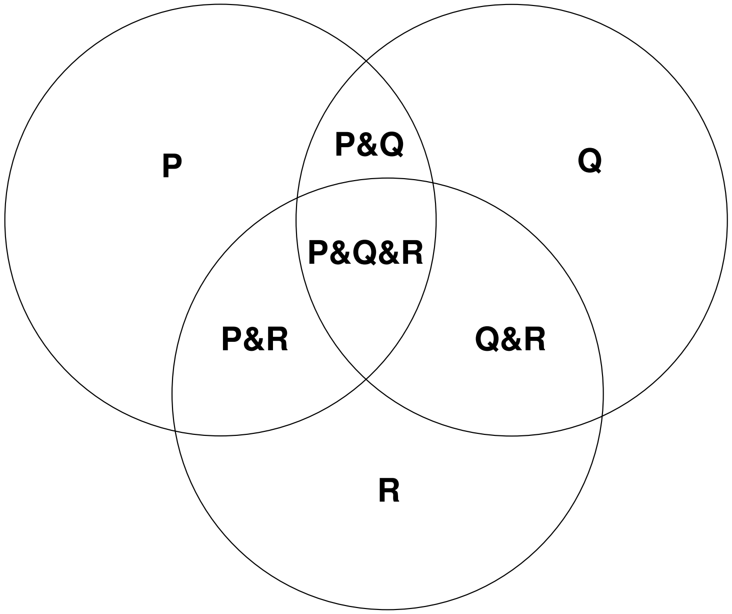
Statecharts aka Harel Charts

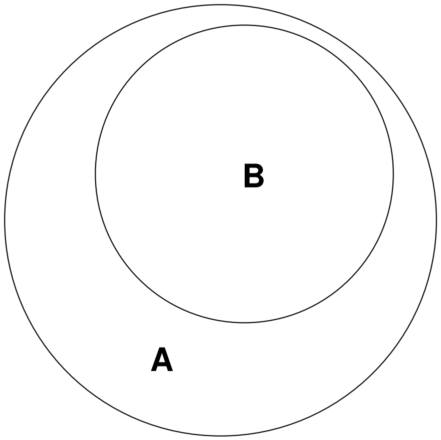
- visual formalism
- higraph based (rigour)
- diverse applications;
  in particular: concurrent systems behaviour

Visualising Information

- complex
- non-quantitative, structural
- topological, not geometrical
- Euler
  - graphs (nodes, edges: binary *relation*); hypergraphs
  - Venn diagrams (Jordan curve: inside/outside): enclosure, intersection

Venn diagrams, Euler circles





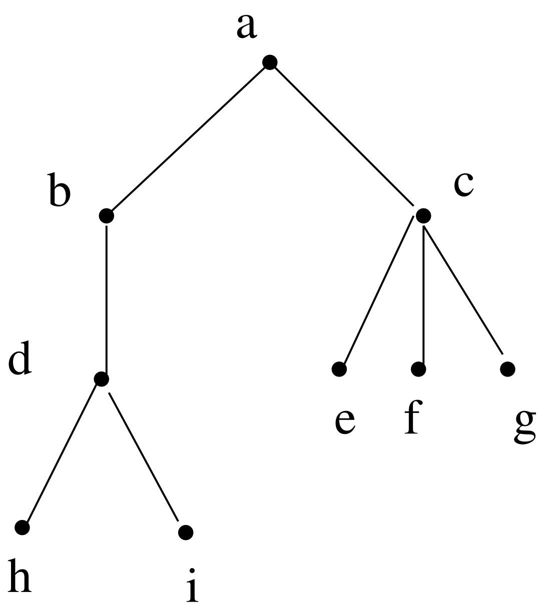
- *topological* notions:

  enclosure, exclusion, intersection

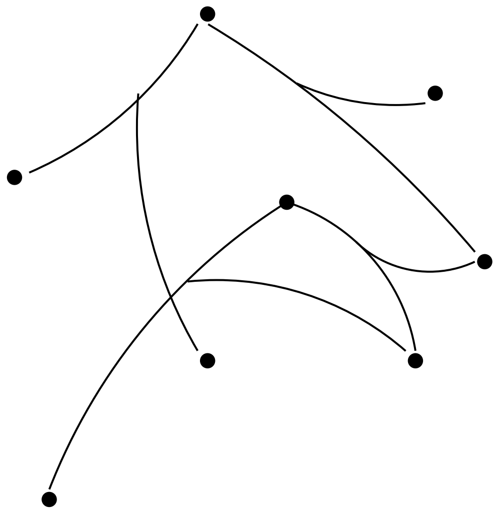
- Used to represent *mathematical* set operations:

  union, intersection, difference

Hypergraph



a graph



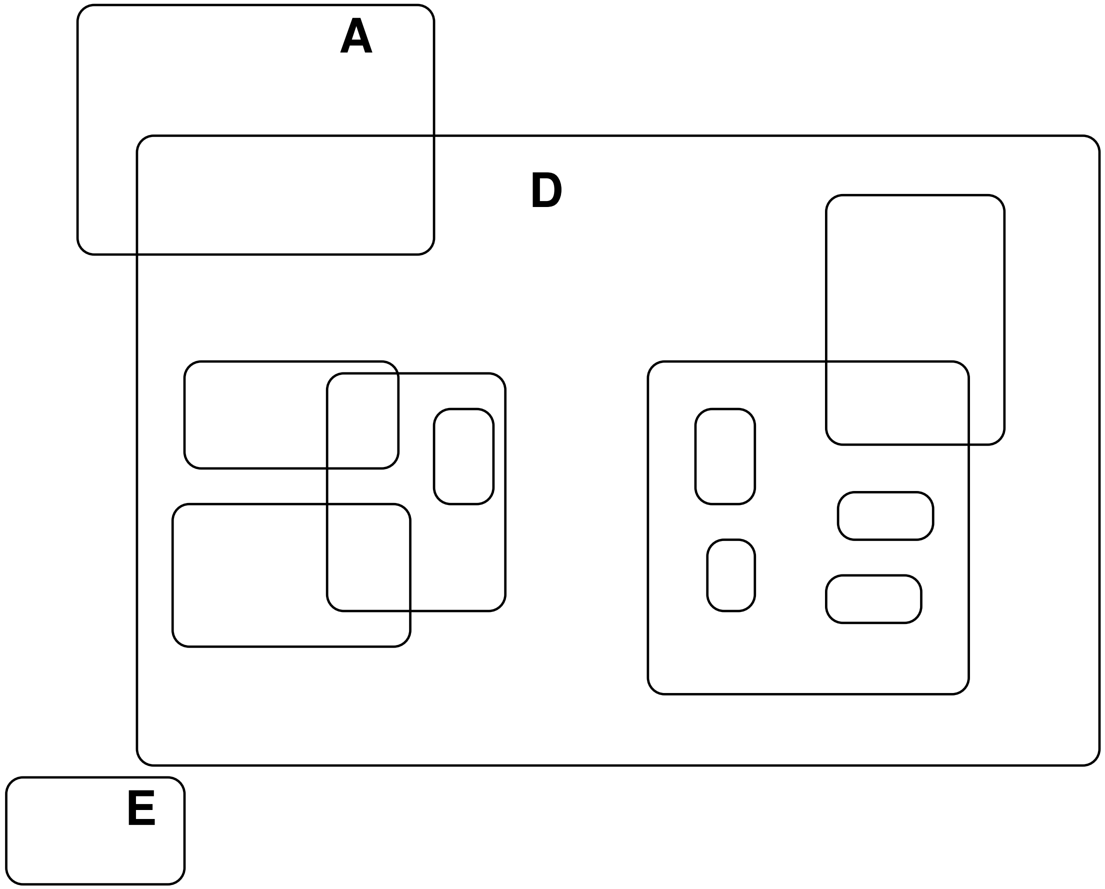
a hypergraph

- *topological* notion: connectedness
- Used to represent *relations* between sets.
- Hyperedges: $\subset 2^X \times 2^X$

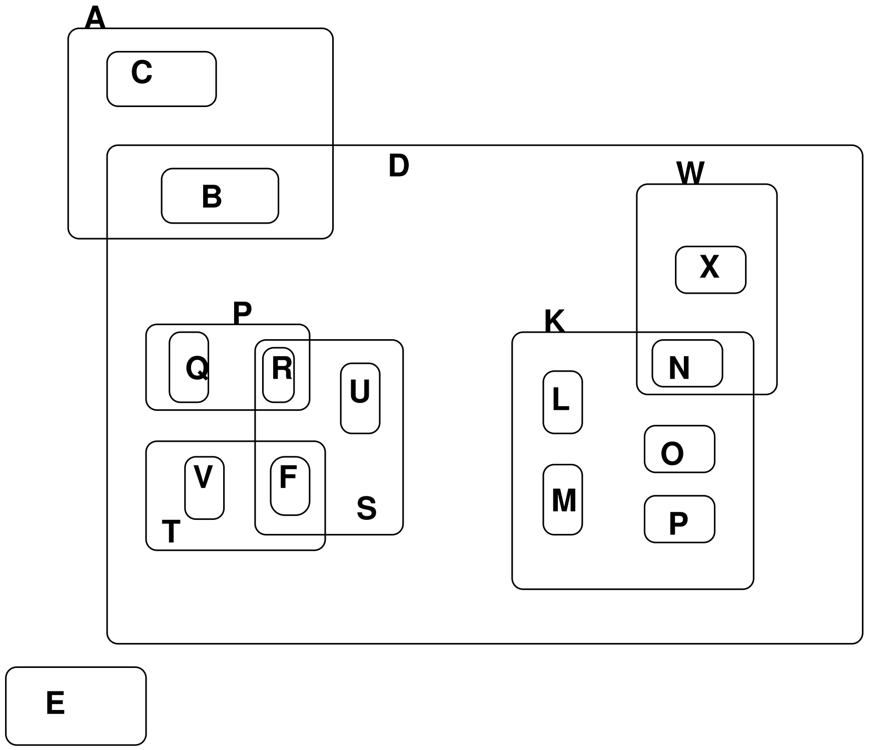
Higraphs: combining graphs and Venn diagrams

- sets + cartesian product
- hypergraphs

Blobs: set *inclusion, not membership*

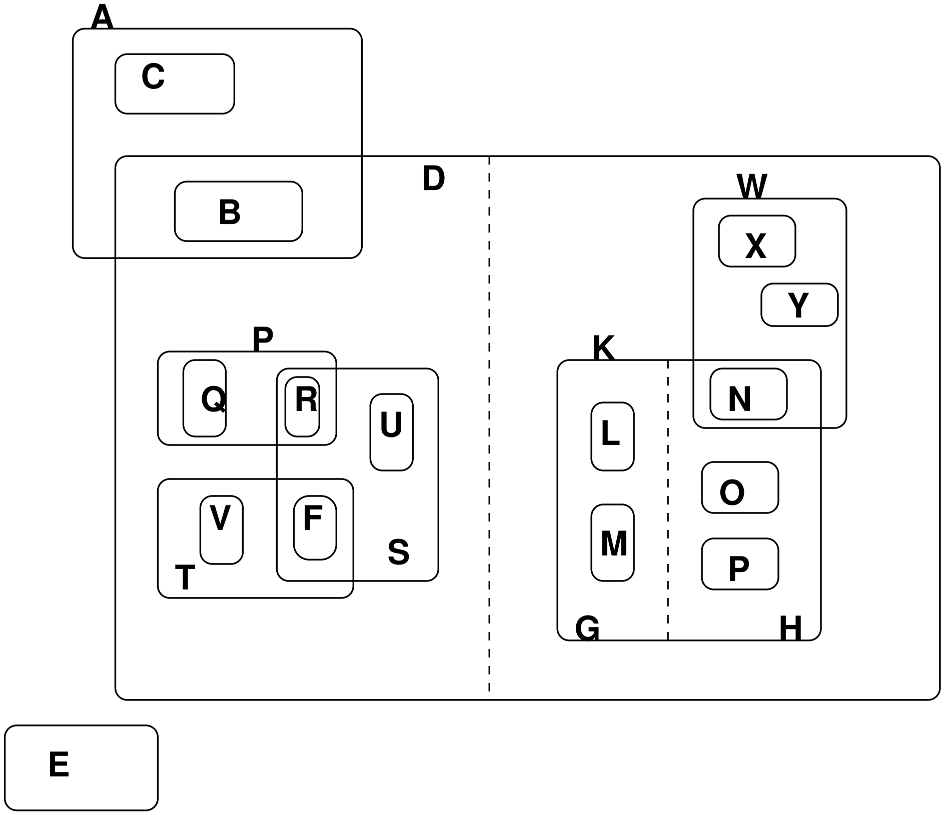


Unique Blobs (atomic sets, no intersection)



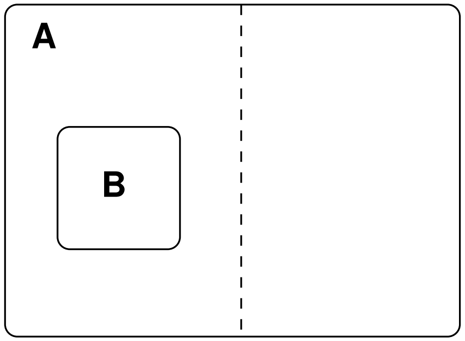
- empty space has no meaning, intersection must be identified
- atomic blobs are identifiable sets
- other blobs are union of enclosed sets

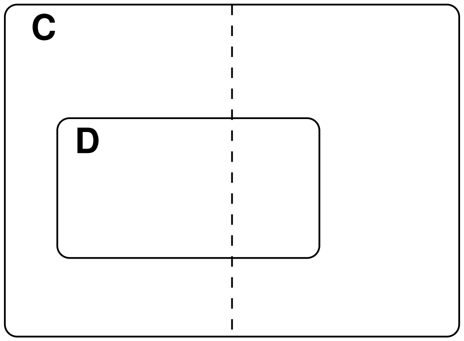
Unordered Cartesian Product: *Orthogonal* Components



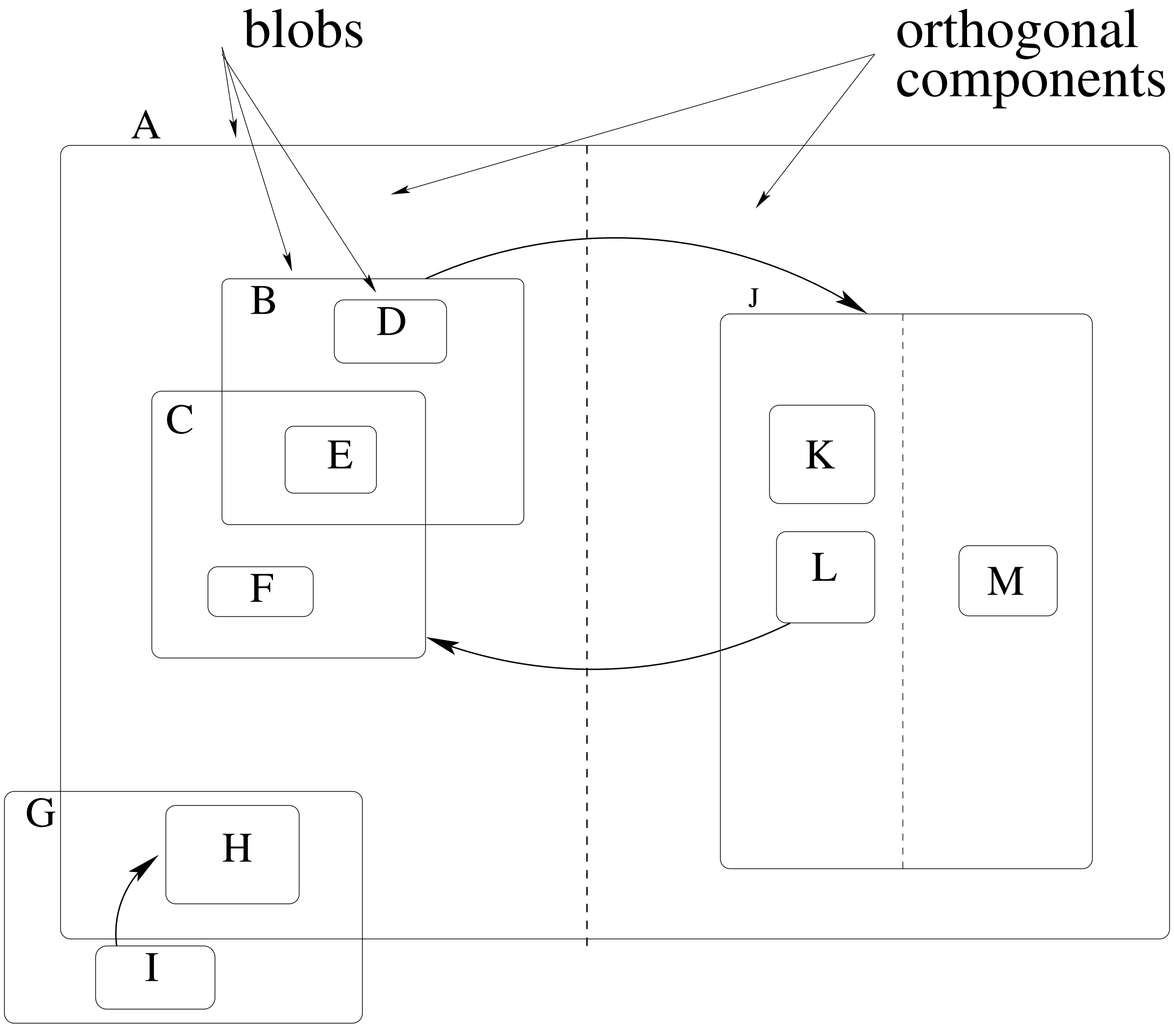
$$K = G \times H = (L \cup M) \times (N \cup O \cup P)$$

Meaningless constructs

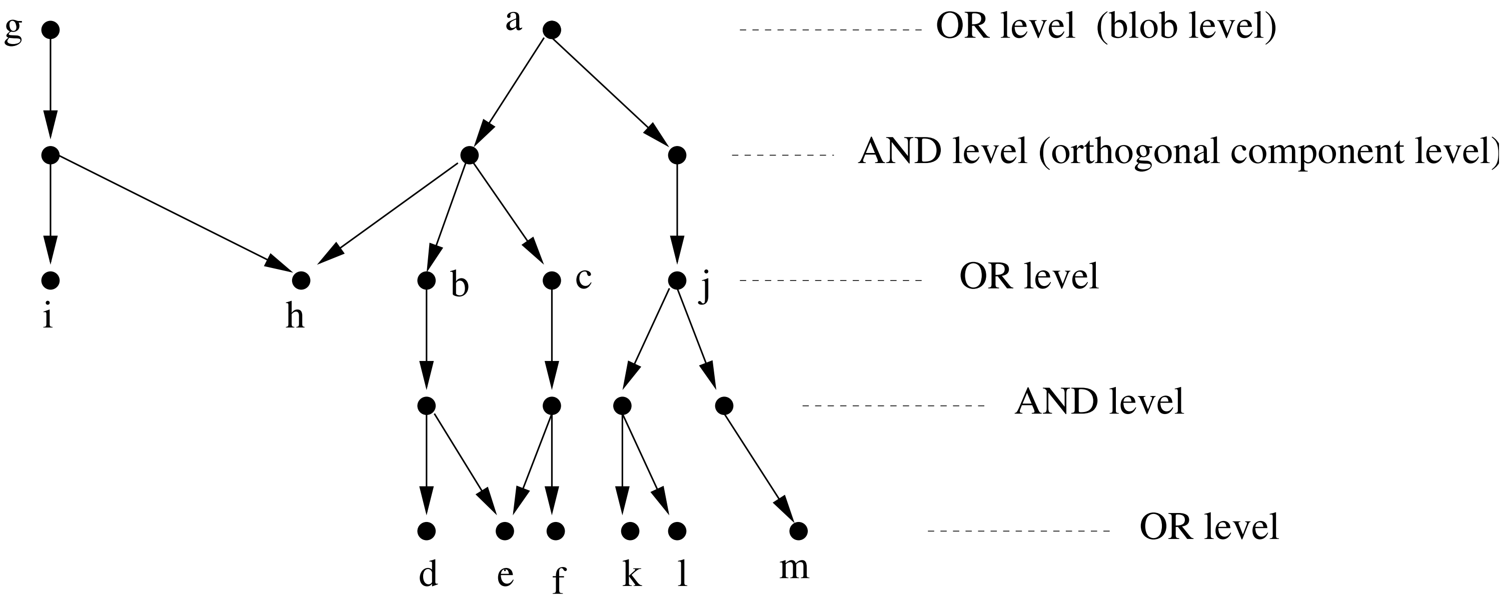




Simple Higraph



Induced Acyclic Graph (blob/orth comp alternation)



OR level (blob level)

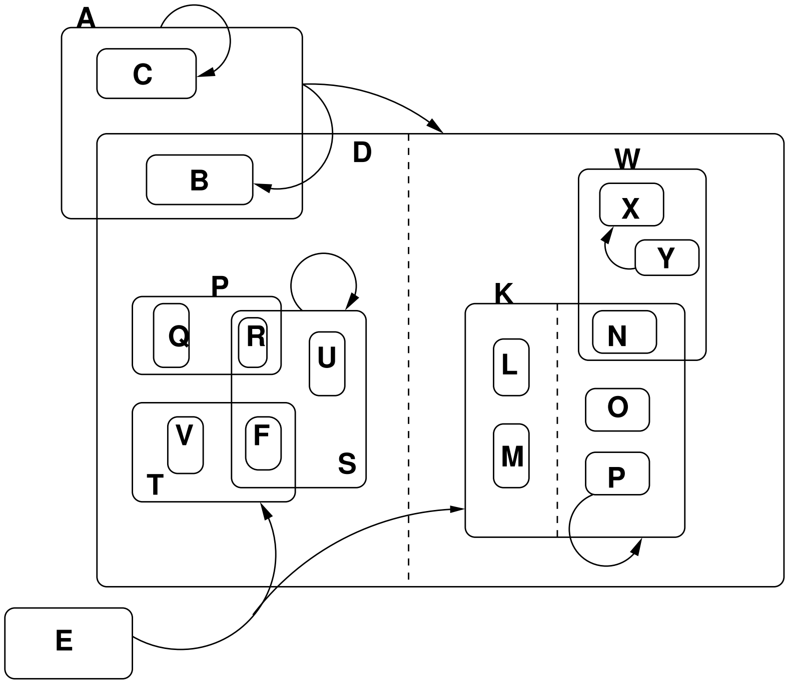
AND level (orthogonal component level)

OR level

AND level

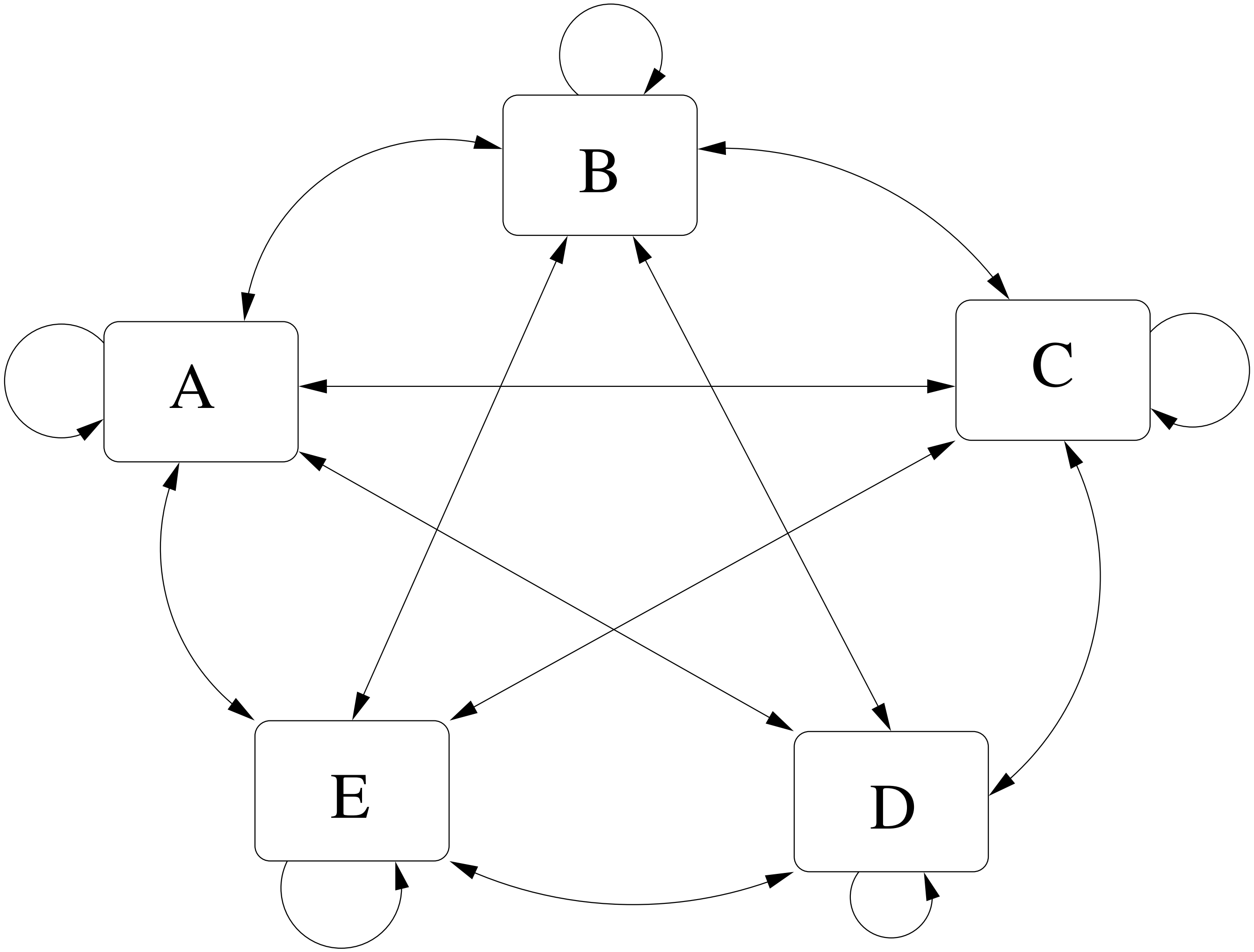
OR level

Adding (hyper) edges

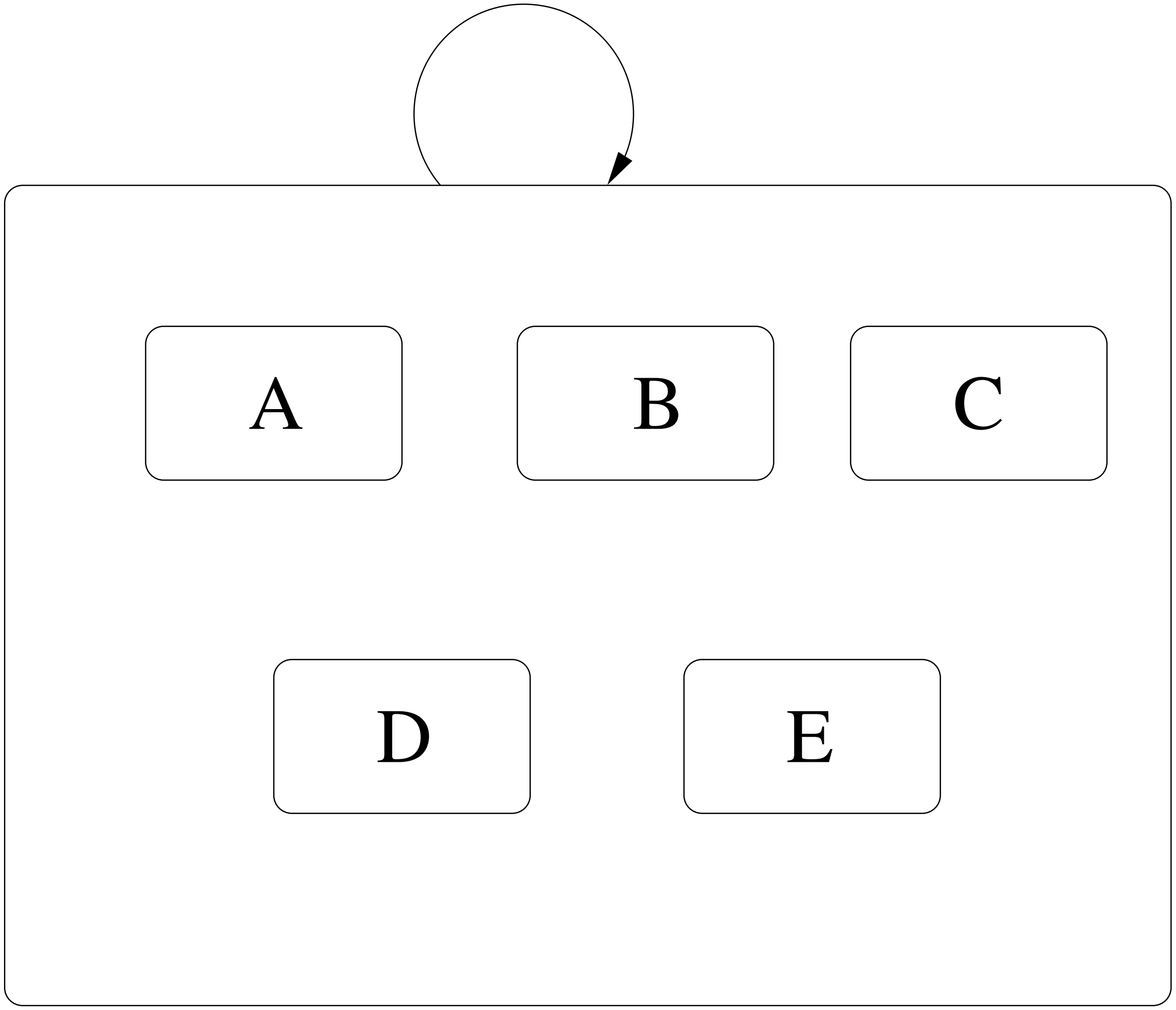


- hyperedges
- attach to contour of any blob
- inter-level possible (global variables binding)

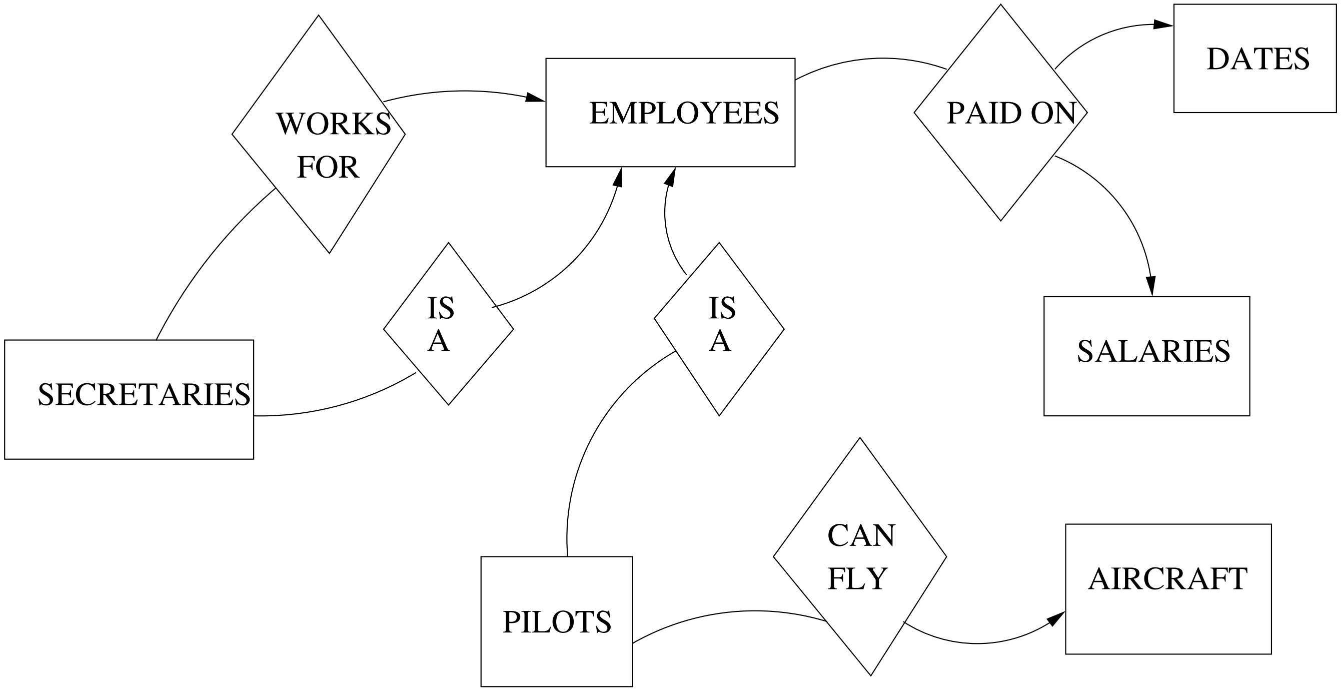
Clique Example



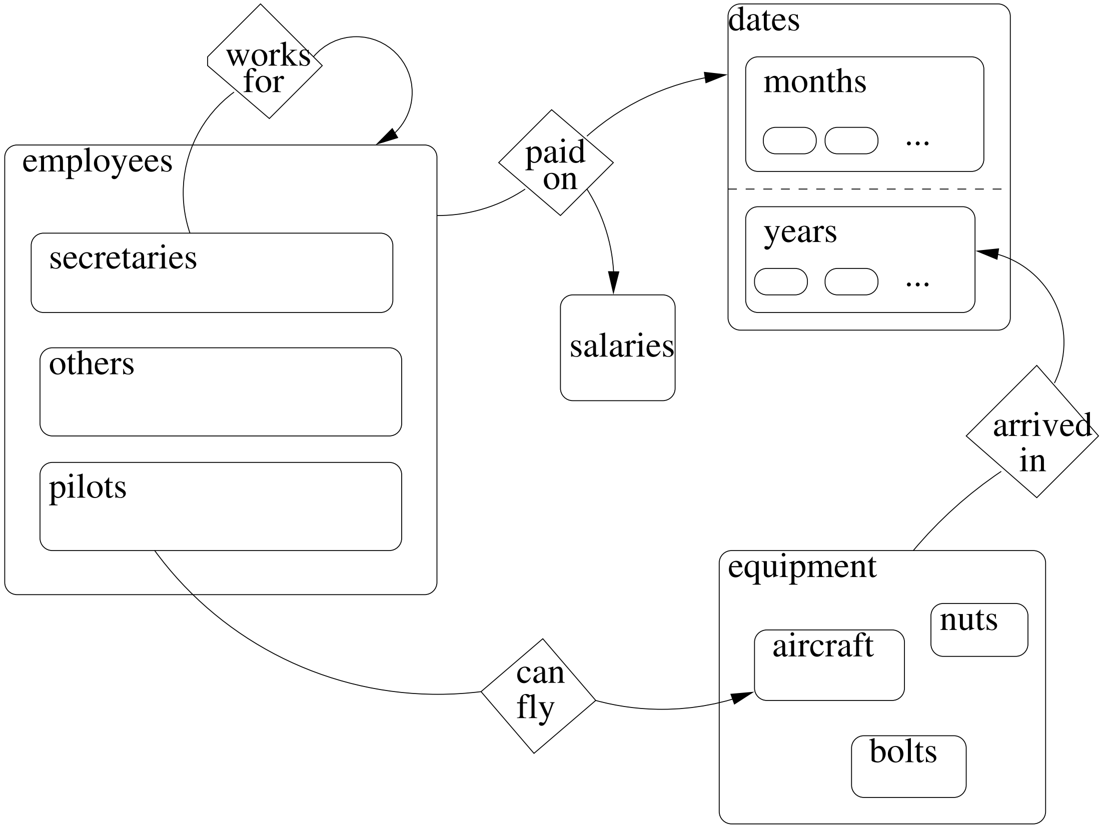
Clique: fully connected semantics



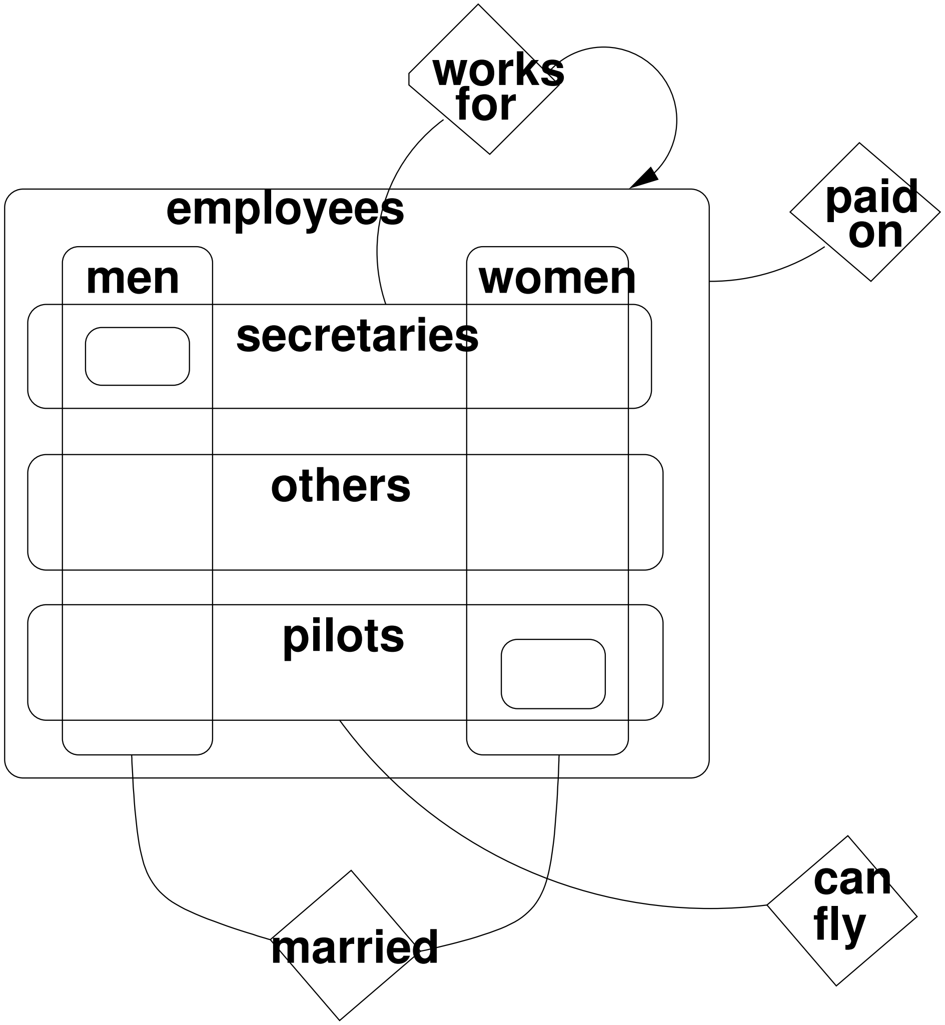
Entity Relationship Diagram (`is-a`)



Higraph version of E-R diagram



Extending the E-R diagram



Higraph applications

- E-R diagrams
- data-flow diagrams (activity diagrams)
  edges represent (flow of) data
- inheritance
- StateCharts

Formally

A higraph H is a quadruple (B, E, ρ, Π)

B: set of all unique blobs

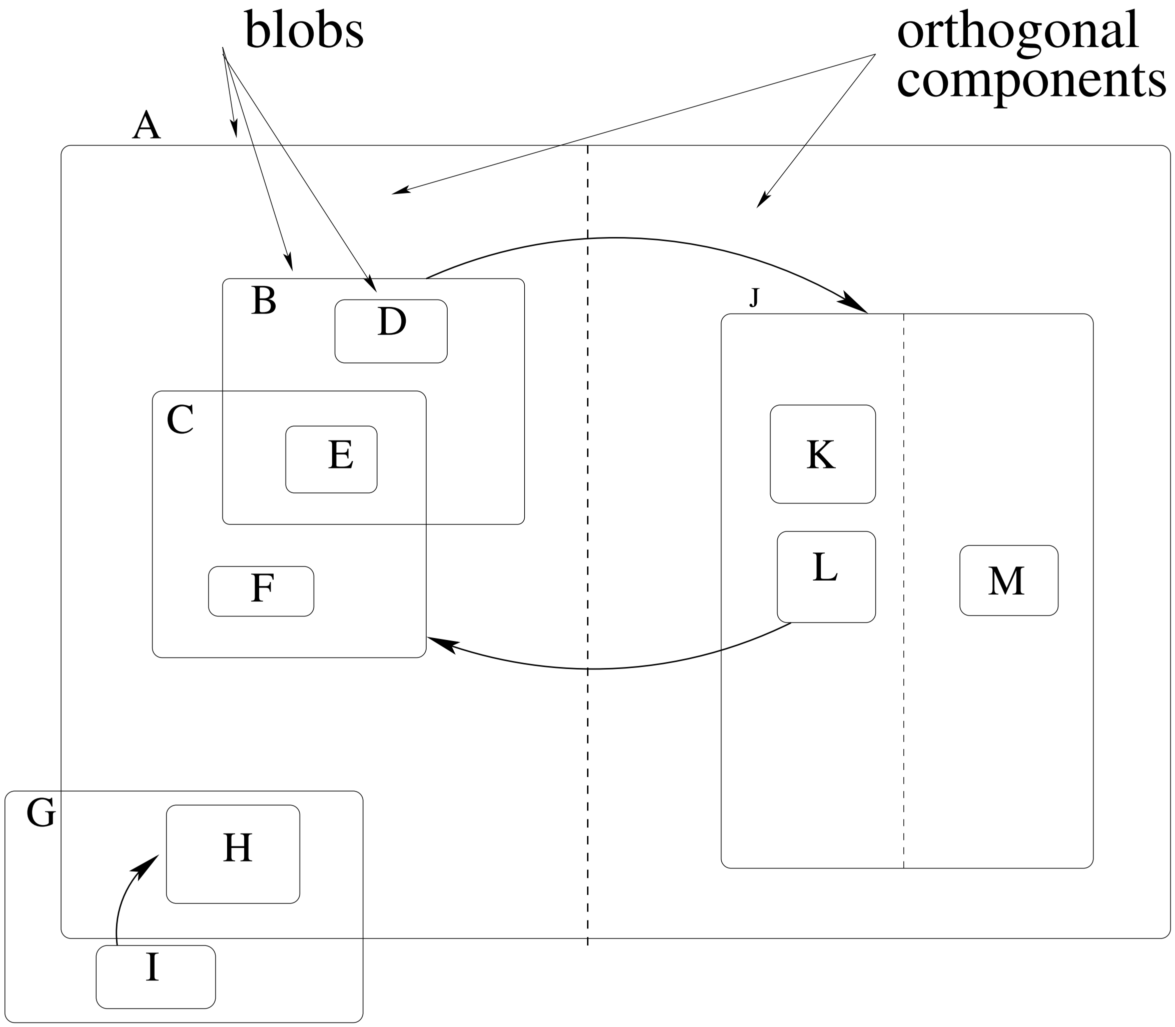
E: set of hyperedges $\subset 2^X \times 2^X$

$\rho : B \to 2^B$, the hierarchy function

$\Pi : B \to 2^{B \times B}$, the partitioning function (equivalence relationship)

ρ defines the direct descendants of a blob
$\rho^0(x) = \{x\}$
$\rho^+(x) = \bigcup_{i=1}^{+\infty} \rho^i(x)$, cycle free: $x \notin \rho^+(x)$

Simple Higraph



Induced Orthogonal Components

$$B = \{A,B,C,D,E,F,C,G,H,I,J,K,L,M\}$$

$$E = \{(I,H),(B,J),L,C)\}$$

$$\rho(A) = \{B,C,H,J\}, \rho(G) = \{H,I\}, \rho(B) = \{D,E\}, \rho(C) = \{E,F\},$$

$$\rho(J) = \{K,L,M\}, \rho(D) = \rho(E) = \rho(F) = \rho(H) = \rho(I) = \rho(K) = \rho(L) = \rho(M) = \emptyset$$

$$\Pi(J) = \{(K,K),(K,L),(L,L),(L,K),(M,M)\}$$

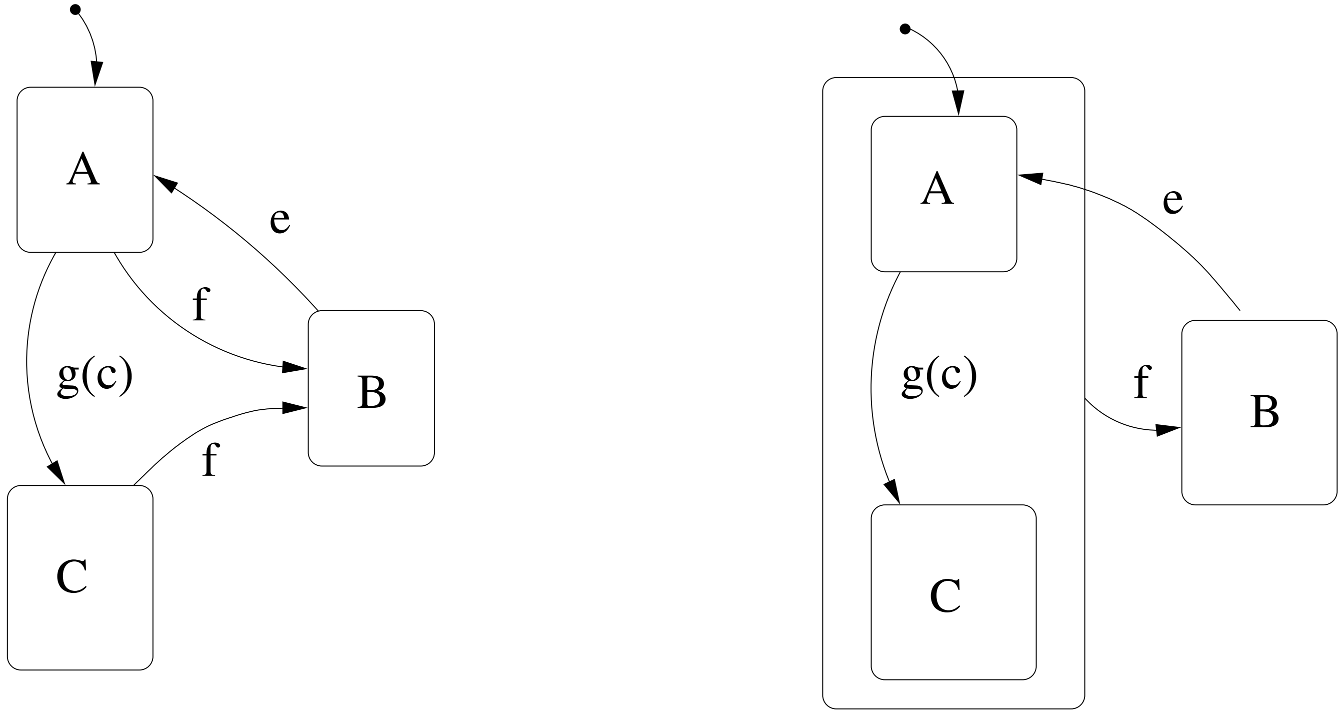
Induces *equivalence classes* $\{K,L\}$ and $\{M\}$, ...

These are the *orthogonal components*

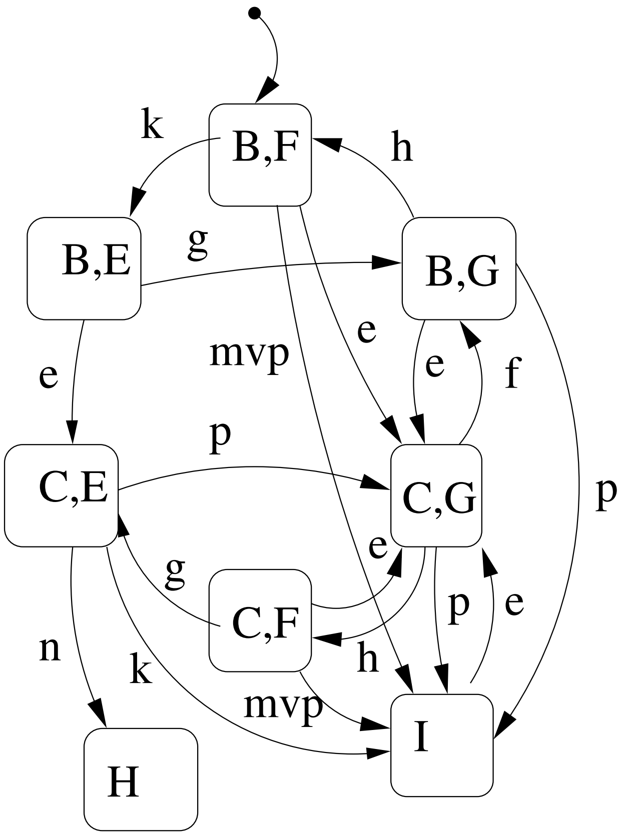
StateCharts = state diagrams + depth + orthogonality + broadcast

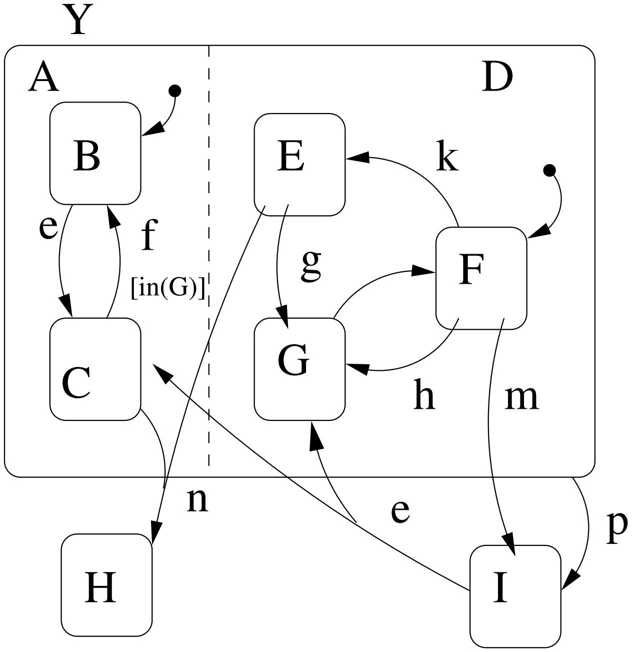
- Reactive Systems (event driven, react to internal and external stimuli)
- like Petri Nets, CSP, CCS, sequence diagrams, . . .
- graphical but formal and rigourous for
  - analysis
  - code generation
- solve FSA problems:
  - flat ⇒ hierarchy ⇒ re-use
  - represent large number of transitions concisely
  - represent large number of states concisely
  - sequential ⇒ concurrent

Depth (XOR)

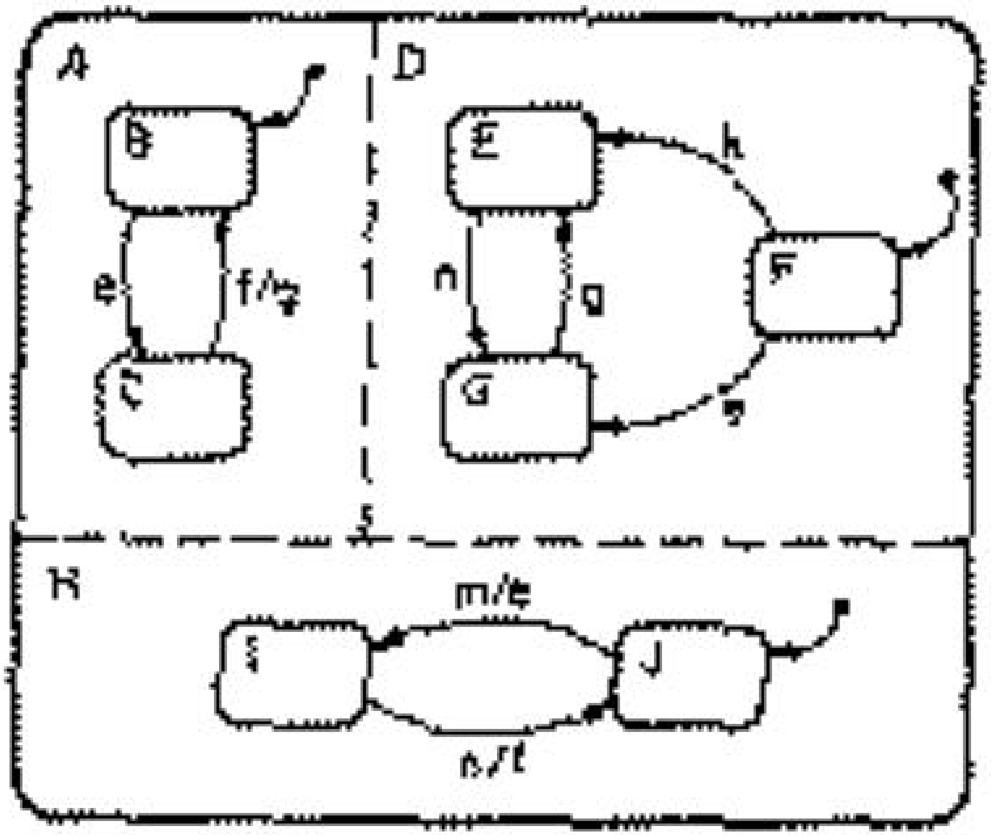


Orthogonality (AND), flattening ⇒ semantics

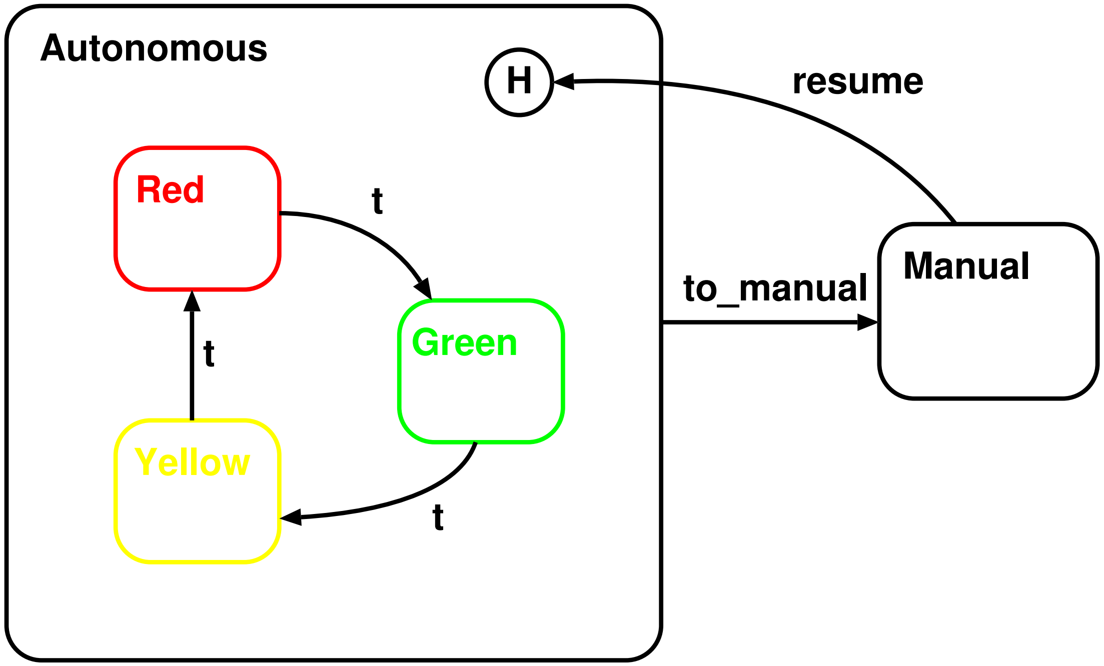


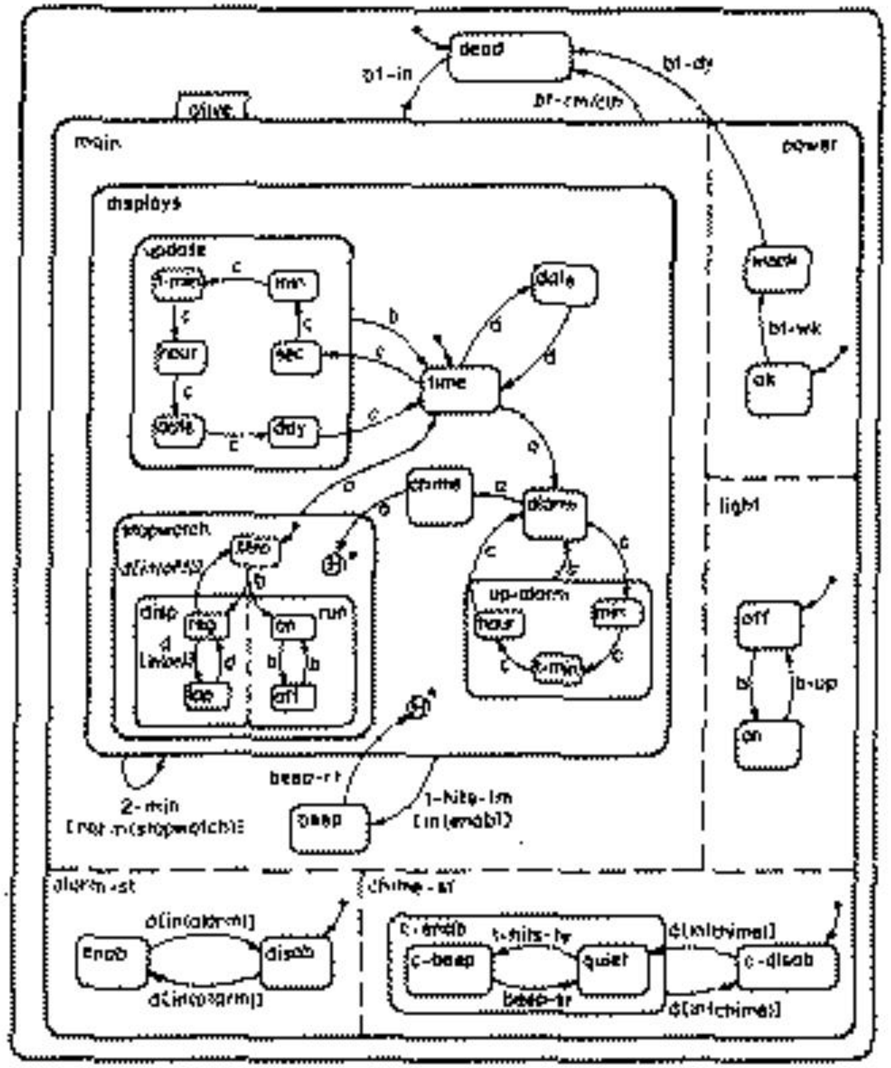


Broadcasting (output events)



History States




dead
bt-in
bt-dy
bt-rm/d
alive
main
power
displays
update
10-min
min
c
hour
sec
date
day
b
time
d
date
chime
a
alarm
stopwatch
zero
d[in(off)]
H*
disp
reg
lap
d
[in(on)]
run
on
off
b
up-alarm
hour
min
10-min
weak
bt-wk
ok
light
off
on
b
b-up
beep-rt
2-min
[not in(stopwatch)]
beep
T-hits-tm
[in(enabl)]
alarm-st
d[in(alarm1)]
enab
disab
d[in(alarm1)]
chime-st
c-enab
t-hits-hr
c-beep
quiet
beep-rt
d[in(chime)]
c-disab
d[in(chime)]

Executable Object Modelling

- analysis ⇒ use cases ⇒ sequence diagrams
- analysis ⇒ use cases ⇒ class diagrams
- ⇒ Statecharts ⇒ sequence diagrams ⇒ test use cases

Executable Object Modelling with Statecharts

- OO development: intuitive *and* rigourous
- fully executable models (simulation)
- code synthesis

Executable Object Modelling with Statecharts

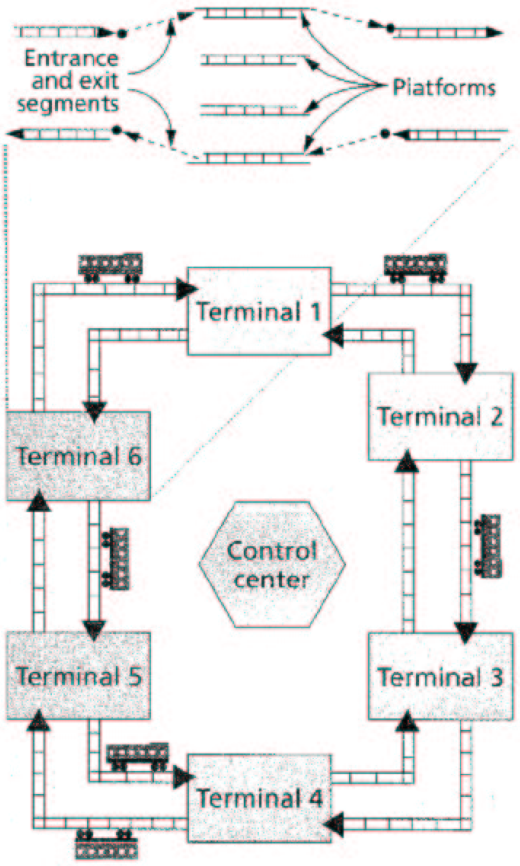
- Structure (classes, multiplicities, relationships)

  ⇒ Object-model diagrams (higraph version of ER-diagrams)

- Behaviour

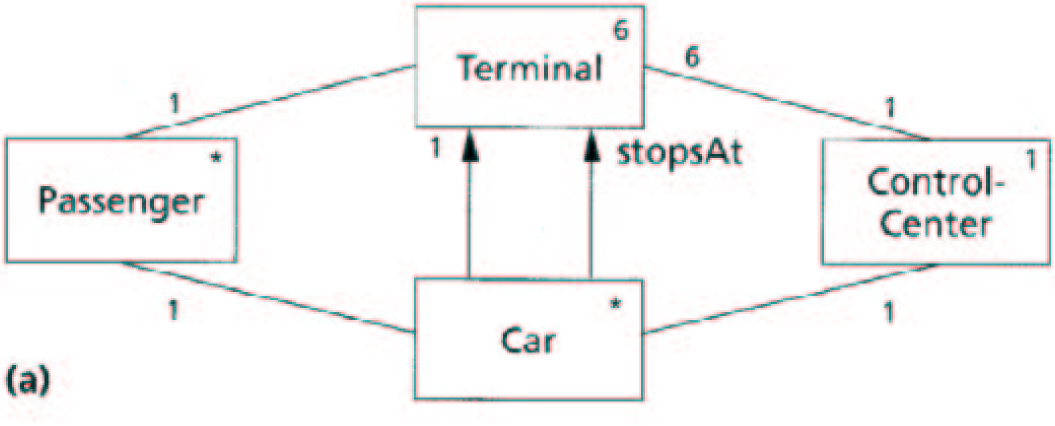
  ⇒ StateCharts

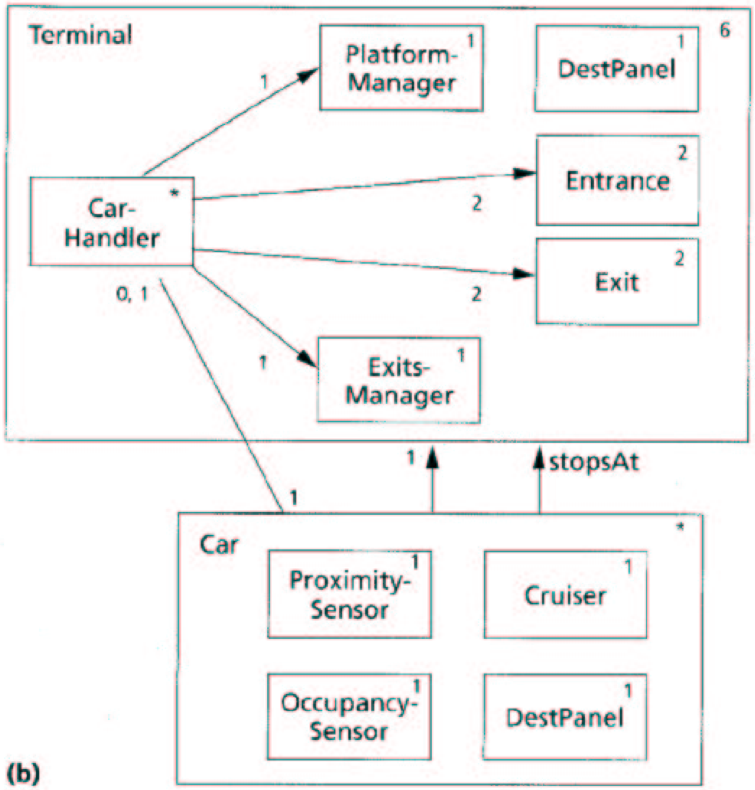
Automated Railcar System

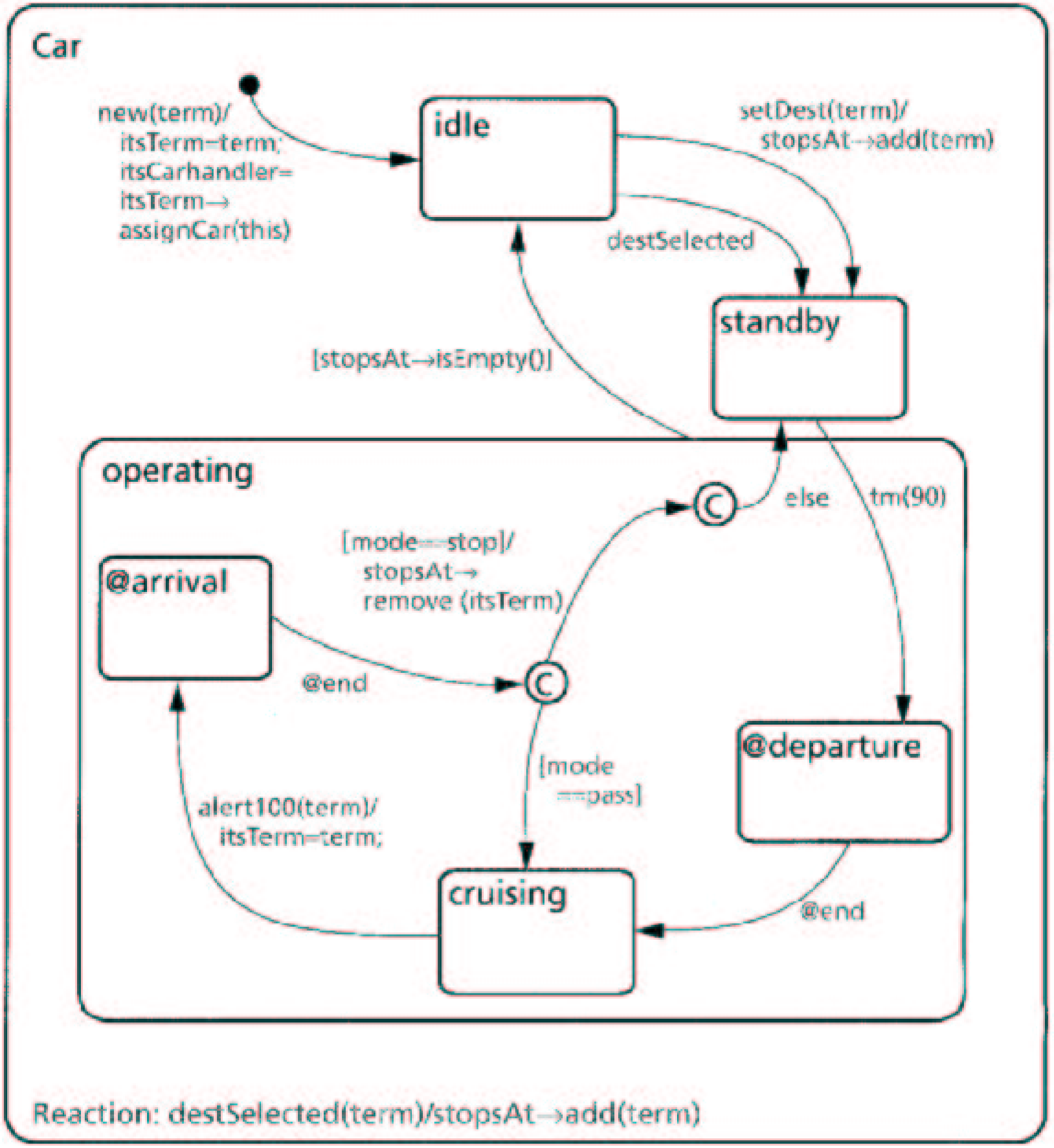


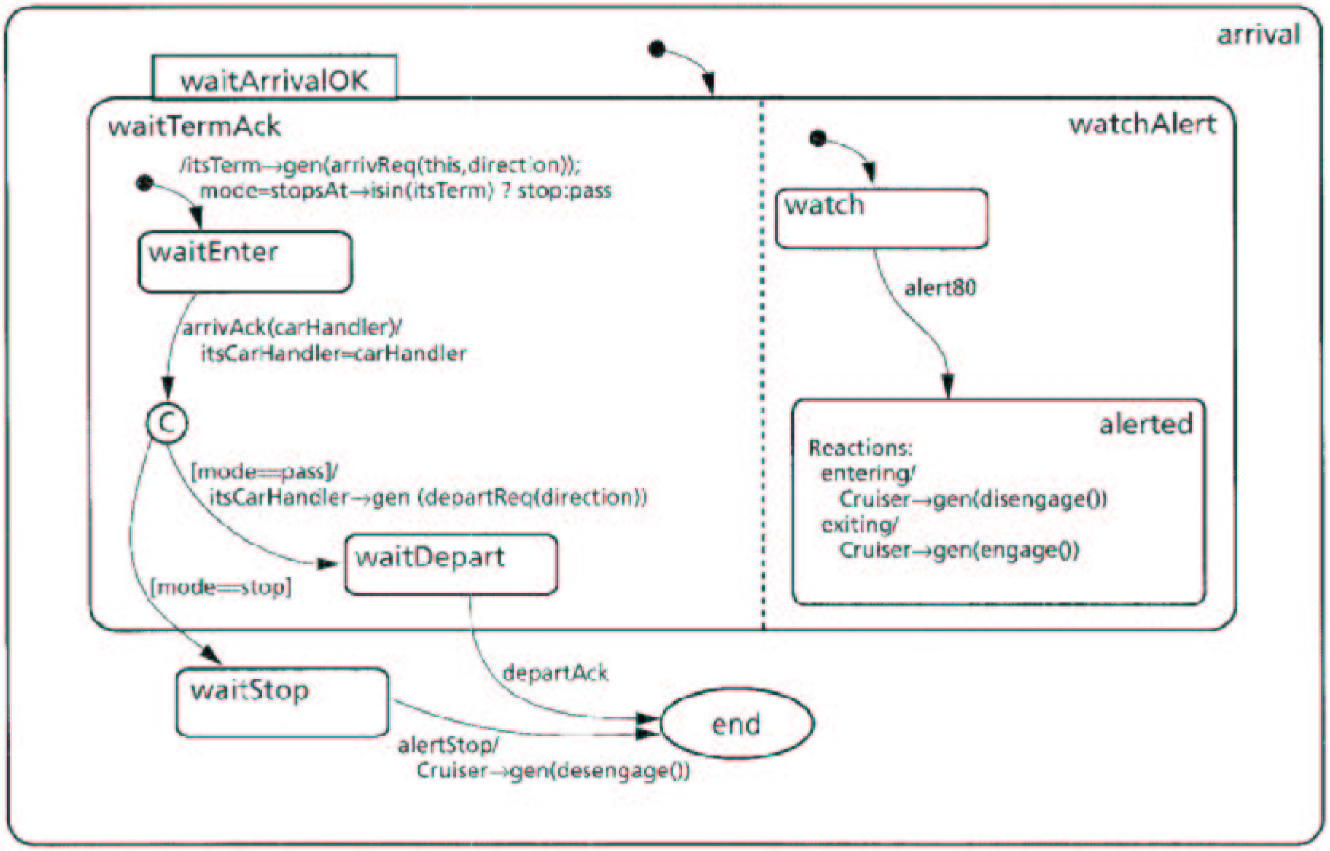
Scenarios (Use Cases)

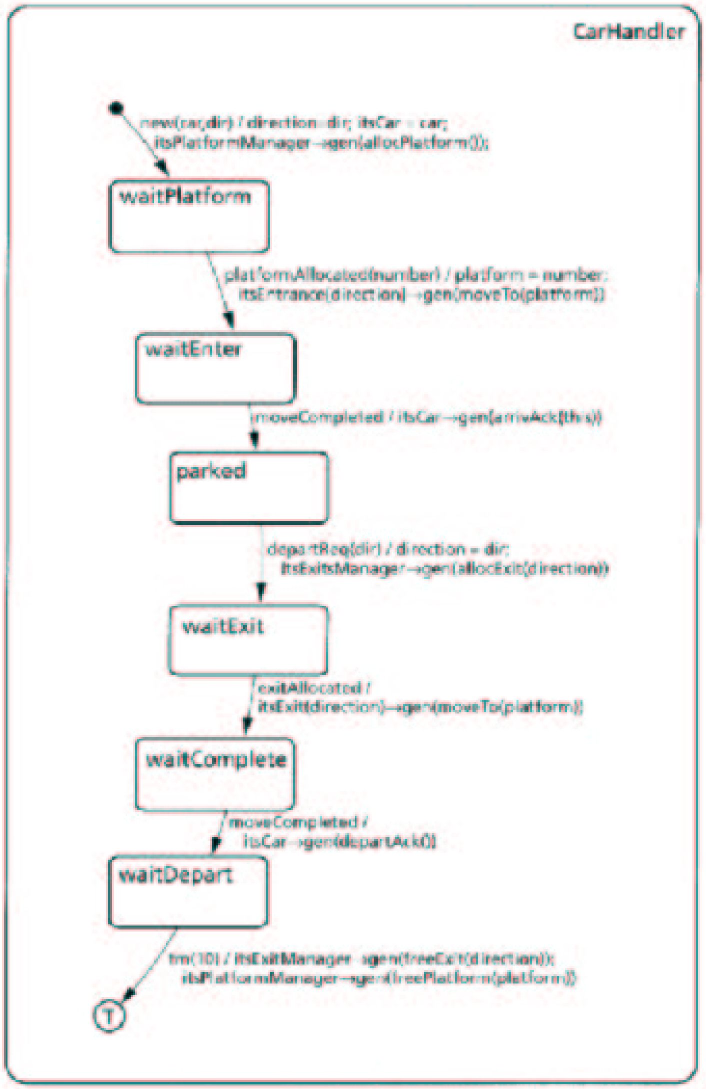
- Car approaches terminal
- Car departs from terminal
- Passenger in terminal


Terminal
6
6
1
1
1
stopsAt
Passenger
*
Control-Center
1
1
1
Car
*
(a)


Terminal
6
Platform-Manager
1
DestPanel
1
Entrance
2
Exit
2
Exits-Manager
1
Car-Handler
*
1
2
2
1
0, 1
1
1
stopsAt
Car
*
Proximity-Sensor
1
Cruiser
1
Occupancy-Sensor
1
DestPanel
1
(b)

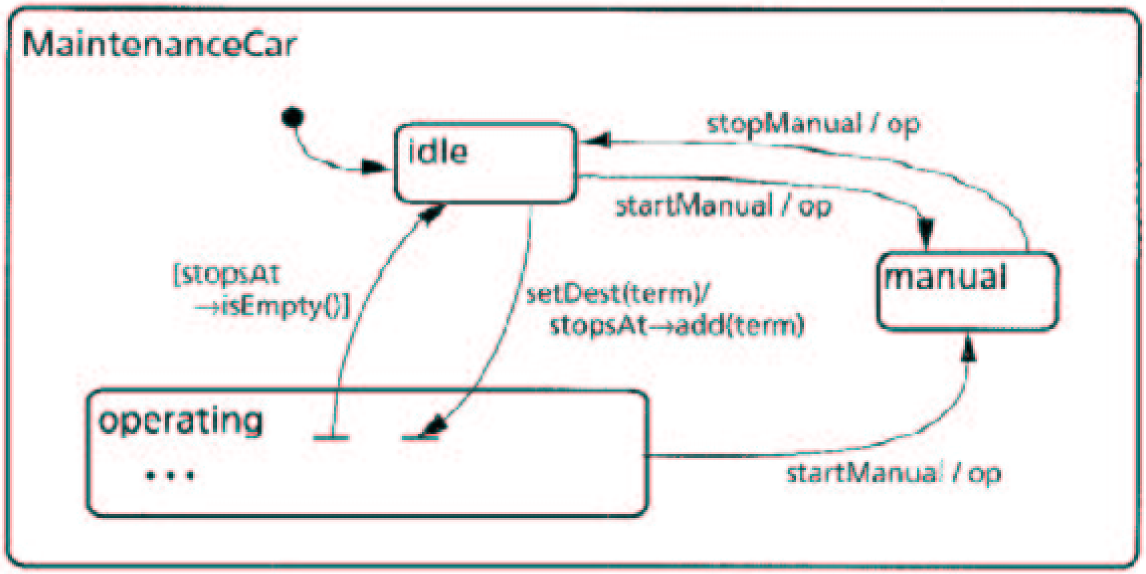

Car
new(term)/
itsTerm=term;
itsCarhandler=
itsTerm→
assignCar(this)
idle
setDest(term)/
stopsAt→add(term)
destSelected
standby
[stopsAt→isEmpty()]
operating
C
else
tm(90)
[mode==stop]/
stopsAt→
remove (itsTerm)
@arrival
@end
C
[mode
==pass]
@departure
alert100(term)/
itsTerm=term;
cruising
@end
Reaction: destSelected(term)/stopsAt→add(term)


arrival
waitArrivalOK
waitTermAck
/itsTerm→gen(arrivReq(this,direction));
mode=stopsAt→isin(itsTerm) ? stop:pass
waitEnter
arrivAck(carHandler)/
itsCarHandler=carHandler
C
[mode==pass]/
itsCarHandler→gen (departReq(direction))
waitDepart
[mode==stop]
watchAlert
watch
alert80
alerted
Reactions:
entering/
Cruiser→gen(disengage())
exiting/
Cruiser→gen(engage())
waitStop
departAck
end
alertStop/
Cruiser→gen(desengage())


CarHandler
new(car,dir) / direction=dir; itsCar = car;
itsPlatformManager→gen(allocPlatform());
waitPlatform
platformAllocated(number) / platform = number;
itsEntrance(direction)→gen(moveTo(platform))
waitEnter
moveCompleted / itsCar→gen(arrivAck(this))
parked
departReq(dir) / direction = dir;
itsExitsManager→gen(allocExit(direction))
waitExit
exitAllocated /
itsExit(direction)→gen(moveTo(platform))
waitComplete
moveCompleted /
itsCar→gen(departAck())
waitDepart
tm(10) / itsExitManager→gen(freeExit(direction));
itsPlatformManager→gen(freePlatform(platform))
T

Inheritance

- structural or behavioural
- interface subtyping
- Modify states
  - Decompose state in OR or AND components
  - Add sub-states to OR state
  - Add orthogonal components to any state


MaintenanceCar
idle
stopManual / op
startManual / op
manual
[stopsAt
→isEmpty()]
setDest(term)/
stopsAt→add(term)
operating
...
startManual / op