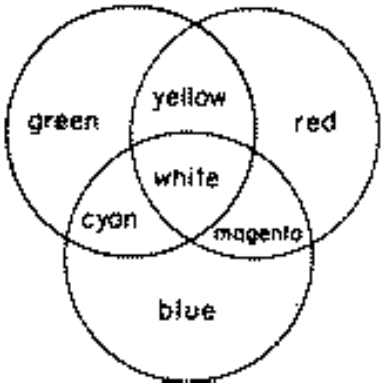
Statecharts aka Harel Charts

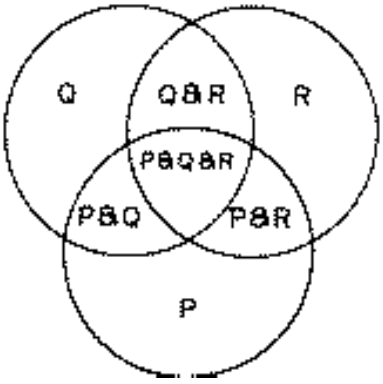
- visual formalism
- higraph based (rigour)
- diverse applications;
  in particular: concurrent systems behaviour

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

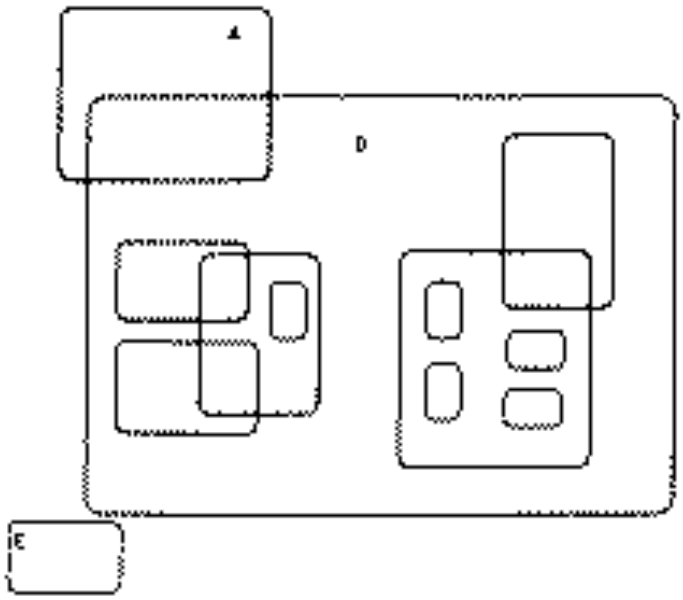




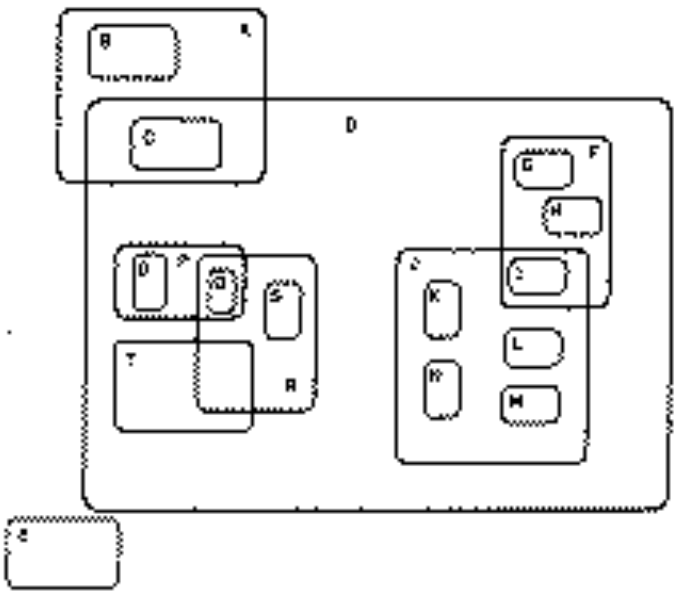
Higraphs: combining graphs and Venn diagrams

- hypergraphs
- sets + cartesian product

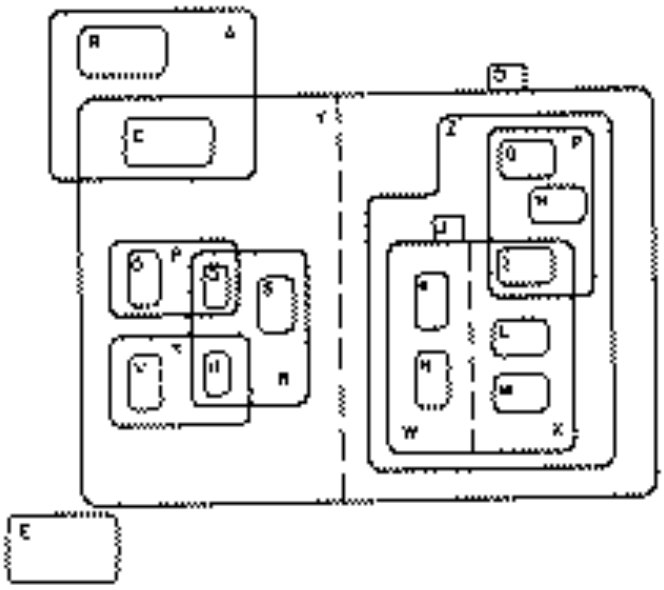
Blobs: set *inclusion, not membership*



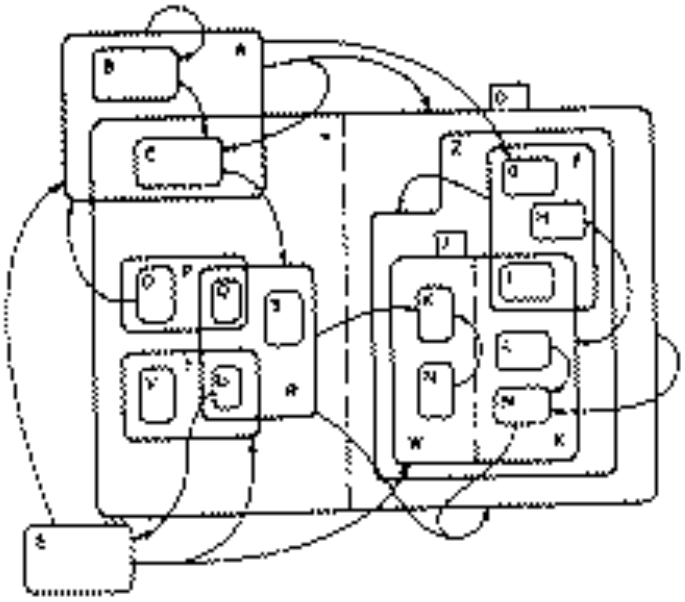
Unique Blobs (atomic sets, no intersection)



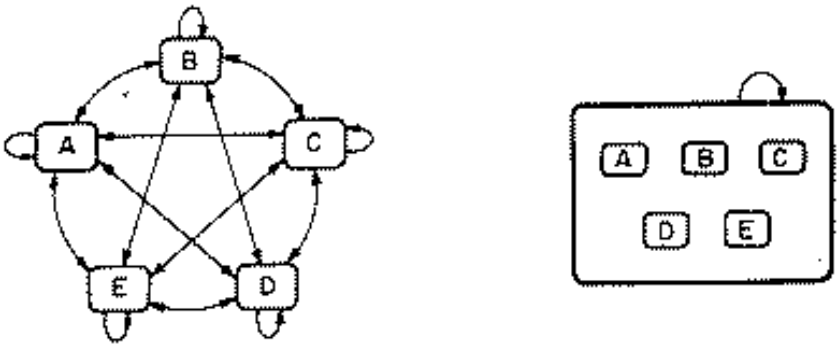
Unordered Cartesian Product: *Orthogonal* Components



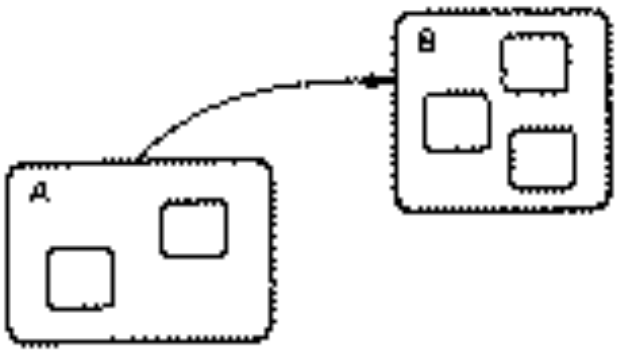
Adding (hyper) edges



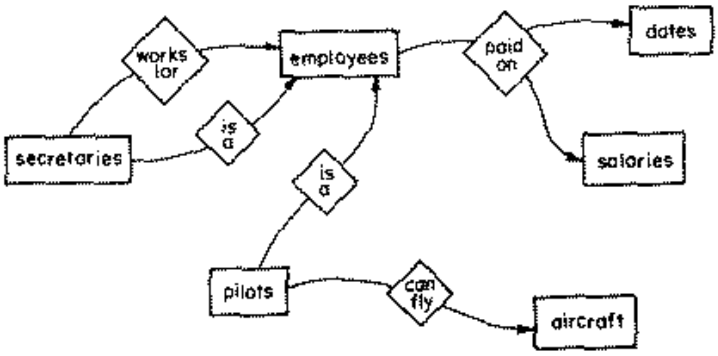
Clique Example: all connected to all



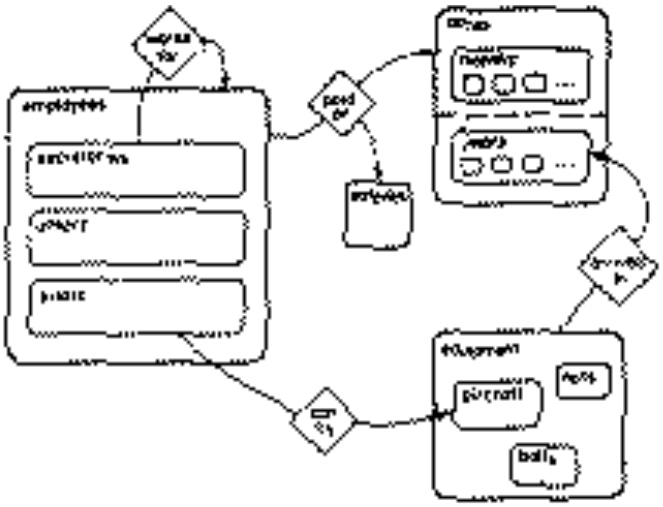
Expressiveness: class of graphs



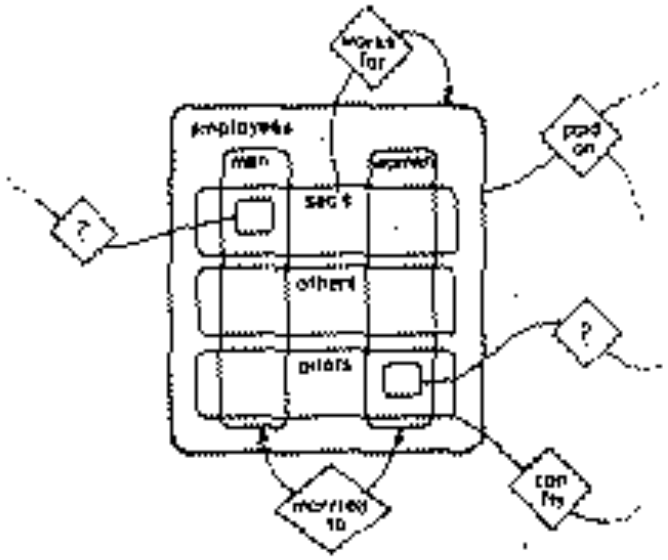
Entity Relationship Diagram (`is-a`)



Higraph version of E-R diagram



Extending the E-R diagram



Higraphs applications

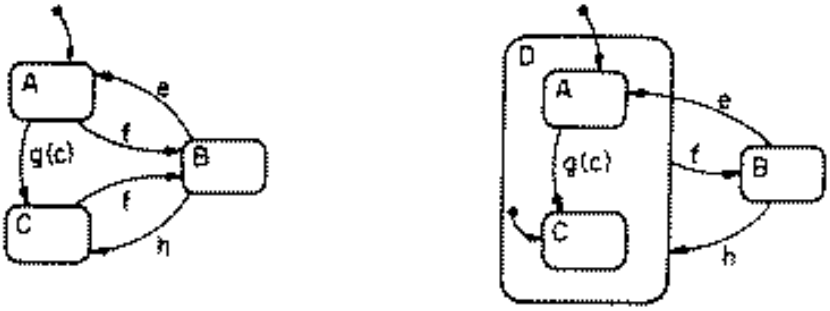
- E-R diagrams
- data-flow diagrams (activity diagrams)
  edges represent (flow of) data
- inheritance
- Statecharts

StateCharts

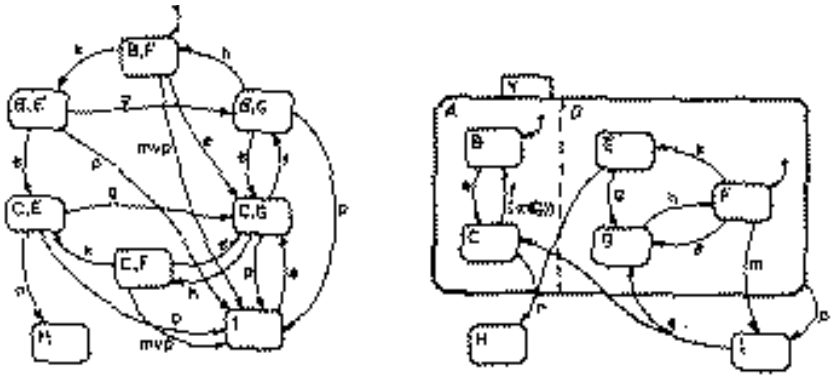
- for Reactive Systems (event driven, react to internal and external stimuli)
- like Petri Nets, CSP, CCS, sequence diagrams, . . .
- graphical but formal and rigourous for
  - analysis
  - code generation
- solve FSA problems:
  - flat
  - number of transitions
  - number of states
  - sequential

State Charts = state diagrams + depth + orthogonality + broadcast

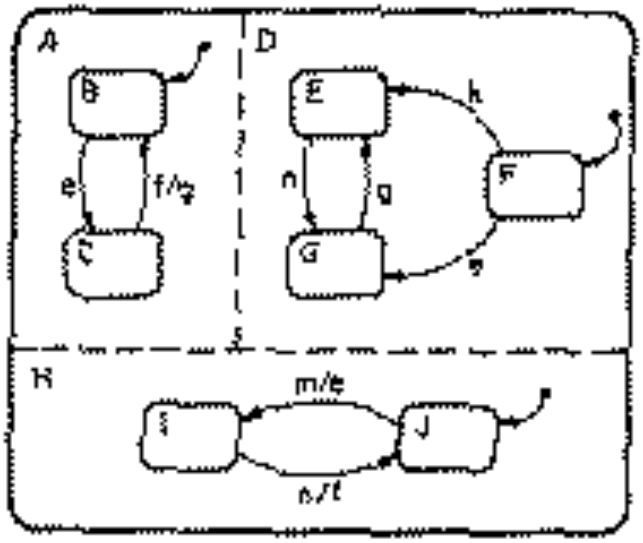
Depth (XOR)



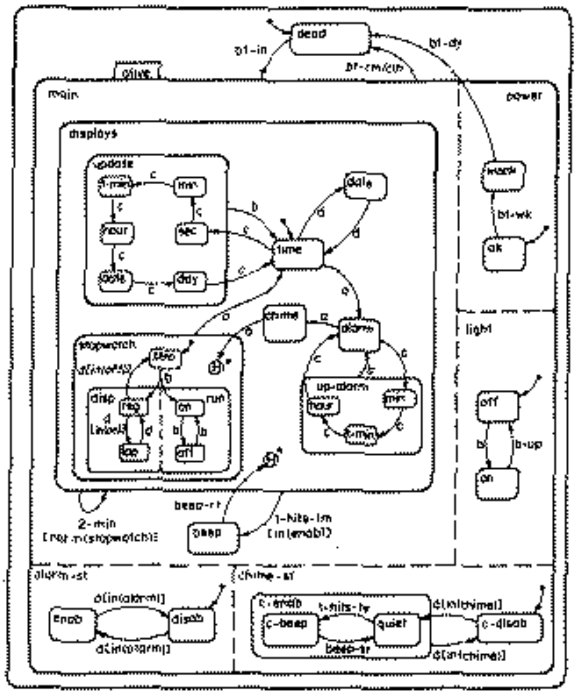
Orthogonality (AND), flattening $\rightarrow$ semantics



Broadcasting (output events)



History States



Executable Object Modelling

- analysis $\rightarrow$ use cases $\rightarrow$ sequence diagrams
- analysis $\rightarrow$ use cases $\rightarrow$ class diagrams
- $\rightarrow$ Statecharts $\rightarrow$ sequence diagrams $\rightarrow$ test use cases

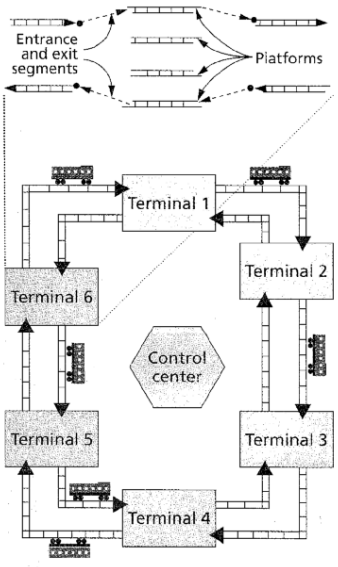
Executable Object Modelling with Statecharts

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

Executable Object Modelling with Statecharts

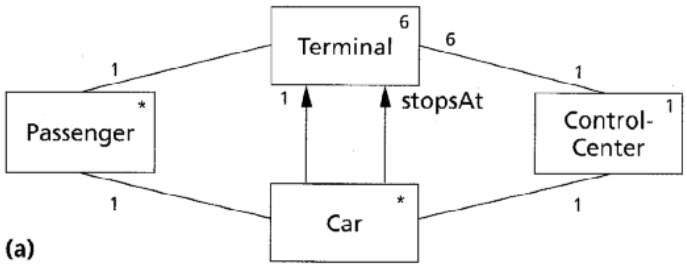
- Structure (classes, multiplicities, relationships)
  Object-model diagrams (higraph version of ER-diagrams)
- Behaviour
  Statecharts

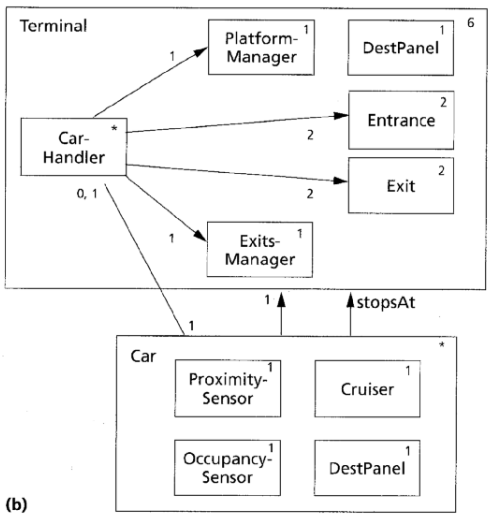
Automated Railcar System

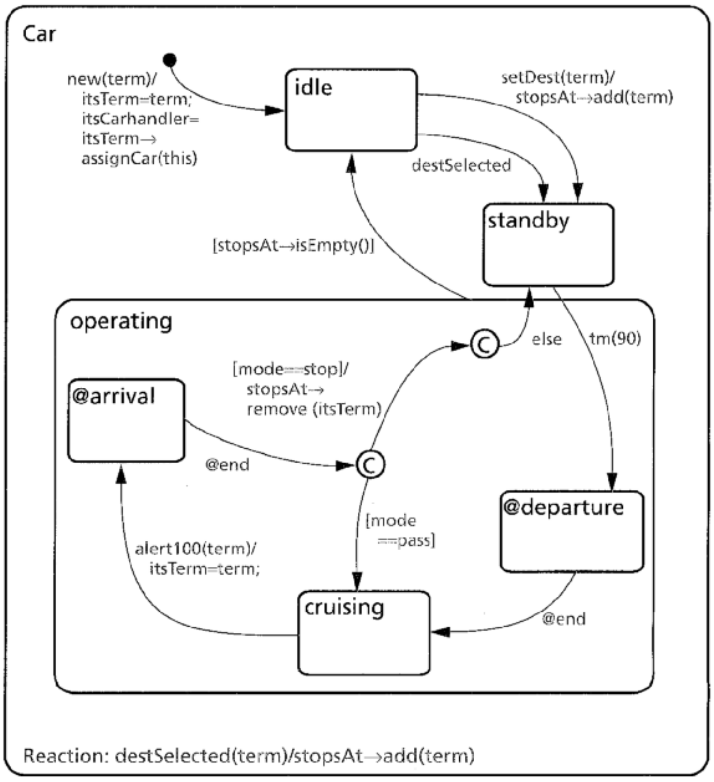


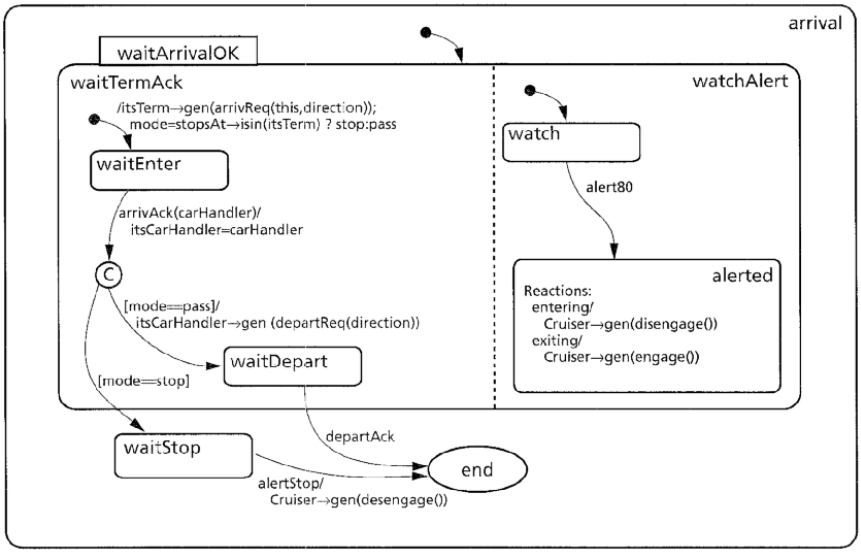
Scenarios (Use Cases)

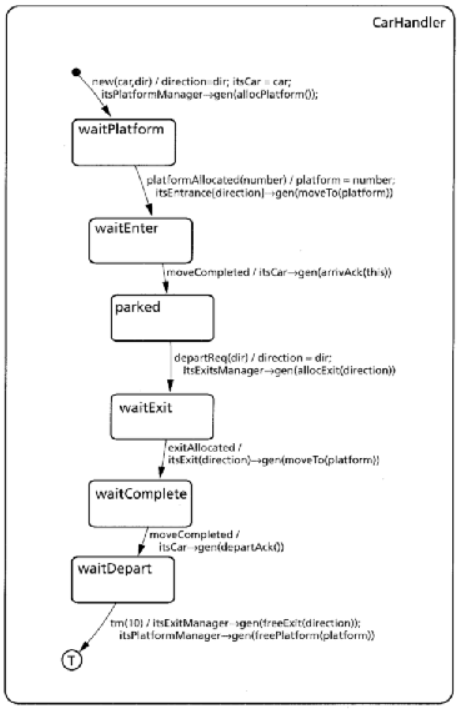
- Car approaches terminal
- Car departs from terminal
- Passenger in terminal


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


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

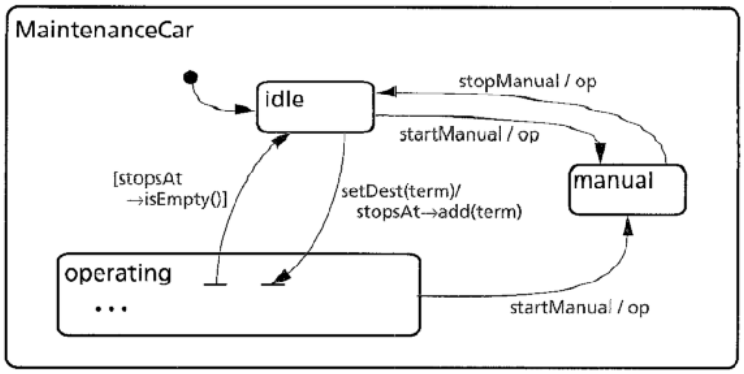

Car
new(term)/
itsTerm=term;
itsCarhandler=
itsTerm→
assignCar(this)
idle
setDest(term)/
stopsAt→add(term)
destSelected
standby
[stopsAt→isEmpty()]
operating
else
tm(90)
[mode==stop]/
stopsAt→
remove (itsTerm)
@arrival
@end
@departure
[mode
==pass]
alert100(term)/
itsTerm=term;
cruising
@end
Reaction: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


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



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