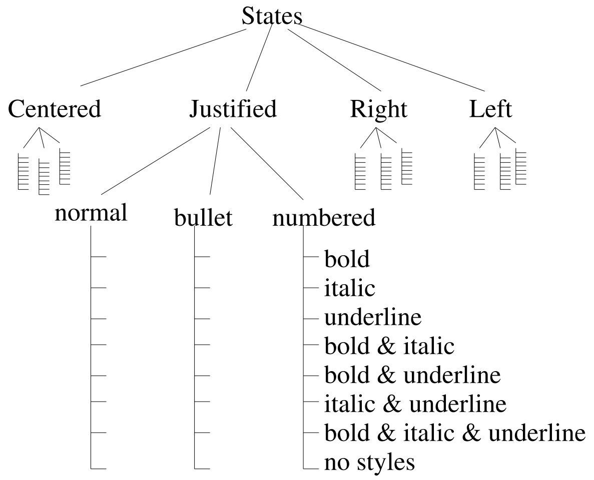
Simultaneous States

- suppose we want to model the behaviour of three functions of a word processor:
  1. font change (bold, italic, underline)
  2. paragraph alignment (centred, left, right, justified)
  3. paragraph layout (normal, bullet list, numbered list)

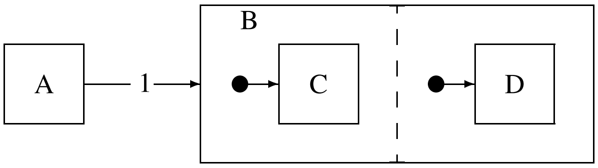


96 different states!!

- Observation: the three characteristics (font, alignment and layout) are in fact independent
- the states do not affect each other (a text being bold does not affect its kind of alignment)
- the three characteristics only happen to be able to occur *at the same time*
- representing all possible combination of states is then
  - superfluous (add unnecessary complexity to the system)
  - wrong (it gives the impression that there are some relations among states which in fact do not exist)
- we would like to be able to represent independent states that, however, do form a unique interface

Concurrency

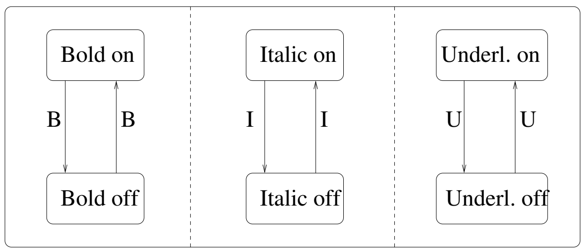
- statecharts can represent the fact that several states can be active at the same time very effectively
- the concurrency between two states is represented by a dashed line which separates them in the super-state they are part of
- for example, the following statechart means that, if event 1 occurs while the system is in A, then the system will enter in state C **AND** in state D simultaneously



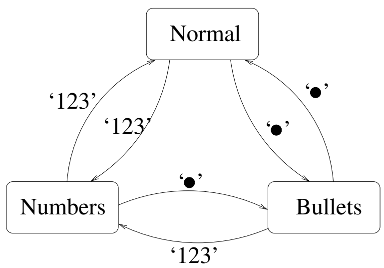
- Therefore, the description of the state of the system in a concurrent statechart is given by ALL the states of the sub-charts which hold simultaneously

Word Processor Example

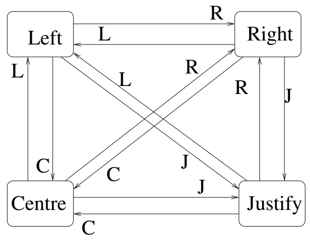
- back to the example: we can think of the three font types as three concurrent states:



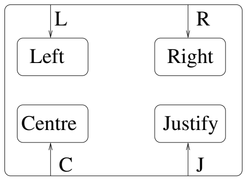
- independently, we can design the sub-chart for the type of paragraph (suppose you have only buttons for “numbers” and “bullets”, and if none is pushed down then you have “normal”)



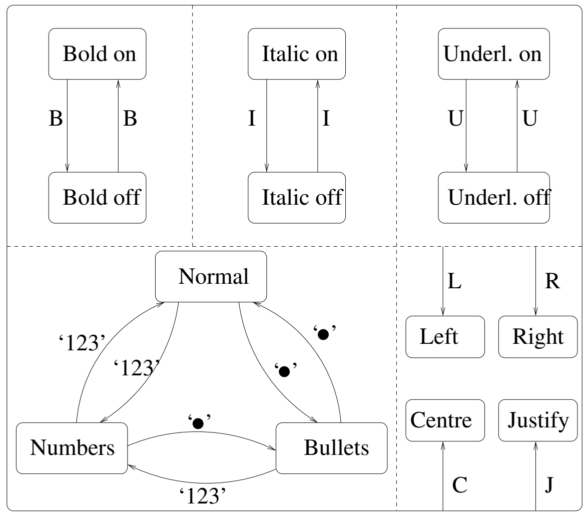
- finally, we need to represent the behaviour of the alignment states:



- the chart is not easy to read
- we may observe that:
  - each state is connected to any other state
  - every arrow terminating at a state has the same label (i.e. the state can be reached if the event on the label occurred, no matter who caused it)
- a much better representation can then be:

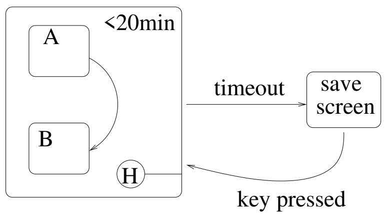


Putting Things Together

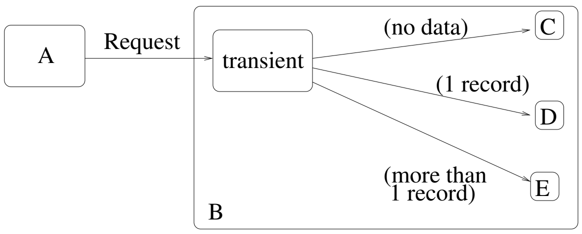


Other Notations

- delay



- transient states: the system waits for an external event

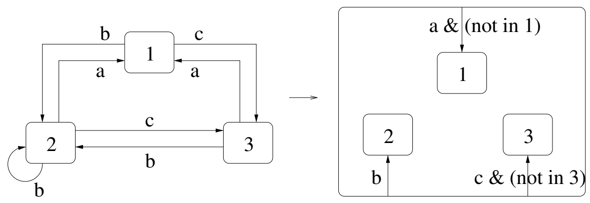


note: there are only conditions on the arrows from the transient state

Design Techniques

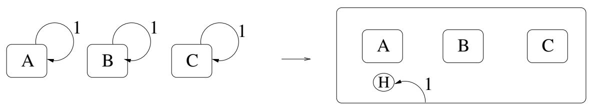
Refinement: a state, or a group of states, behaves in identical way with each of its neighbors, or two or more states are substantially identical

Clustering: there is a set of states and the same event causes a transition from each of them. A superstate is created and each event is replaced by just one arrow attached to the superstate

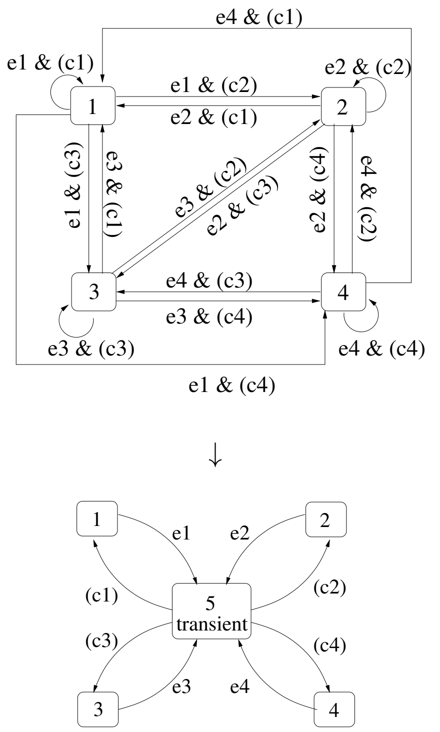


Concurrency: groups of closely related states can be identified and states in one group are not related to states in another group

History: the same event is attached to a set of states



Transient states: only the conditions associated with some events determine the next state



Design Heuristics

- Identify the high level objects:
  - in most cases they correspond to the main screens of the application, with their windows and subwindows
  - design one screen at a time but start to understand how two screens interact as soon as you can
  - if a window only appears in a screen, it should be placed at low levels
  - if a window is called from more than one screen, it should appear at the highest level in the hierarchy
- Write down all the items in or associated with a screen: buttons, menu items, text items, radio buttons, scrolling lists, keyboard shortcuts, etc.
- Establish whether the behaviour of an item is constant:
  - when a button is clicked, will the action be always the same, or will it depend on the context?
  - is an item always available, or is it sometimes disabled?

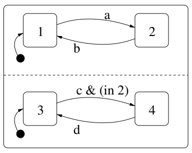
Screen rules

- Exit and Entry rules:
  - what events cause a screen to be entered/exited?
  - what events cause an application to be closed?
  - do exiting or entrying change according to the context (e.g. closing a file without saving)?
  - are there permissions granted to some users only?
- Mode rules:
  - what states in which the software can be cause the effect of user events to change? (e.g. in a drawing package the pencil, spray and paint brush modes change the cursor appearance)
  - what characterizes these modes and why and when a mode may change? (e.g. can a mode change within a screen, or just on entry to it?)
- Behavioural rules:
  - what items change their behaviour with the interaction? (enabled/disabled, red/blue etc.)
  - are there items whose behaviour is constant, but whose actions may differ under different circumstances (e.g. Cut with no selection)?

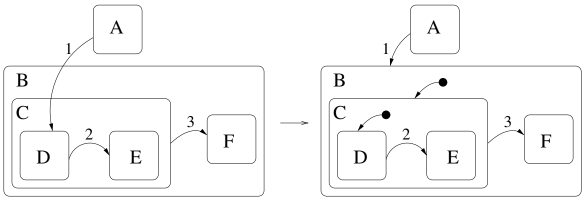
Creating the Statechart

1. identify the states:
   - main screen and subscreens → top level states
   - screen modes → the statechart may be designed to switch between modes (e.g. file open/closed)
   - a way to identify modes is to consider the objects with varying behaviour (Save+Exit / Exit)
2. consolidate related behaviour:
   - search for common events and conditions: items that affects each other behaviour are likely to be in the same state
   - search for groups of items that work together as an entity: they are likely to be in the same concurrent state
   - a screen item should be controlled by precisely one part of the statechart
3. separate unrelated behaviour:
   - identify aspects of a screen that are completely independent: each of them is likely to form one of a set of concurrent states
   - use extensively refinement and depth

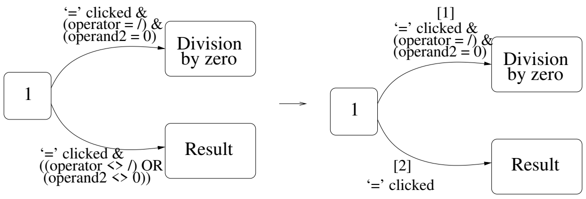
4. eliminate dependencies in concurrent parts (consider merging):



5. maintain each level self-contained



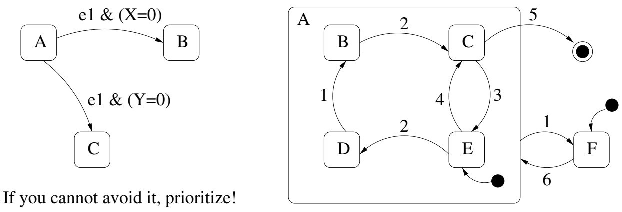
6. avoid convoluted conditions, by prioritizing events:



Design Tests

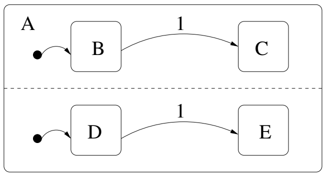
- Has all the required behaviour been modelled?
- Is it possible to reach all the states?
  - don't forget default start states in concurrent parts and in states using the history mechanism
- Are there any dead states?
  - group of states for which there is no end state
  - the end state cannot be reached because of some conditions on the arrow
  - conflict between states and events: e.g. a state can be exited with the event "Save clicked" but the Save button is disabled in that state
- Is the behaviour of the statechart deterministic?

  There are 2 types on non-determinism:



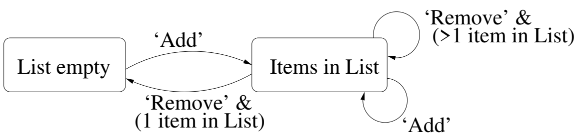
If you cannot avoid it, prioritize!

- Does the statechart rely on the order of events occurring "simultaneously"?



The semantics of the statechart notation state that it does not matter whether event 1 causes the system to enter in state C before state E or viceversa (they are considered simultaneous). If it does matter to you, then you have to design it differently!

- Are the conditions on event arrows directly related to the state the arrow leads to?



Conditions should avoid using variables or the status of an interface object which are not modelled in some way by the destination state.

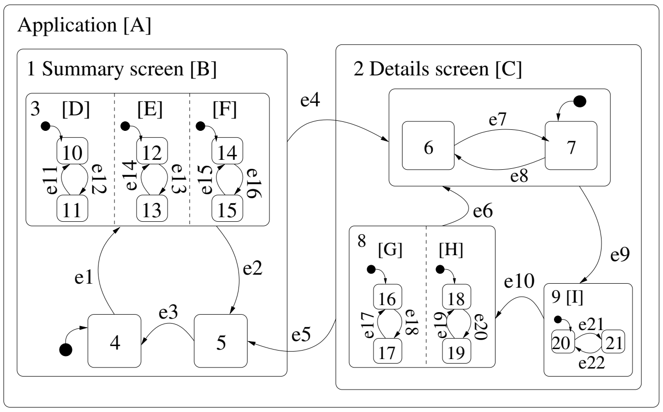
Assigning Names

- choose meaningful names for states, conveying what the state represents (if you cannot think of a name for a state, there might be something wrong in the design!)
- exceptions: transient states (usually labelled with "Transient") and states that are introduced with the unique purpose of clustering (but in many cases the states that can be clustered together do form a meaningful superstate)
- assign identifiers (e.g. numbers) to the states, so that you may use this identifier in tables or when conciseness is important

State Variables

- in STD, a single variable can be used to represent the system's state
- in Statecharts, many are needed, as there can be concurrent states
- state variables are used only for states with a lower level of detail
- the following convention is then used to label a state:

`<State ID> <State Name> [<State Variable>]`



Event-action tables

- often it is not practical to include all the actions associated with an arrow on the statechart, because the description can be long
- the diagram can be drawn without the action labels, and a table can be used to specify the complete details:

Current State	Event	Action	Next State

- as the description of a state is given by many state variables, it can be practical to split the table
- e.g., in the chart on the previous slide: the current state is given by the combination of 9 variables:

Current State	Event	Action	Next State
a b c d e f g h i			a b c d e f g h i
1 5	e3		1 4

- whenever a = 1 we have no values for variables c, g, h and i and whenever a = 2 we have no values for variables b, d, e and f
- it makes sense to define two event tables, one for each main screen, plus an extra table to represent the interface between them

Event action table for the interface between 1 and 2:

Current State			Ev.	Act.	Next State		
a	b	c			a	b	c
1	%		e4	...	2		7
2		%	e5	...	1	5	

(% means "whichever the value of the variable")

Event action table for state 1:

Current State					Ev.	Act.	Next State				
a	b	d	e	f			a	b	d	e	f
1	4				e1	...	1	3	10	12	14
1	5				e3	...	1	4			
1	3	%	%	%	e2	...	1	5			
1	3	10	%	%	e12	...	1	3	11	%	%
1	3	11	%	%	e11	...	1	3	10	%	%
1	3	%	12	%	e13	...	1	3	%	13	%
1	3	%	13	%	e14	...	1	3	%	12	%
1	3	%	%	14	e16	...	1	3	%	%	15
1	3	%	%	15	e15	...	1	3	%	%	14

Event action table for state 2:

Current State					Ev.	Act.	Next State				
a	c	g	h	i			a	c	g	h	i
2	6				e7	...	2	7			
2	7				e8	...	2	6			
2	8	%	%		e6	...	2	6			
2	6				e9	...	2	9			20
2	7				e9	...	2	9			20
2	9			%	e10	...	2	8	16	18	
2	8	16	%		e18	...	2	8	17	%	
2	8	17	%		e17	...	2	8	16	%	
2	8	%	18		e20	...	2	8	%	19	
2	8	%	19		e19	...	2	8	%	18	
2	9			20	e21	...	2	9			21
2	9			21	e22	...	2	9			20