Executable Object Modelling

- analysis $\Rightarrow$ use cases $\Rightarrow$ class diagrams
- analysis $\Rightarrow$ use cases $\Rightarrow$ (message) sequence diagrams
- $\Rightarrow$ Object-model diagrams
- $\Rightarrow$ Statecharts $\Rightarrow$ sequence diagrams $\Rightarrow$ test use cases
Executable Object Modelling with Statecharts

- OO development: intuitive/graphical \textit{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: Physical View
Scenarios (Use Cases)

1. Car approaches terminal
2. Car departs from terminal
3. Passenger in terminal
Use Case:  
Car approaches terminal

When the car is 100 yards from the terminal, the system allocates it a platform and an entrance segment, which connects it to the incoming track.

If the car is to pass through without stopping, the system also allocates it an exit segment.

If the allocation is not completed within 80 yards from the terminal, the system delays the car until all is ready.
Use Case:
Car departs from terminal

A car departs the terminal after being parked for 90 seconds. The system connects the platform to the outgoing track via the exit segment, engages the car’s engine, and turns off the destination indicators on the terminal destination board. The car can then depart unless it is within 100 yards of another car; if so, the system delays departure.
A passenger in a terminal wishes to travel to some destination terminal, and there is no available car in the terminal travelling in the right direction. The passenger pushes the destination button and waits until a car arrives. If the terminal contains an idle car, the system will assign it to that destination. If not, the system will send a car in from some other terminal. The system indicates that a car is available with a flashing sign on the destination board.
Toplevel object-model diagram

- object classes
- object multiplicities
- structural relationships (including navigability and arity)
Object Navigation, Creation/Initialization

- navigatability
  - no relation name ⇒ its
  - Passenger->itsCar->stopsAt

- 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

![Diagram of object modeling with state charts]

October, 2003

hv@cs.mcgill.ca

Object Modelling with State Charts
Dynamics of Object Communication and Collaboration

1. Objects generate events which are queued
   \[ \text{serverObject} \rightarrow \text{gen(event(<params>))} \]

2. Objects can directly invoke an operation/method
   \[ \text{serverObject} \rightarrow \text{method(<params>)} \]
Car dynamics
Car

idle

setDest(term)/
itsTerm→add(term)

destSelected

standby

[stopsAt→isEmpty()]

operating

@arrival

[mode→stop]/
stopsAt→ remove (itsTerm)

@end

@departure

reaction: destSelected(term)/stopsAt→add(term)

@cruising

[mode→pass]

@end

new(term)/
itsTerm=term;
itsCarhandler= itsTerm→ assignCar(this)
Arrival dynamics
CarHandler lifecycle

```
newCarCar / direction-dir; itsCar = car;
   itsPlatformManager = gen(alloctPlatform());

waitPlatform
  platformAllocated(number) / platform = number;
  itsEntrance(direction) = gen(moveTo(platform))

waitEnter
  moveCompleted / itsCar = gen(arrivalAck(this))

parked
  departReq(dir) / direction = dir;
  itsExitManager = gen(alloctExit(direction))

waitExit
  exitAllocated / itsExit(direction) = gen(moveTo(platform))

waitComplete
  moveCompleted / itsCar = gen(departAck())

waitDepart
  tm(IQ) / itsExitManager = gen(freeExit(direction));
  itsPlatformManager = gen(freePlatform(platform))
```
Zooming

```
MaintainanceCar

idle
[stopsAt
→isEmpty()]

manual

operating

stopManual / op
startManual / op
startManual / op

setDest(term)/
stopsAt→add(term)
```
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