Executable Object Modelling

- analysis ⇒ use cases ⇒ class diagrams
- analysis ⇒ use cases ⇒ (message) sequence diagrams
- ⇒ Object-model diagrams
- ⇒ Statecharts ⇒ sequence diagrams ⇒ *test* use cases

Executable Object Modelling with Statecharts

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

Executable Object Modelling with Statecharts

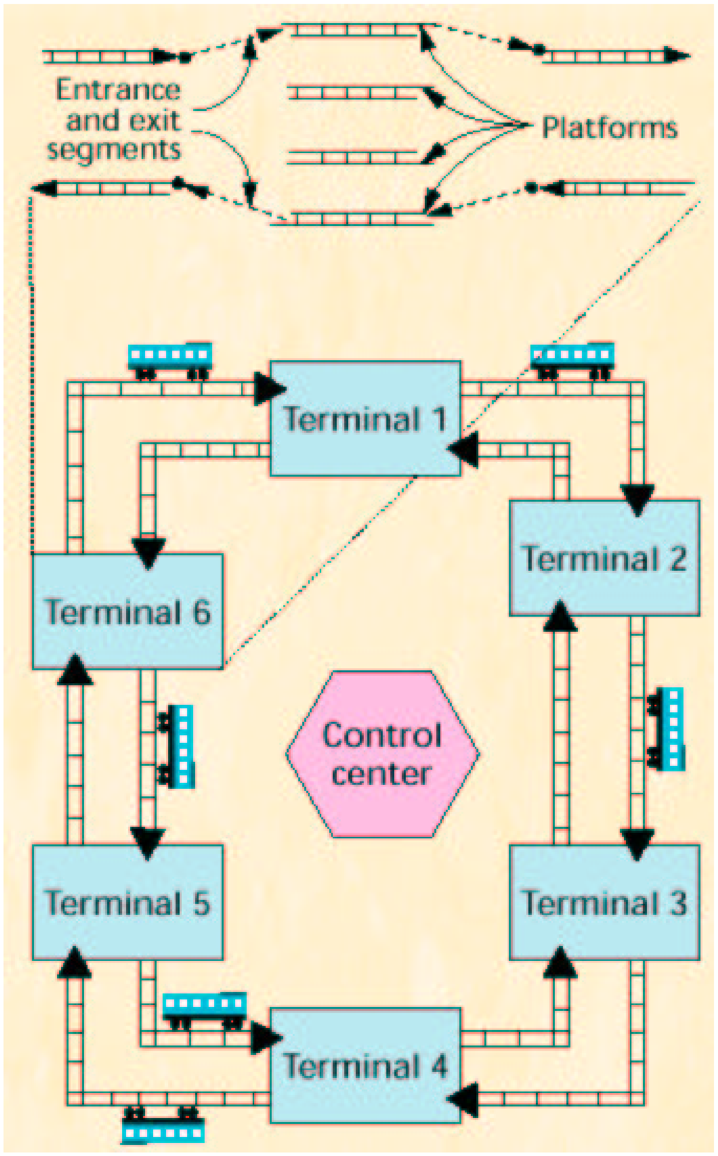
- Structure (classes, multiplicities, relationships)

  $\Rightarrow$ Object-model diagrams (higraph version of ER-diagrams)

- Behaviour

  $\Rightarrow$ StateCharts

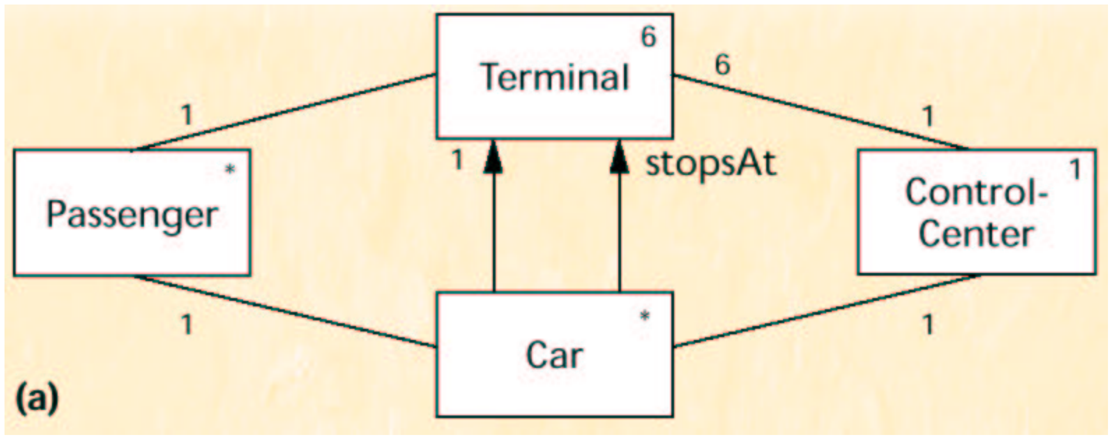
Automated Railcar System: Physical View


Entrance and exit segments
Platforms
Terminal 1
Terminal 2
Terminal 3
Terminal 4
Terminal 5
Terminal 6
Control center

Scenarios (Use Cases)

1. Car approaches terminal
2. Car departs from terminal
3. Passenger in terminal

Toplevel object-model diagram

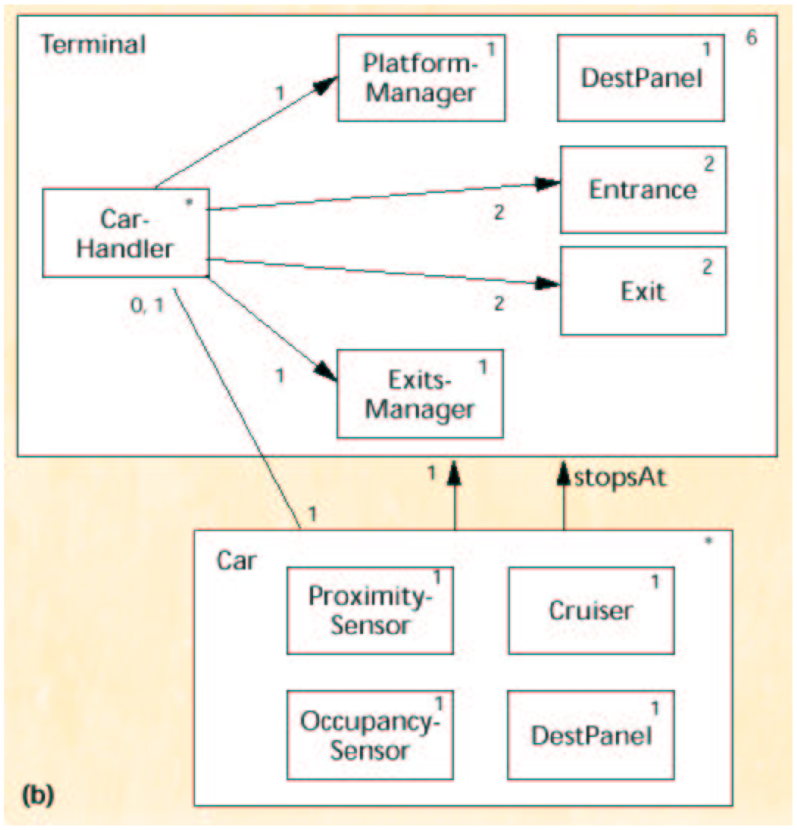


- object classes
- object multiplicities
- structural relationships (including navigatability and arity)

Object Navigation, Creation/Initialization

- navigatability
  - no relation name $\Rightarrow$ `its`
  - `Passenger->itsCar->stopsAt`
  - toplevel: `System->itsTerminal[1:6]`
- Code synthesis: creation/initialization $+$ dynamics over time
- Object multiplicity
- Associations
  1. unambiguous: multiplicities match
  2. ambiguous but bounded: any subset
  3. unworkable: canonical mappings or user defined (scripts)

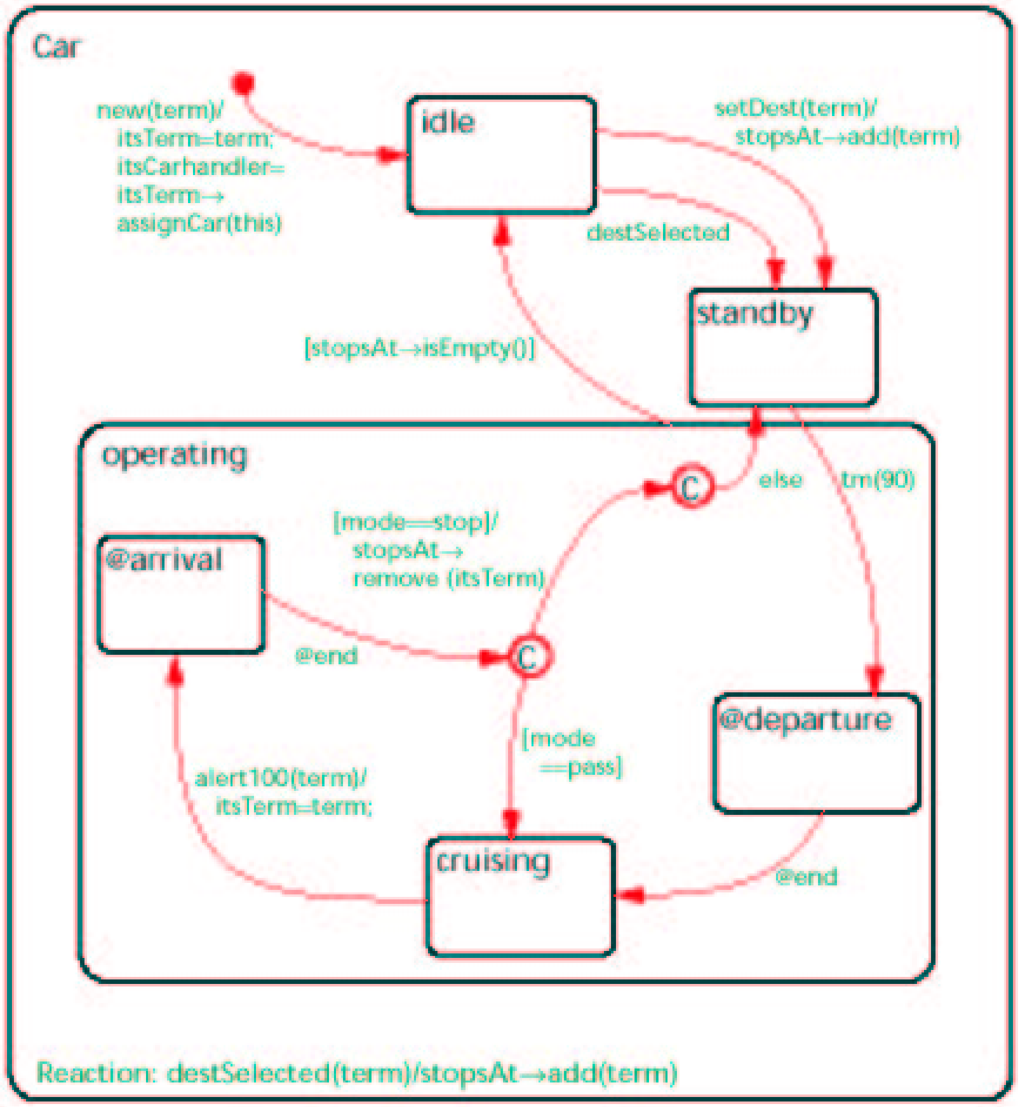
Zoom out: aggregation



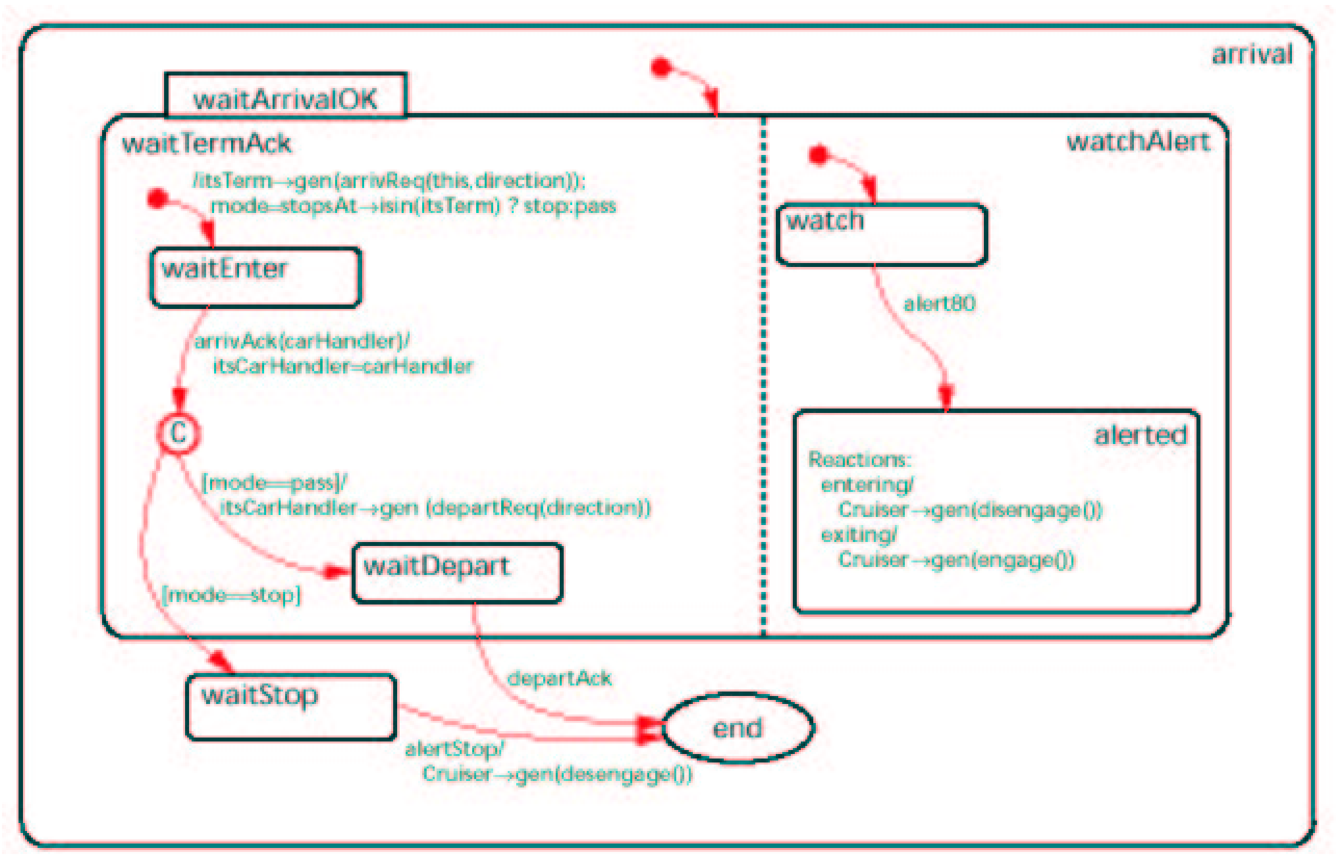
Dynamics of Object Communication and Collaboration

1. Objects generate events which are queued
2. Objects can directly invoke an operation/method

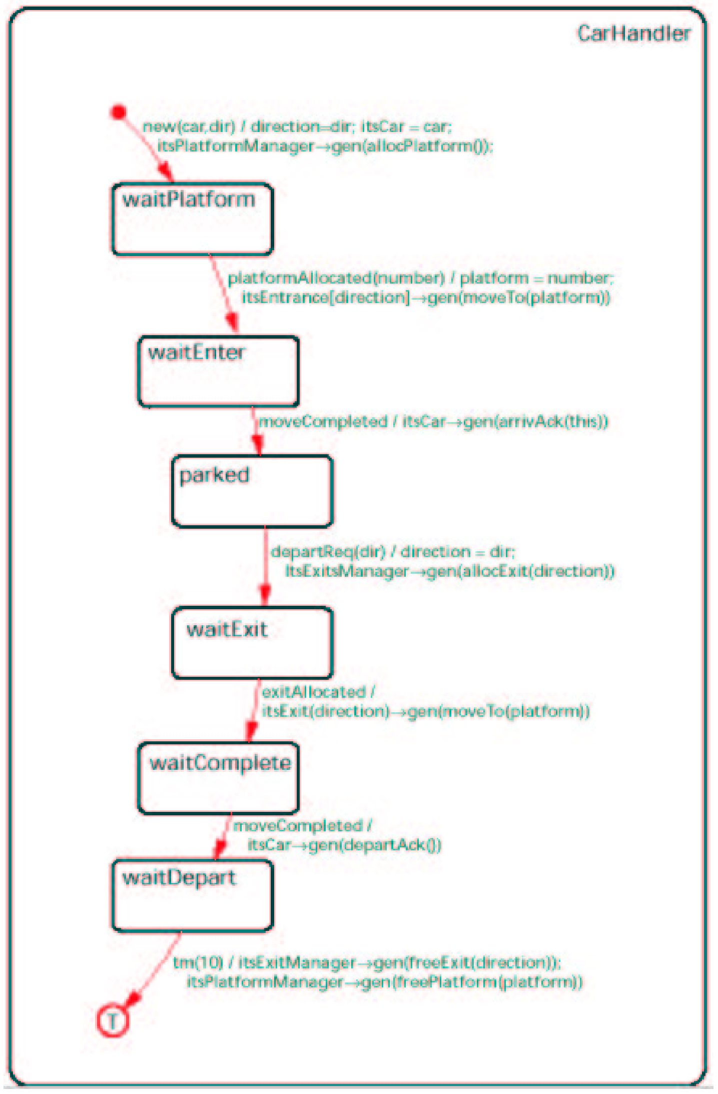
Car dynamics


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

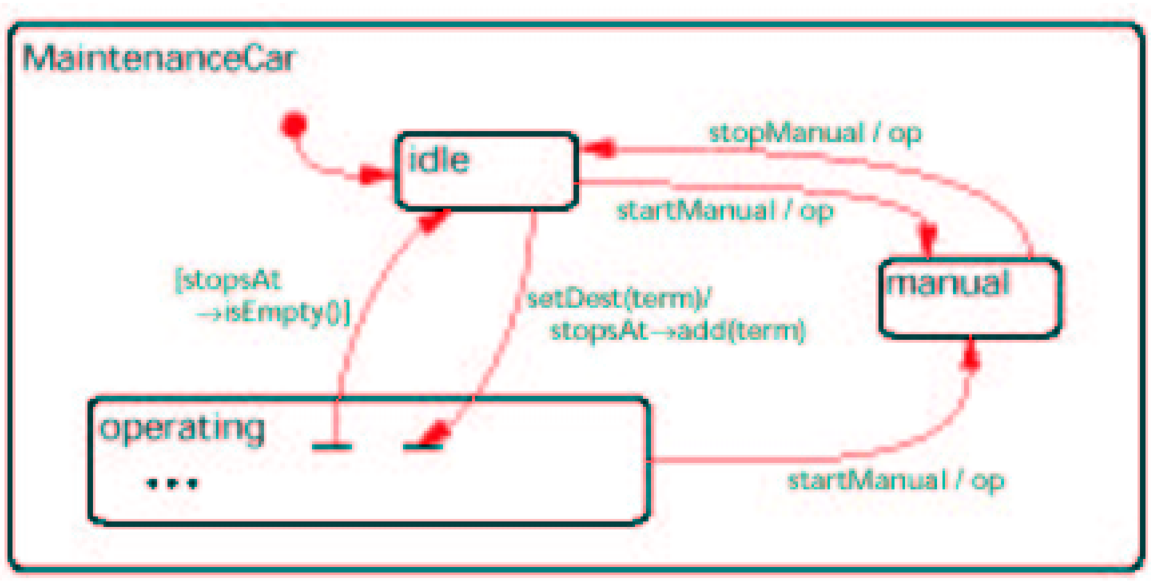
Arrival dynamics



CarHandler lifecycle


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

Zooming


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

Inheritance

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