Charts

- Invented/Discovered by David Harel
- Add to finite-state machines:
  - Concurrency
  - Hierarchy
  - Arbitrary additional variables
  - Timing constraints
- Unlike sequence diagrams, do represent all possible behaviors

StateCharts
(David Harel, Weizmann Inst.)

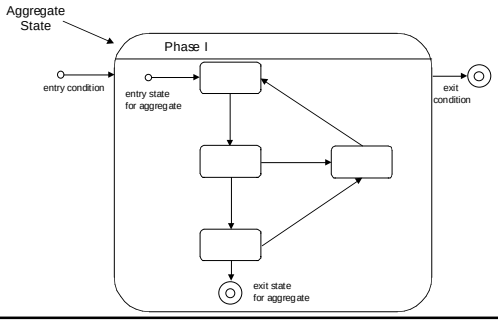
Every ordinary finite-state machine diagram is a statechart.



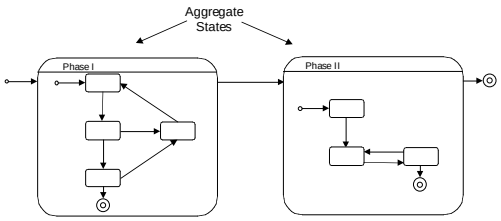
Events vs. Condition/Action pairs

- Events designate some named event occurring.
- Condition/Action represents an event triggered by a condition on program variables and an action that may assign to those variables.
- The null action is that all variables are unchanged.
- The null condition is the same as “true”.

Nuance: Hierarchy in StateCharts



Hierarchy



Words signaling that an aggregate state might be appropriate:

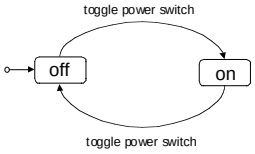
- Phase
- Mode
- Region
- Interval

Statechart Example

- Combination FM/AM Radio and Cassette Tape Player (Auto-reverse)
  - Either the radio or the tape player is playing, but not both.
  - The unit can play only if power is on.
  - The tape plays iff a tape is inserted.
  - The radio can play in AM or FM.

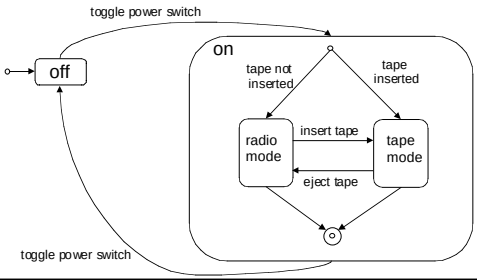
Radio/Tape Player State Diagram

Top-Level View



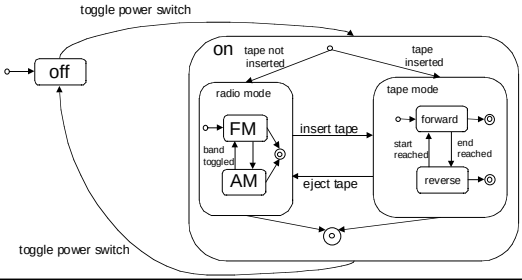
Radio/Tape Player State Diagram

First-Level Expansion



Radio/Tape Player State Diagram

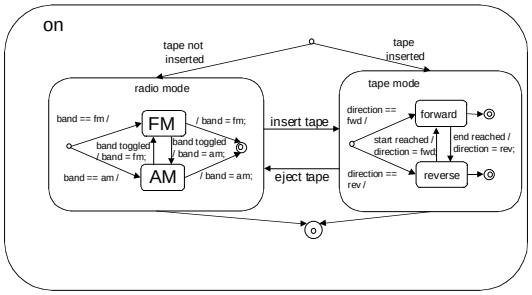
Second-Level Expansion



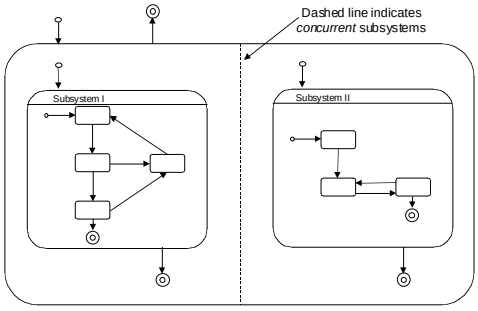
Possible Design Flaws Detected from Diagram

- Radio doesn't remember which band it was in when turned off.
- Tape doesn't remember which direction it was moving when turned off.

Redesign to Remedy Flaws



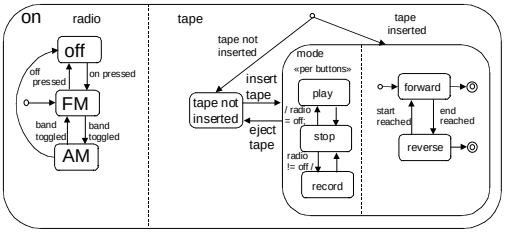
Another Nuance: *Concurrency* in StateCharts



Augment the Radio/Tape Design to Permit Recording from Radio

- The Radio and Tape can now operate concurrently.
- The tape may record in either direction.
- We can no longer let the presence of the tape dictate the mode; must use buttons.

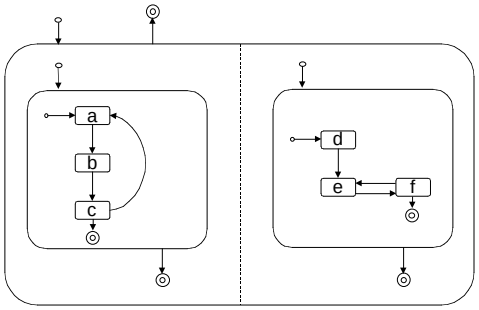
Partially-Developed Radio/Tape Player/Record



Concurrency in StateCharts provide representational economy

- A statechart with concurrency provides for the representation of M x N states with only M+N bubbles.
- The combinatorial explosion of states is thereby reduced.

Exercise: This chart shows 6 labeled states.
Show the actual state transitions, assuming that transitions can occur freely and entry and exits don't get shown explicitly.



Time Constraints in Statecharts

- Use time constraints in conditions:
  clockA $\geq$ 1 sec.
  Annotate transitions with temporal inequalities, such as:
  - $\geq$ 5 ms
  - $\leq$ 30 ms
  - $\leq$ transitionB + 10 ms

ModeCharts

- This term is used by Al Mok at UT Austin.
- It is a model in which upper and lower time bounds can be provided for each transition.
- There is a semantics in RTL (Real-Time Logic).
- This allows various behavioral assertions to be verified formally.

Exercise: Develop a specification for the following real-time problem:

- A home alarm system has the following objects:
  - Siren
  - Entry switch (indicates when door is open)
  - Keypad (for arming/disarming)
- The keypad is located inside the home and one must use the door to get to it.

Specification continued

- If the system is not armed, the siren will not sound.
- When the system is set to be armed via the keypad, the user has 30 seconds in which to use the door before the system actually becomes armed.
- If the door is opened while the system is armed, the siren will sound, *unless* the combination is entered on the keypad within 15 seconds.
- Once the siren begins sounding, it will continue until disarmed at the keypad or until for 15 minutes have elapsed, at which time it will enter an advisory state that indicates an alarm occurred.
- The system will stay in the advisory state until disarmed at the keypad.

Develop Specification

- Compare using:
  - Sequence diagram
  - State chart

Homework, Part 1

- Code the solution to the problem in a real-time high-level language of your choice.

Artificial Emotion:
The next step after AI

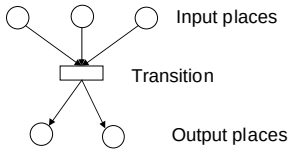
- Signs on service doors in an Atlantic City hotel:

> Do Not Open.
> Door is alarmed.

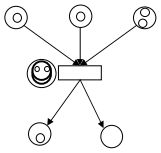
Petri Nets

- Petri net components:
  - "Places" (circular nodes): represent distributed state: Each place has 0 or more **tokens**.
  - "Transitions" (rectangular nodes): represent local state transitions:
    - A transition is enabled or "firable" only when its input places each have at least 1 token.
    - When a transition fires, it removes tokens from its input places and adds tokens to its output places.

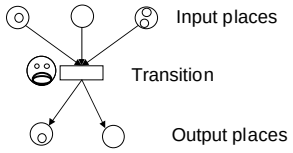
Places and Transitions



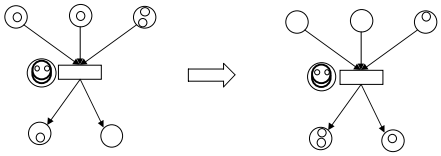
A Firable Transition



An Unfirable Transition



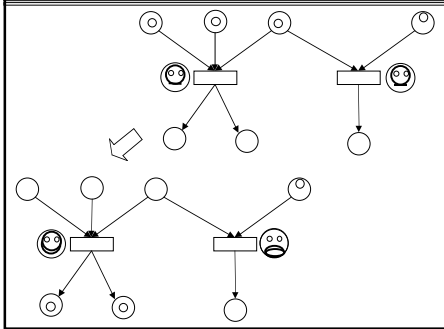
Result of Firing



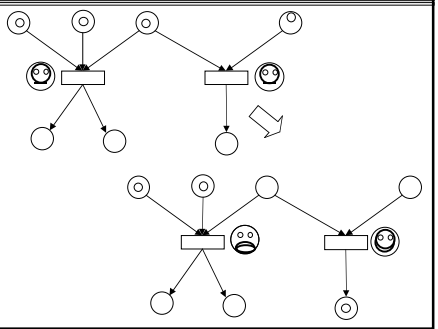
Petri Net Behavior

- If two transitions share input places, firing one transition can disable the other.

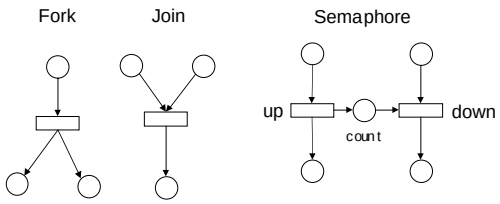
One Transition Disabling Another (called “conflict”)



Alternate Resolution



Common Petri Net Patterns



Common Petri Net Patterns

Mutex