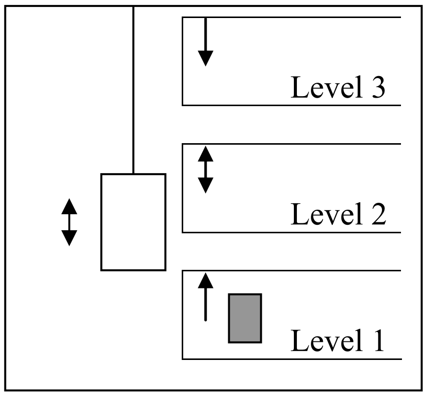
MODEL OF A ***FREIGHT ELEVATOR SYSTEM***

The proposed system to be modeled is the behavior of a freight elevator system that can be found in many different industry applications other than the transportation of people (mining, warehouses, cargo companies, etc.).

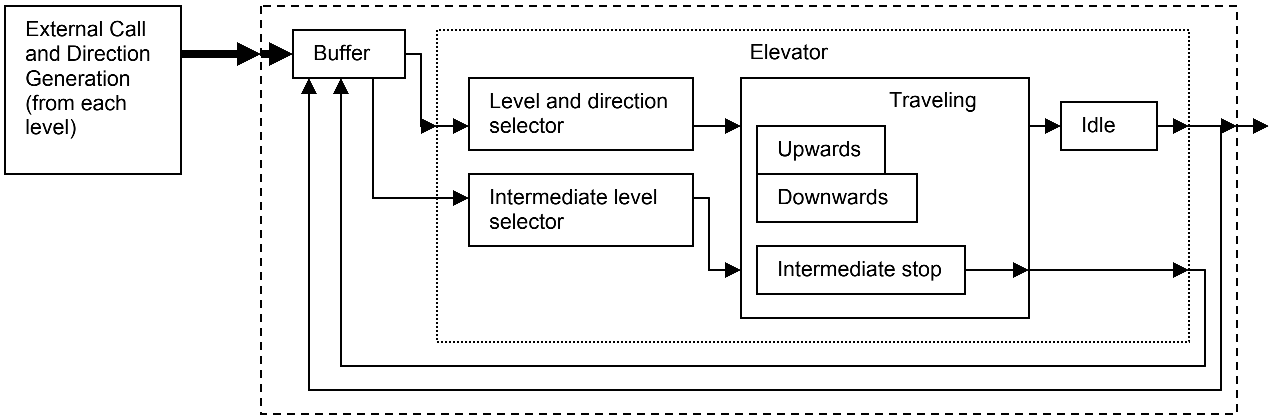
Description of the System

The system (elevator) is in *idle* state until it receives a call from any of the floors, once it has acknowledge the call (and it's a valid call), the elevator system changes its state to *traveling*, after reaching the floor where the call originate it changes its state to *loading.* After this state and after pressing the desired floor button the elevator goes to the *traveling* state once more (cannot go to travel state if in the same floor as desired level), if before it goes to the *traveling* state, there is another call in the same direction of the travel (upward – downward) from one floor that is in the path of the elevator (for example the elevator is loading cargo in the 1 level that's going to level 3 but there is a call from level 2) the elevator should make a stop in the middle level to load more cargo and then go to the desired level. But if the second call is in the opposite direction, the elevator system should attend that call after delivering the cargo in the desired level.

A schematic diagram is presented.



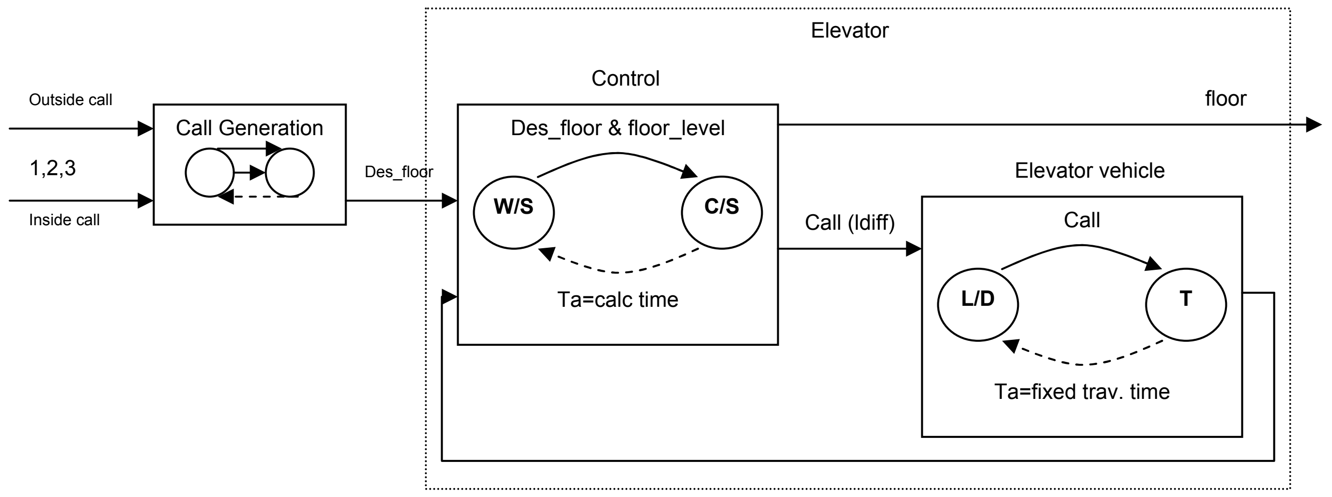
The state variables for the system are: *idle*, *loading-delivering*, *traveling*; the internal states would be (there would be an increase of two more states in the actual implementation): X level call buffering, desired level selection, traveling direction (up – down) and middle stop generation. An approximate model of the system is:


External Call and Direction Generation (from each level)
Buffer
Elevator
Level and direction selector
Intermediate level selector
Traveling
Upwards
Downwards
Intermediate stop
Idle

This model is a general approximate model of the system to be modeled and to be simulated; there will be some variations in the implementation of the proposed model. The labels inside each box tried to be self explanatory, almost every sub model can be constructed as a coupled model from atomic entities, and the higher levels are themselves part of highest levels.

Freight Elevator Model – DEVS Implementation

The model given can be very simplified and arranged in such a form that it will be easy to extract atomic and coupled models, this arrangement will use FSM and graphical DEVS specification. Once the model is extracted it will be easier to add a lot more real details to the model.



W/S -> Wait and Store calls
C/S -> Calculate and Send time multiplier [call(des_floor-floorlevel)]
L/D -> Loading-Delivering State (idle state)
T -> Traveling state [call(ta*call(des_floor-floorlevel))]

Stated that way it is easy to obtain the atomic and couples models of each component, which will be:

<u>Elevator Vehicle</u>

Elevator_v = $<X,Y,S,\delta_{int},\delta_{ext},\lambda,t_a>$

X={call}
Y={floor}
S={loading/delivering, traveling}
δ_{int}=loading/delivering
δ_{ext}=call
λ=floor
$t_a = t_a{}^* \,|\text{desired\_floor-floor\_level}|$

<u>Elevator Control</u>

Elevator_Control = $<X,Y,S,\delta_{int},\delta_{ext},\lambda,t_a>$

X={des_floor, floor}
Y={call(difference)}
S={wait/store, calculate/send}
δ_{int}=calc time
δ_{ext}=des_floor & floor_level
λ=call(ldiff)
$t_a = t_a$

Elevator Coupled Model

Elevator = <X,Y,D,{M_i},{I_i},{Z_{ij}},select>

X={des_floor}
Y={floor}
D=
{M_i}={Elevator_control,Elevator_vehicle}
{I_i}: I(Elevator_control)=Elevator_vehicle
I(Elevator_vehicle)={Elevator_control, self}
{Z_{ij}}: Z(elevator_control)=Elevador_vehicle
Z(Elevator_vehicle)=Elevator_control
Z(Elevator_vehicle)=self
select=({elevator_control,elevator_vehicle})=elevator_control

Call Generador

CGEN = <X,Y,S,δ_{int},δ_{ext},λ,t_a>

X={insidecall, outsidecall}
Y={des_floor}
S={wait, senddesfloor}
δ_{int}=wait
δ_{ext}={insidecall, outsidecall}
λ=des_floor
t_a= 0

Freight Elevator Coupled Model

Elevator = <X,Y,D,{M_i},{I_i},{Z_{ij}},select>

X={insidecall, outsidecall}
Y={floor}
{M_i}={CGEN,Elevator}
{I_i}: I(CGEN)=Elevator
I(Elevator)={self}
{Z_{ij}}: Z(CGEN)=Elevator
Z(Elevator)=self
select= ({CGEN,elevator_control,elevator_vehicle})=elevator_control
({elevator_control,elevator_vehicle})=elevator_control

Test Strategies

Basically the Freight Elevator system can be represented as a processor performing one task at any given time (carry cargo between floors), the principal different characteristics are: the travel time between floors is the same, the number of floors is fixed but can be chosen arbitrarily, the call can be generated outside (door) or inside the vehicle (desired floor).

Then the call generation block should behave as an OR operand to the system, meaning that the system is not aware of from where the call has been generated, so feeding the block with different floor calls from inside or outside the vehicle, the output for this should be the floor that generated the call (outside 2 or inside 2 will have the same output).

The elevator control block once feed with the desired floor should perform the distance calculation based in the modulus of the difference of the actual floor and the desired floor (output, call(ldiff)) IF the elevator is in the idle state (loading/delivering), if not the block should store the desired floor information until the elevator vehicle is ready for another trip.

For the elevator vehicle the testing strategy is straight forward, since it should be in traveling state (busy) once it gets a call from one floor, and it should leave this state once a fixed time multiplied by the difference of floors has elapsed (ta*call(diff)).

The test of the Elevator system is basically is slightly different than that of the control block as there is only one input to the system, once a desired floor is introduced and the elevator is free, it should output the desired floor after a certain time (we'll assume that the traveling time between floors is 1 min), that will become the level floor for the next trip. If it gets a call in between trips the elevator block should be capable of store the desired floor and start the trip by itself once free, after finishing all the trips it should remain in the last desired floor for indefinite time.

The test of the Freight Elevator system is similar to the Elevator with the difference that there will be two inputs and the call generated inside the elevator will have preeminence over the call generated outside of it. So the output in the case of two almost simultaneous calls should be to attend first the call generated inside the vehicle and then initiate the trip to the desired floor generated by the outside call.

<u>Freight Elevator – Results of the Simulation on CD++ Implementation</u>

<u>Elevator Vehicle</u>

Due to the simplified model given the results can be easily analyzed from the output files of the simulation as well as the log files generated by the simulator engine.

For the Elevator Vehicle we have (assuming that the travel time between floors is 1 min):

Evehicle.ma - Evehicletest.ev	Evehicle.out
`00:10:00:000 cdiff 3`	`00:13:00:000 diff 3`
`00:21:00:000 cdiff 2`	`00:23:00:000 diff 2`
`00:24:30:000 cdiff 1`	`00:25:30:000 diff 1`
`00:28:00:000 cdiff 1`	`00:29:00:000 diff 1`
`00:29:00:001 cdiff 3`	`00:32:00:001 diff 3`
`00:32:10:000 cdiff 2`	`00:34:10:000 diff 2`

According to our testing assumption the value that enters the block is the difference between the desired level and the actual level times the fixed travel time, thus the elevator will be ready once it has finished the current trip, this is clearly seen comparing the initial time, the input value and the time of the output (the output value is just for testing purposes, also for this purposes there are input values greater than 2, which is the maximum value that the difference can have in a 3 level freight elevator).

<u>Call Generation</u>

The generator can receive calls from the inside or the outside of the vehicle, and feed this data to the controller, but is capable of distinguish which call has priority over the other (the calls generated inside have priority over the calls generated outside), the results for the model:

Ecall.ma - Ecall.ev	Ecall.out
`00:05:00:000 outside_call 2`	`00:05:00:000 call_gen 2`
`00:11:00:000 inside_call 3`	`00:11:00:000 call_gen 3` ←
`00:11:00:000 outside_call 1`	
`00:15:00:000 outside_call 2`	`00:15:00:000 call_gen 2`
`00:18:00:000 inside_call 2`	`00:18:00:000 call_gen 2`
`00:20:00:000 inside_call 3`	`00:20:00:000 call_gen 3`
`00:25:00:000 outside_call 1`	`00:25:00:000 call_gen 1`
`00:45:00:000 outside_call 2`	`00:45:00:000 call_gen 3` ←
`00:45:00:000 inside_call 3`	

The calculation time is instantaneous (in fact it takes a very small time, but compared with the speed of the elevator is almost zero), in the result file it can be seen that the logic of this controller assigns priority to the calls generated inside the vehicle if for some reason two events (inside and outside calls) occur at the same time.

Elevator Control

For the test of this atomic model there are some assumptions:

a) the value associated with “acall” is the desired floor level
b) the fback value is the multiplier value (the basic time travel unit is 1 min, between two consecutive floors) that is returned by the elevator vehicle once it has finished the travel.
c) In the output file the timem value is the multiplier value that will be send to the elevator vehicle, after the this value times 1 min, the Econtrol block should display the actual floor, this can be seen in the value associated to the floor port.
d) The elevator always starts idle at level 1.

Econtrol.ma - Econtrol.ev	Econtrol.out
`00:11:00:000 acall 2`	`00:11:00:000 timem 1`
`00:12:00:000 fback 1`	`00:12:00:000 floor 2`
`00:15:00:000 acall 6`	`00:15:00:000 timem 4`
`00:20:00:000 fback 4`	`00:20:00:000 floor 6`
`00:27:00:000 acall 1`	`00:27:00:000 timem 5`
`00:28:00:000 acall 2`	`00:30:00:000 floor 1`
`00:30:00:000 fback 5`	`00:31:00:000 timem 1`
`00:32:00:000 fback 1`	`00:32:00:000 floor 2`
`00:31:00:000 acall 6`	`00:33:00:000 timem 1`
`00:36:00:000 acall 3`	`00:35:10:000 floor 6`
`00:40:00:000 acall 2`	`00:36:00:000 timem 4`
`00:35:10:000 fback 5`	`00:38:00:000 floor 3`
`00:38:00:000 fback 3`	`00:40:00:000 timem 3`
`00:41:00:000 fback 1`	`00:41:00:000 floor 2`

Elevator Coupled Model

The test of this model, based on the results of the vehicle and control results, shows the behavior of the vehicle when the Econtrol block takes control of the travel times of the vehicle. The destination floor should be reached after the travel time set by the Econtrol DEVS atomic model, as before the same assumptions are considered for this case.

Elevator.ma - Elevatortest.ev	Elevator.out
`00:05:00:000 acall 2`	`00:06:00:000 floor 2`
`00:10:00:000 acall 3`	`00:11:00:000 floor 3`
`00:15:30:000 acall 1`	`00:17:30:000 floor 1`
`00:20:00:000 acall 1`	`00:20:00:000 floor 1`
`00:23:00:000 acall 2`	`00:24:00:000 floor 2`
`00:25:00:000 acall 3`	`00:26:00:000 floor 3`
`00:27:01:000 acall 1`	`00:29:01:000 floor 1`
`00:29:30:000 acall 2`	`00:30:30:000 floor 2`
`00:34:00:000 acall 1`	`00:35:00:000 floor 1`
`00:36:00:000 acall 3`	`00:38:00:000 floor 3`

Freight Elevator Complete Coupled Model

The model behaves as expected, introducing values that represent calls generated from inside and outside the vehicle, for example a cargo that needs to use the elevator, have to wait the time that the elevator vehicle takes to go from one level to the originating level plus the time that it takes to go from the originating level to the desired level.

The chart can be read as follows: at 00:12:00:00 the first event is generated by a level call at level 3 (elevator in level 1), travel time to attend the event (2 units, 1 unit = 1 min, thus 2 min), cargo load at level 3, second event at 00:15:00:000, desired level generated by a call inside the vehicle (level 1), as a result the vehicle delivers the cargo after completing 2 travels (4 min), assuming that the cargo was loaded in a 1 min time window. After that the same assumptions can be made for every pair of data.

At time stamp 21:00:000 there are two simultaneous calls generated inside and outside the vehicle, as in real situation the elevator goes for the inside call, choosing the inside call over the outside call.

FElevator.ma - FE_test.ev	Ecall.out
00:12:00:000 outside_call 3	00:14:00:000 floor 3
00:15:00:000 inside_call 1	00:17:00:000 floor 1
00:18:00:000 outside_call 2	00:19:00:000 floor 2
00:21:00:000 inside_call 3	00:22:00:000 floor 3
00:21:00:000 outside_call 2	00:27:00:000 floor 1
00:25:00:000 inside_call 1	00:30:00:000 floor 3
00:28:00:000 outside_call 3	00:37:00:000 floor 1
00:30:00:000 inside_call 1	00:40:00:000 floor 2
00:35:00:000 outside_call 2	00:45:00:000 floor 1
00:39:00:000 inside_call 1	00:56:00:000 floor 3
00:44:00:000 outside_call 3	00:58:00:000 floor 2
00:54:00:000 inside_call 2	
00:57:00:000 outside_call 1	

Conclusion

One of the main advantages of the DEVS modeling tool is that it allows the modeling of very complicated real systems using a deterministic approach, this modeling scheme can be used to model a lot of process that would take too much time and effort using differential equations or other discrete or continuous methods.

We have constructed a fairly simple system, but knowing the possible advantages of the DEVS formalism and the modeling tool, there is possible to make the system as real as we would like, including all types of discrete sensors and basing the behavior in a discrete event system.