HYBRID SIMULATION MODELING OF HOIST DOWN-PEAK OPERATIONS IN CONSTRUCTION SITES

Shousheng Xiang[a], Mehrdad Arashpour[b] and Ron Wakefield[b]

[a]School of Management Engineering, Xi'an University of Finance and Economics, Xi'an, China
[b]School of Property, Construction and Project Management, RMIT University, Melbourne, Australia
E-mail: xiangshousheng@126.com, mehrdad.arashpour@rmit.edu.au, ron.wakefield@rmit.edu.au

Abstract – The **down-peak period is an important aspect of hoist operations in construction projects. To make the vertical transportation service more efficient, the operation parameters of down-peak should be modeled precisely in the process of planning. A simulation model is developed to analyze the operations of elevator in down-peak. The model incorporates hybrid use of discrete-event simulation and agent-based modeling to provide a robust methodology to analyze the two most important parameters for elevator operation planning: time spent and average waiting time. The developed model is validated under several conditions, and its usage is expanded to random situations.**

Keywords –

Agent-based modeling, Automation, Building projects, Construction management, Discrete event simulation, Elevator planning, Robustness, Process visualization, Validation

1 Introduction

Research on the physics and dynamics of construction hoists has attracted the attention of many researchers [1-2]. Now with the increase in high-rise construction, hoists have become a key important vertical transport vehicle for workers. So planning of the hoist capacity for a building construction has become very important. Each building has its own critical traffic period which is the busiest time of the hoist [3]. For each kind of building, there may have three critical traffic periods: up-peak (morning rush hours), down peak (evening rush hours) and down-up peak (lunchtime rush hours) [4]. If a hoist is sufficient to serve the critical traffic periods of the building, the rest of the time will not be a problem [3]. Two commonly used parameters to select hoists for a building are total time needed to send all the workers to their destinations and average waiting time for workers [5-6].

During the period of design and construction of a tall building, some key parameters for the hoist operation have to be considered such as average waiting time for workers, or total time needed to send all workers to their destinations [6-8]. Some authors use mathematical methods to determine these parameters for hoist planning [9-10]. These methods while useful, have some practical problems; it is not easy for those with limited mathematical knowledge to understand and use these methods to analyze the time spent and waiting time criteria when designing hoists arrangements for building design or construction site.

In this paper, a simulation model is proposed to analyze the operation of hoists in down-peak periods for a moderately tall building (for example 15 floors) with one hoist which have a maximum capacity of 14 workers. The model is a hybrid of agent based methodology with discrete event methodology to easily get parameters discussed above for hoist operation.

In the paper, we first describe the down-peak, next build a model for this situation; then the validation of the model is tested, and lastly the base model is expanded to random situation with any arrival distribution.

2 Description of the down-peak mode of construction hoists

In the lunch break or after work time of workers on floors above the ground want to go down to the ground floor (floor1). Workers from a given floor will first leave their work site and arrive at the waiting area of the floor, then call the hoist, if the hoist is just there taking the hoist and leaving or waiting for the hoist. If we assume the hoist first stays idle at the ground floor (floor 1). If there is user calling for it from a given floor above, the hoist will go upward to pick the user. During the going upward process, if someone whose floor number is bigger than the floor number of the user who just called the hoist, the hoist will go to the user with the highest floor number. During the downward process, when the hoist arrives at each floor, if it has spare capacity and there are workers waiting on this floor, it will load workers on this floor, if the hoist is full loaded

or there is no user waiting on this floor, it will go down directly to the next floor. When the hoist arrives at the ground floor, it will unload all the workers in it and wait for another call. During the down-peak, it is assumed that there is no user who wants to go upward. And also there is no user who wants to go upward or downward between the floors. The situation can be schematically illustrated as Fig.1.

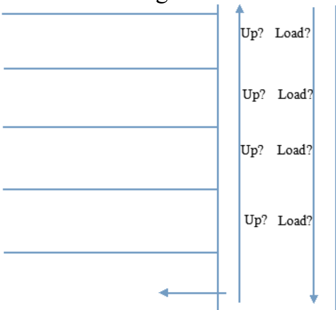

Fig. 1. Hoist's operation in down-peak

3 Base model of the down-peak situation

3.1 The behaviour of user

Each user leaves his work site in the building and arrives at the waiting area of his floor waiting for the hoist. The workers' behavior can be treated as a process. A process flow chart is used to express the behavior of workers on each floor (see fig.2).

The behavior of workers on each floor is expressed with two blocks. The first block (for example *f2*) is a source block to express workers' behavior of leaving his work area on a given floor (floor2) and arriving at the waiting area. So when a user arrives at the waiting area of the floor, the source block will generate an entity. The second block (for example q2) is a queue block to express the workers' behavior of waiting on a given floor (floor2). The function of the block is to store the entity generated by the corresponding source block. A *person* agent is predefined to represent the user entity.

Fig. 2. User's behavior in down-peak

3.2 The behaviour of hoist system

The hoist's behavior can be represented as a series of events that can occur in one or more possible states. A state chart of agent-based modeling methodology is used to define the behavior of the hoist system (see fig.3).

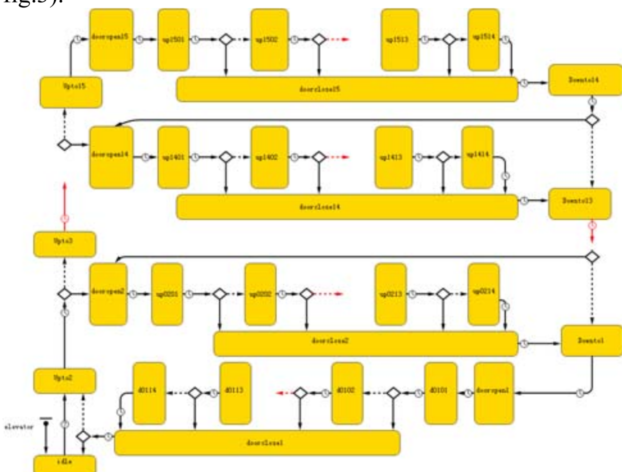

Fig.3. Behavior of hoist system in down-peak

The hoist first stays idle on the ground floor (state *idle*). If there are workers calling for it, it will go upward (transiting to state *Upto2*) to pick them up, or the hoist will stay idle, so the transition from the state *idle* to state *Upto2* is a condition transition. It will take the hoist some time to arrive at floor2, so the state *Upto2* is followed a time transition, and all states in fig.3 are followed by time transition except state *idle* because time are needed to transit from these states to other states. When the hoist arrives at floor2, it will either keep going upward or opening door and loading workers on this floor. So the time transition coming from state *Upto2* is followed by two branches, one goes to state *Upto3*, the other goes to state *dooropen2*.

If the hoist opens the door and loads on this floor, its state will transit to *dooropen2*, it will then load the first user on the floor (transiting to state *up0201*), after loading the first user in the waiting queue of the floor, it will then either load the second user (transiting to state *up0202*) or close the door (transiting to state *doorclose2*), because after the loading the hoist may not have spare capacity to hold user or there may not have workers waiting on the floor. The hoist can load 14 workers in maximum because the maximum capacity of the hoist is set to 14 workers. So after loading the 14th user, the hoist will close the door. If the hoist keeps going up (transiting to state *Upto3*) but not to load on the floor, when it arrives at floor3, it will have similar behavior as behavior on floor2. These similar behaviors on floor3 to floor13 are omitted in fig.3.

When the hoist loads workers on a given floor, after closing the door, it will go down to the next floor (transiting to state *Downto#*). When it arrives at the next floor, if the hoist has some spare capacity and there are workers waiting on the floor, the hoist will open the door and load workers, if the hoist has no spare capacity or there is no user waiting on the floor, the hoist will keep going down to another floor. That means when the hoist goes down to a given floor above the ground, it either loads workers on the floor or keeps going down to another next floor. So, each of the time transition of state *Downto#* is followed by two branches. For example, if the hoist closes the door on floor15 (state *doorclose15*), it will go down to floor 14 (transiting to state *Downto14*). When the hoist arrives at floor14, it will make a decision about opening the door (transiting to state *dooropen14*) and load workers or going down to floor13 (transiting to state *Downto13*).

When the hoist goes down to ground floor (floor1), it will open the door (state *dooropen1*), and unload the first user (state *d0101*) in the hoist, after unloading the first user, the hoist will check if there are workers in the hoist, if there are, the hoist will keep unloading the second user until the last user is unloaded, if there is not, the hoist will close the door. There are 14 possible workers to be unloaded (states *d0101* to *d0114*) because the hoist's maximum capacity is 14 workers. The time transition of each unloading state (states *d0101* to *d0113*) is followed by two branches except the state *d0114*. One is pointing to the next unloading state (one of the states *d0102* to *d0114*) and the other is pointing to state *doorclose1*.

After unloading workers and closing the door, if there are workers waiting on the floors, it will choose to go up again, if there is not, it will stay idle on the ground floor. So the time transition of state doorclose1 is followed by two branches, one is pointing to state *Upto2*, the other is pointing to state *idle*.

3.3 Parameters, collection, variables, Java codes or expressions for the base model

The behaviors of workers and hoist system are expressed with different methodologies. To construct the model , the *Anylogic* simulation tool (education version) is used to incorporate the discrete-event process flows chart with the agent based state chart. It is said that *AnyLogic* is the only simulation tool that supports all the most common situation methodologies in place today: system dynamics, process-centric, and agent based modeling [11-13]. The discrete-event process flow chart is put in the *main* agent of AnyLogic with a *hoist* agent collection named hoists. The state chart of the hoist system is embedded in the *hoist* agent of collection hoists.

Some parameters, collection, variables, events, Java codes and expressions are used to construct the model. The hoist first stays idle on the ground floor (floor1), when there is any user calling for it, it will leave state *idle*. So the trigger condition for the transition from the state *idle* is that at least one of the queues on the floors is not empty (see fig.4).

Condition:

```
get_Main().q2.size()>0||
get_Main().q3.size()>0||
get_Main().q4.size()>0||
get_Main().q5.size()>0||
get_Main().q6.size()>0||
get_Main().q7.size()>0||
get_Main().q8.size()>0||
get_Main().q9.size()>0||
get_Main().q10.size()>0||
get_Main().q11.size()>0||
get_Main().q12.size()>0||
get_Main().q13.size()>0||
get_Main().q14.size()>0||
get_Main().q15.size()>0
```

Fig.4. Trigger condition for the hoist leaving state idle

A predefined parameter in the *hoist* agent named *Onefloortime* is used to represent the time needed for the hoist to go through one floor (see fig.5). The timeout values of the transitions from states *Upto#* and *Downto#* are defined by *Onefloortime* respectively.

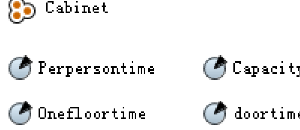

Fig.5. Parameters and collection in the hoist agent

When the hoist arrives at floor2, the condition for the hoist to open the door and loads workers on this floor is that there are workers waiting for the hoist on floor2 but if there is not any user waiting for the hoist above this floor at this moment. That means the queue on floor2 is not empty, but all the queues above floor2 are empty (see fig.6), the condition for the hoist to to go up to floor3 (state *Upto3*) can be set to default.. If the hoist goes up to a given floor, the condition for the hoist choosing to open the door can be similarly defined. If the queue of the floor is not empty, but the queues above the floor are empty, the condition for the other branch is set to default and all the condition for the branches with dashed lines are set to default.

Condition:

```
get_Main().q2.size()>0&&
get_Main().q3.size()==0&&
get_Main().q4.size()==0&&
get_Main().q5.size()==0&&
get_Main().q6.size()==0&&
get_Main().q7.size()==0&&
get_Main().q8.size()==0&&
get_Main().q9.size()==0&&
get_Main().q10.size()==0&&
get_Main().q11.size()==0&&
get_Main().q12.size()==0&&
get_Main().q13.size()==0&&
get_Main().q14.size()==0&&
get_Main().q15.size()==0
```

Fig.6. Condition for the hoist to transit from state Upto2 to dooropen2.

A parameter named *doortime* is predefined in the hoist agent to represent the time needed to open or close the door (see fig.5). All the timeout values of states *dooropen#* and *doorclose#* are defined by the *doortime* parameter. Another parameter named *Perpersontime* is also predefined in the hoist agent, this parameter is used to represent time needed to load or unload a user. The timeout value of each state to express the time need to load or unload a user is defined by this parameter.

When the hoist begins to load a user, the first user in the queue of the floor will go into the hoist. The user's positive action is treated as passive action executed by the hoist. The action of a user going into the hoist is treated as the hoist removing a user from the waiting queue of the floor and adding it into the hoist. So when the hoist enters into a loading state, it will execute these actions. For example, if the hoist is loading user on floor2, the hoist will execute the actions in fig.7. *Cabinet* is an agent collection predefined in the *hoist* agent to represent the car of the hoist (see fig.5). The Java codes in fig.7 show that the hoist removes the first user (person agent) from queue of *q2*, define this user as *man*, and add this *man* into the hoist (*Cabinet*).

Entry action:

```
Person man=get_Main().q2.removeFirst();
Cabinet.add(man);
```

Fig.7. Action executed by loading state on floor2

After loading each of the workers on a given floor, the condition for the hoist to choose to close the door is that the queue on the floor is empty or the hoist is full loaded now. For example, if the hoist is now on floor2, after loading a user, the condition for the hoist to close the door is `get_Main().q2.size()==0||(Capacity-Cabinet.size()==0)`. Where *Capacity* is a parameter predefined in the *hoist* agent to represent the maximum capacity of the hoist, its value is 14 in our model. (See fig.5).

If the hoist has loaded workers and closed the door, it will go down to the next door (transiting to state *Downto#*), when it arrives at the next floor (floor#). The condition for the hoist to open the door to load user on the floor (state *dooropen#*) is that the hoist still has some spare capacity and there are workers waiting on floor# (`get_Main().q#.size()==0&&(Capacity-Cabinet.size()==0)`). But when the hoist arrives at ground floor (floor1), the hoist opens the door and loads workers.

When the hoist begins to unload a user it will remove a user from its car, the action of unloading state on the ground floor is to remove a *person* agent from the *Cabinet* collection (`Cabinet.removeFirst();`). When a user is unloaded, the total number of workers moved down is recorded by a predefined variable named *totalpersonmoved* in the *main* agent. So when the hoist leaves an unloading state (states *d0101* to *d0114*), it will execute an action to make the parameter *totalpersonmoved* increased by one (`get_Main().totalpersonmoved++`). After unloading a user on the ground floor, the condition for the hoist to close the door is that there is not any user in the hoist (`Cabinet.size()==0`).

If all the workers on the floors are moved down to the ground floor, the hoist will go into state idle, the moment is set as the time spent to move all the workers. So when the hoist enters into state *idle* and the total workers moved are equal to the number of workers needed to be moved, the hoist will record the time. The entry action of state idle will be like action in fig.8 if the total number of workers needed to be moved is 336. *Timespent* is a predefined variable in the main agent to represent the time needed to remove all workers down to ground floor.

Entry action:

```
if(get_Main().totalpersonmoved==336)
get_Main().Timespent=time()
```

Fig.8. Entry action of state idle

To get the waiting time and average waiting time of workers on each floor, eight variables are used for each queue of a given floor. For example, eight variables in fig.9 are used for the queue on floor2 to record the waiting time and average waiting time of workers on the floor. Number 2 is used to represent the floor number; variables for the queues on other floors can be expressed in the similar way. The variables *Intime2*, *Outtime2* and *waitingtimearray2* are array variables, the others are plain variables. When a user enters into the queue, an index is tagged to him to represent the entering order of the user and when a user goes out of the queue, an out index is tagged him to the out order of the user. For example, the first user entering the queue is tagged by number 1 and the second by number 2. The first user out of the queue is tagged by number 1 and the second by number 2, and so on. When a user enters into the queue, the entering time is recorded by *Intime2* variable (see the on enter action in fig.9a). And when a user goes out of the queue, the *outtime* variable is used to record the out time of the user (see the on remove action in fig.9a). The waitingtimearray2 is used to record the waiting time of the user going out of the queue. The other variables are used to get the average waiting time of workers having gone out of the queue (see the on remove action in fig.9b).

Inindex2 Intime2
Outindex2 Outtime2
Waitingtimearray2 waitingtime2
Totalwaitingtime2 Averagewaitingtime2

Fig.9a. Variables used for the queue on floor2

On enter:

```
Intime2[Inindex2]=time();
Inindex2++;
```

On remove:

```
Outtime2[Outindex2]=time();
Waitingtimearray2[Outindex2]=Outtime2[Outindex2]-Intime2
waitingtime2=Waitingtimearray2[Outindex2];
Totalwaitingtime2+=waitingtime2;
Outindex2++;
Averagewaitingtime2=Totalwaitingtime2/(Outindex2);
```

Fig.9b. Actions for the queue on floor2.

4 Validation test for the base model

To test the model's validation, the values of parameters for the hoist are assumed to be certain, the values of door time, per person time and one floor time are 1.5, 1, and 1.5 seconds respectively. The capacity of the hoist is 14 workers. The total number of workers that need to be moved is 168, and all the workers are waiting in the queue on a floor when the simulation starts.

4.1 Testing the time spent parameter

All the workers are first put on floor15 then floor14 and finally on the floor2. To put all the workers on a given floor at the beginning of the simulation, a system function `inject()` is used in the *on startup* field of *main* agent. For example, if all the 168 workers are on floor15, `f15.inject(168)` is used in the *on startup* field of agent Main to put all the 168 workers in the queue on floor15. All the outcomes for each situation are show in fig.10. If all the workers are on floor15, the total time spent is 912 seconds, if all the workers are floor9; the total time spent is 696 seconds and if all the workers are on floor2, the total time spent is 444 seconds.

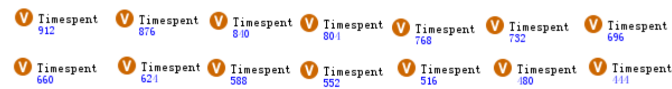

Fig.10. Time spent outcome for the time spent parameter testing.

To check if these outcomes are correct, the values of time spent parameter for each of these situations are calculated (see table1). The total workers are 168; it will take the hoist to go 12 runs (168/14) for each of the situation. The time spent for one run will be decided by the door time, one floor time, per person time and workers' floor number (*doortime**4+*perpersontime**14*2+*onefloortime**(floor number -1)*2). If all the workers are on floor15, the time spent for run is 76 seconds (1.5*4+1*14*2+1.5*14*2). The total time needed for moving down all the workers on floor15 is 912 (12*76). Total time needed for moving all the workers down to floor1 in other situation are calculated in table1, the

outcomes of simulations (see fig.10) are the same with the calculations values (see table1).

Table 1. the values of time spent and average waiting time for 1-3lifts situation.

Workers' floor number	Time needed for one run	Total time needed to move all the workers down to floor1
15	76	912
14	73	876
13	70	840
12	67	804
11	64	768
10	61	732
9	58	696
8	55	660
7	52	624
6	49	588
5	46	552
4	43	516
3	40	480
2	37	444

4.2 Testing the average waiting time parameter

If all the workers are on floor15, the simulation outcome of the workers' average waiting time is 447 seconds (see fig.11). The calculation outcome of this situation is 447 seconds too (see table2).

Fig. 11. Average waiting time outcome of simulation for workers all on floor15

In this situation, for the first run, the waiting time of the first user in the queue is the time spent by the hoist to move itself from floor1 to floor15 and open the door. It is equal to 22.5 seconds (1.5*14+1.5), the second workers waiting time will be increased by one second because he will wait for the first user to go into the hoist. The waiting times of the first workers in the second run, third run will be 98.5 (76+22.5), 174.5(76*2+22.5). And the waiting time of the first user in the last run will be 858.5 (76*11+22.5). The average waiting time for all the workers is 447 seconds.

Table 2. The calculation values of average waiting time for all workers on floor15.

Number of run	1	2	3	4	5	6	7	8	9	10	11	12
Workers moved	14	28	42	56	70	84	98	112	126	140	154	168
1st user's waiting	22.5	98.5	174.5	250.5	326.5	402.5	478.5	554.5	630.5	706.5	782.5	858.5
2nd user's waiting	23.5	99.5	175.5	251.5	327.5	403.5	479.5	555.5	631.5	707.5	783.5	859.5
3rd user's waiting	24.5	100.5	176.5	252.5	328.5	404.5	480.5	556.5	632.5	708.5	784.5	860.5
4th user's waiting	25.5	101.5	177.5	253.5	329.5	405.5	481.5	557.5	633.5	709.5	785.5	861.5
5th user's waiting	26.5	102.5	178.5	254.5	330.5	406.5	482.5	558.5	634.5	710.5	786.5	862.5
6th user's waiting	27.5	103.5	179.5	255.5	331.5	407.5	483.5	559.5	635.5	711.5	787.5	863.5
7th user's waiting	28.5	104.5	180.5	256.5	332.5	408.5	484.5	560.5	636.5	712.5	788.5	864.5
8th user's waiting	29.5	105.5	181.5	257.5	333.5	409.5	485.5	561.5	637.5	713.5	789.5	865.5
9th user's waiting	30.5	106.5	182.5	258.5	334.5	410.5	486.5	562.5	638.5	714.5	790.5	866.5
10th user's waiting	31.5	107.5	183.5	259.5	335.5	411.5	487.5	563.5	639.5	715.5	791.5	867.5
11th user's waiting	32.5	108.5	184.5	260.5	336.5	412.5	488.5	564.5	640.5	716.5	792.5	868.5
12th user's waiting	33.5	109.5	185.5	261.5	337.5	413.5	489.5	565.5	641.5	717.5	793.5	869.5
13th user's waiting	34.5	110.5	186.5	262.5	338.5	414.5	490.5	566.5	642.5	718.5	794.5	870.5
14th user's waiting	35.5	111.5	187.5	263.5	339.5	415.5	491.5	567.5	643.5	719.5	795.5	871.5
Total waiting	406.0	1470.0	2534.0	3598.0	4662.0	5726.0	6790.0	7854.0	8918.0	9982.0	11046.0	12110.0
Accumulated total waiting time	406	1876	4410	8008	12670	18396	25186	33040	41958	51940	62986	75096
Average waiting time	29.00	67.0	105.0	143.0	181.0	219.0	257.0	295.0	333.0	371.0	409.0	447.0

5 Expand the base model to random situations

In reality, the parameters of the hoist and arrival distribution of workers may be random. To expand the base model to the random situation, the arrival distributions of workers should be expressed as random. The other is that enough runs of simulation should be done to get robust simulation outcomes.

Some assumptions are made to explain the expanding. The one floor time is assumed to be between 1.5 and 2 seconds uniformly, the door time is between 1and 1.5 seconds uniformly, the per person time is between 0.5 and 1 uniformly. There are 24 workers on each floor. Their arrival distribution is one user every 20 to 30 seconds.

5.1 Random variables in the model

In the random situation, some of the parameters or variables are random. The system function of the *AnyLogic* software can be used to define these random parameters and variables. For the assumption numerical example, function `uniform()` is used to express the parameters of the hoist. The default values of parameters *doortime*, *onefloortime* and *perpersontime* are set to `uniform(1,1.5)`, `uniform(1.5,2)` and `uniform(0.5,1)` respectively.

In the example, the arrival distributions of workers are random too; events can be used to define the arrival distribution. There are three kinds of events: Timeout, Rate, and Condition. A Timeout event with cyclic mode is used to define the arrival distribution of workers on each floor in our example. The Event in fig.12 is used to define the arrival distribution of workers on floor2. In 20 to 30 seconds, a user will come, and the number of user on this floor should not more than 24. `count()` is a system function used to count the number of user that have been injected into the source block *f2*.

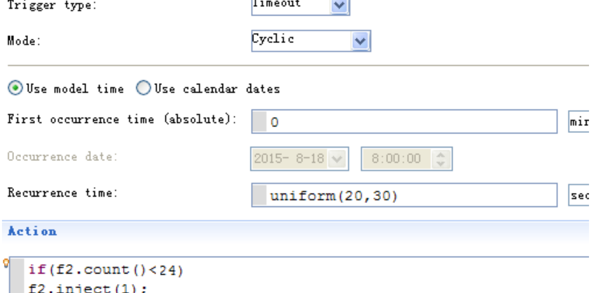

Fig.12. Arrival distribution defining with an event

5.2 Run the simulation for enough time

To get robust simulation outcomes, the model should be able to run enough times. The simulation is repeated. A variable named N is first defined in the agent of *main* to represent the number of runs that has been finished setting the initial value of N to 1. Then the finishing condition of each run is defined in the Entry action field of state *idle* of the hoist. For the example, the finishing condition is defined as when the simulation is on the Nth run, its finishing condition is that the workers that have been moved (*totalpersonmoved*) is N times of total workers needed to be moved (variable *Totalpersonneedtomove*) , variable is a predefined variable in the *main* agent, one run is finished, the value of N will be increased by one (see fig.13).

Entry action:
```
if(get_Main().totalpersonmoved==get_Main().N*get_Main().Totalpers
get_Main().N++;
}
```

Fig.13. Finishing condition for each run.

When the simulation runs are repeated, the arrival distribution is also repeated. The change in code is shown in fig.14.

Action
```
if(f2.count()<N*24)
f2.inject(1);
```

Fig.14. Repeating workers' arrival distribution for workers on floor2

When the simulation has finished the specified runs, average time spent of all runs and average waiting time for all the workers are calculated. An event named *Pausesimulation* is used to pause the simulation. Two variables, *Averagetimespent* and Averagewaitingtime are used to calculate the two time parameters; these two variables are predefined in the *main* agent. The condition of the event is defined by the variable N and the specified runs. If the specified runs are 100, then the condition and the action of the event can be expressed as condition and action in fig.15.

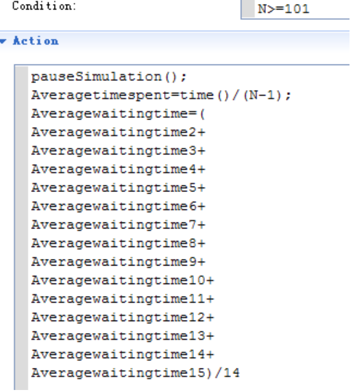

Fig.15. Condition and action of the event Pause simulation

For the assumed example, after 100 runs of simulation, values of variable *Averagetimespent* and *Averagewaitingtime* can be determined. The average time spent is 1303 seconds and average waiting time is 473 seconds (see fig.16).

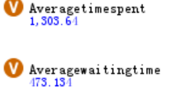

Fig.16.Outcome of the simulation after specified runs

6 CONCLUSION

In this paper we propose a down-peak model with single hoist. The model hybrids discrete-event methodology

with agent based methodology. The model can be used in certain condition and random situations. Using this model we can analyze the time spent parameter and average waiting time parameters for hoist planning. If there are two more hoists but each hoist is with its independent waiting queue, it can also be treated as independent single hoist situation.

In this paper, a down-peak model with single hoist is proposed. The model is a hybrid of discrete-event methodology with agent based methodology to simulate the operation of hoist in the down peak. The discrete-event methodology is used to simulate the behavior of the workers on each floor, and the agent-based methodology is used to simulate the behavior of the hoist system. The model's validation is tested under deterministic conditions and then expanded to handle random situations. The two commonly used operation parameters (time spent and average waiting time) can be determined for hoist planning through model simulation. With two or more hoists, during the rush hours, if each hoist has a separate waiting line, each hoist can be treated as an independent hoist. If all the hoist s only have one waiting line, hoists' behavior will be affected by each other, in that case, the model will need to be modified. That will form the next stage of this work.

REFERENCES

[1] Bao Ding, Yong-Ming Zhang, Xi-Yuan Peng (2013). A hybrid approach for the analysis and prediction of hoist user flow in an office building. Automation in construction, 35 69-78 DOI 10.1016/j.autcon.2013.03.003

[2]M.L. Siikonen (1997). Customer service in an elevator system during up-peak, Transportation Research 31(2) 127-139

[3] G.F. Newell(1998). Strategies for serving peak hoist traffic, Transportation Research 32(8) 583-588

[4] T. Nagatani(2003). Complex behaviour of hoists in peak traffic, Physica A 326 556-566 DOI:10.1016/S0378-4371(03)00278-4

[5] T. Nagatani(2004). Dynamical transitions in peak hoist traffic, Physica A 333 441-452 DOI 10.1016/j.physa.2003.10.001

[6] Yutae Lee, Tai Suk Kim, Ho-Shin Cho, Dan Keun Sung, Bong Dae Choi (2009). Performance analysis of an elevator system during up-peak, Mathematical and Computer modelling, 49 423-431DOI 10.1016/j.mcm.2008.09.006

[7] Yoonseok Shin, Hunhee Cho, Kyung-In Kang (2011). Simulation model incorporating genetic algorithms for optimal temporary hoist planning in high-rise building construction, Automation in construction, 20 550-558 DOI 10.1016/j.autcon.2010.11.021

[8] Stefan Heinz, JÖrg Rambau, Andreas Tuchscherer (2014). Computational bounds for hoist control policies by large scale linear programming, Math Meth Oper Res 79 87-117 DOI 10.1007/s00186-013-0454-5

[9] U Beißert, M König, H-J Bargstädt (2010). Soft constraint-based simulation of execution strategies in building engineering, Journal of simulation, 4 222-231 DOI 10.1057/jos.2010.8

[10] Strakosch, G. R. (2010). *The vertical transportation handbook* (Vol. 4). Hoboken, NJ, USA: John Wiley & Sons.

[11] Arashpour, M. and M. Arashpour (2015). "Analysis of Workflow Variability and Its Impacts on Productivity and Performance in Construction of Multistory Buildings." Journal of Management in Engineering 31(6): 04015006.

[12] Arashpour, M., R. Wakefield, N. Blismas and T. Maqsood (2015). "Autonomous production tracking for augmenting output in off-site construction." Automation in Construction 53: 13-21.

[13] Andrei Borshchev (2013), The big book of simulation modeling: multi-method modeling with AnyLogic 6, Lisle,IL:Anylogic North America.

[14] Arashpour, M., R. Wakefield, N. Blismas and B. Abbasi (2016). "Quantitative analysis of rate-driven and due date-driven construction: Production efficiency, supervision, and controllability in residential projects." Journal of Construction Engineering and Management 142(1): 04015006.