INVITED PAPER *Special Section on Concurrent/Real-time and Hybrid Systems: Theory and Applications*

MceSim: A Multi-Car Elevator Simulator

Toshiyuki MIYAMOTO[†a)] ***and*** **Shingo YAMAGUCHI**[††b)], ***Members***

SUMMARY Multi-Car Elevator (MCE) systems, which consist of several independent cars built in the same shaft, are being considered as the elevators of the next generation. In this paper, we present *MceSim*, a simulator of MCE systems. MceSim is an open source software available to the public, and it can be used as a common testbed to evaluate different control methods related to MCE systems. MceSim was used in the group controller performance competition: CST Solution Competition 2007. This experience has proven MceSim to be a fully functional testbed for MCE systems.

key words: *CST Solution Competition, multi-car elevator, group control, evaluation*

1. Introduction

With the appearance of high-rise buildings, the efficient transportation of people or objects between floors has become a problem. Since elevators have been used as transportation means in high-rise buildings from a long time ago, it is necessary to improve their transportation capacity. To improve the transportation capacity of elevators, it has been thought about the optimization of the elevator driving control and the improvement of the elevator functions such as their traveling speed. However, there is no solution more essential than the increase in the number of elevator cars. However, generally the number of elevator cars contained by a shaft is just one, and therefore the increase in the number of elevator cars implies also an increase in the number of shafts. In addition, an increase in the number of shafts suggests an increase in the area occupied by elevators within the building; thus the number of shafts that is convenient to add becomes limited. Double-deck elevators, in which two cars are attached one on top of another, are being used in some places; however a method to improve its continuous transportation capacity is still being investigated. Consequently, the Multi-Car Elevator (MCE), which consists of several independent cars built in the same shaft, is being considered as the elevator of the next generation.

As implementation methods for the MCE systems, the rope-driven method used in conventional elevators [1], [2] and the rope-less method, which implements the installation of linear motors [3], [4], have been proposed. ThyssenKrupp, a German elevator manufacturer company, is currently using the rope-driven method in its MCE TWIN system, which consists of two elevator cars built in the same shaft [5][*]. In addition, different kinds of research related to the MCE driving control are being performed. For example, Yamashita et al. have presented a model to measure the MCE transportation capacity and the operation interval in the main floors. By using this model, they propose an MCE design method [6]. Shiraishi et al. have proposed an autonomous distributed control method using learning automata [7]. Suzuki et al. have proposed an optimization method for MCE driving control rules using a GA [8], [9]. Markon et al. have proposed a control algorithm using a continuously running real-time GA method [10]. Ikeda et al. have proposed application of evolutionary multi-objective optimization and an examplar-based policy representation to design traffic-sensitive MCE controller [11], [12]. Suzuki has performed a study on intra-shaft operating methods for MCE systems [13].

In this paper we present MceSim, a simulator of MCE systems. MceSim is an open source software available to the public, and it can be used as a common testbed to evaluate different control methods related to MCE systems. A common testbed available to the public is associated to the promotion of investigation, and we consider it as a contribution to the development of the MCE control technology. The structure of this paper is described below. Section 2 provides an outline of MCE systems and Sect. 3 explains about the simulator MceSim. Finally, in Sect. 4 we perform computational experiments using MceSim and evaluate its validness as a testbed.

2. Multi-Car Elevators

The MCE being considered in this paper consists of the following elements.

- **Cars** that transport passengers.
- **Shafts** that provide the space for cars to travel up and down. When moving within a shaft, cars cannot surpass other cars traveling in the same shaft.
- **Terminal Floor**, the first floor of the building where

Manuscript received May 30, 2008.
Manuscript revised July 22, 2008.

[†]The author is with the Department of Electrical, Electronic and Information Engineering, Osaka University, Suita-shi, 565-0871 Japan.
[††]The author is with the Graduate School of Science and Engineering, Yamaguchi University, Ube-shi, 755–8611 Japan.
a) E-mail: miyamoto@eei.eng.osaka-u.ac.jp
b) E-mail: shingo@yamaguchi-u.ac.jp

DOI: 10.1093/ietfec/e91–a.11.3207

[*]A program TWIN Game to experience a multi-car elevator system is currently available at ThyssenKrupp's HP.

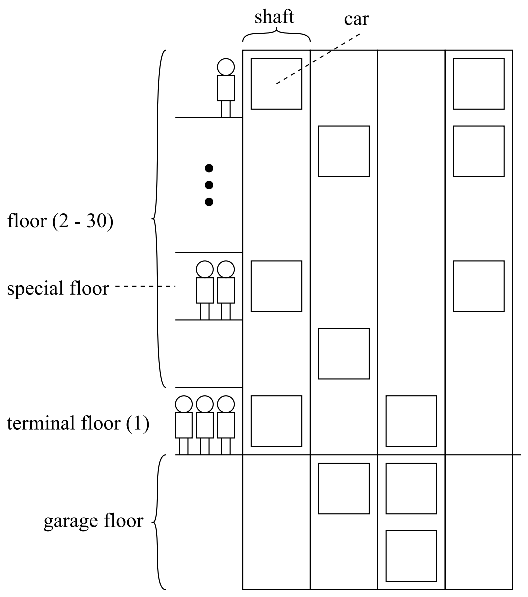


Fig. 1 MCE example: $4S3C30F$.

its entrance is located and mass traffic requests from/to this floor appear.

- **General Floors**, floors located above the terminal floor and where passengers board and descend from the cars.
- **Special Floor**, a general floor for which the traveling demand is specially high.
- **Garage Floors**, floors located below the terminal floor. Garage floors are built so lower cars can take shelter and all cars can be able to provide service to the terminal floor. Passengers are not allowed to board or descend at garage floors.

We use the notation $xSyCzF$ to represent an MCE system, where x indicates the number of shafts, y the number of cars per shaft, and z the number of service floors in the building. The number of garage floors is given by $y-1$. As an example Fig. 1 shows a $4S3C30F$ configuration.

At the moment of calling a car, instead of the conventional up and down buttons, the passengers also indicate their destination floor. Then, the passengers are indicated which car they must board by the MCE system. Hall calls are not generated just randomly, but instead, depending on the time of the day several generation patterns exist. In this paper, the following 4 types of hall call generation patterns are considered.

- **Up-peak**: The demand concentrates on travels from the terminal floor to the general floors.
- **Down-peak**: The demand concentrates on travels from the general floors to the terminal floor.
- **Special Floor**: The demand concentrates on travels from the special floor to the general floors and vice versa.
- **Ordinary**: The demand is random.

In the control of MCE, not only the optimization of an objective function, but also the following conditions must be considered.

Safeness Cars in the same shaft must not overlap each other.
Liveness Cars must not fall into a deadlock state.
No travel deviation Cars will transport passengers boarded at a departure floor up to their destination floor without changing direction in the process.
No transshipment Cars will transport passengers from their departure floor to their destination floor without requiring them to change cars at any floor along the way.

In actual MCE systems due to the action of safety devices it is not possible for the collision among cars to occur. However, in the simulator the state in which the position of cars overlap may occur. In this paper we refer to this state as the overlapping state.

In MceSim, the controller of $xSyCzF$ is composed of one group controller, ($x\times y$) car controllers, and x shaft controllers. See Fig. 2. The group controller allocates the hall calls to the car. In addition, the group controller prevents the overlapping of cars, and in order to increase the efficiency of the whole system, it can allocate *pseudo calls* and extend the stop time of cars. A pseudo call has no relation with the hall calls, and it is just generated to make cars move. An allocated pseudo call is processed with higher priority than hall calls.

The car controllers independently perform the scheduling based on the hall call allocation, and each car controller performs the driving control of its corresponding car. However, the car controllers do not perform the overlapping prevention, which is left to the group controller. If the group controller does not use the pseudo calls and the stop time extension properly, an occurrence of the overlapping state cannot be prevented. Each shaft controller evaluates the possibility of overlapping to occur in its corresponding shaft, and when risk of overlapping is detected, it overwrites the control by the car controllers.

In MceSim, MCE systems are modeled by discrete event dynamical systems, and the discrete events (hall call generation, sending of commands to the cars) occur in discrete time $0, 1, 2, \cdots$ seconds. MceSim uses the values in Table 1 as physical parameters of MCE systems.

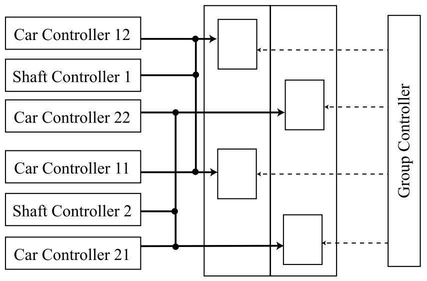


Fig. 2 MCE controllers: The solid line expresses the control event by car controllers and shaft controllers. The broken line expresses the call allocation, the pseudo calls allocation, and the stop time extension by the group controller.

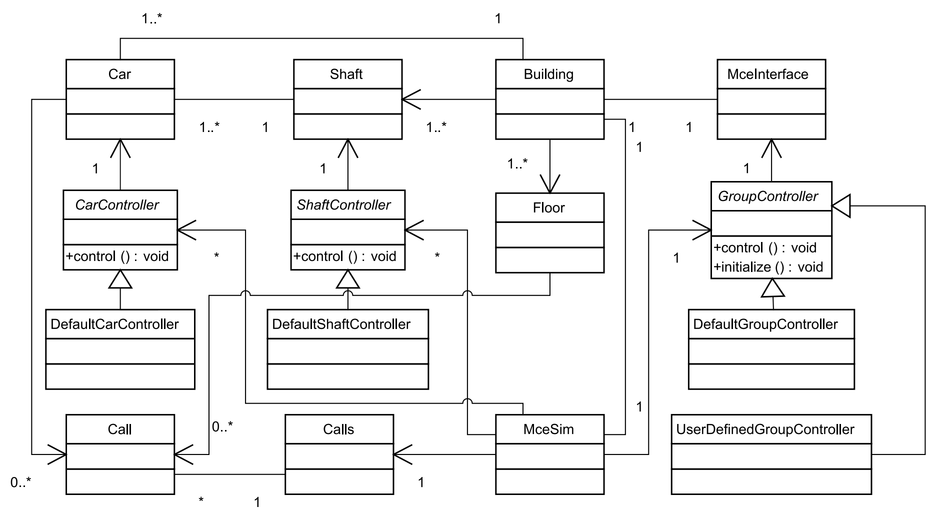


Fig. 3 MCE simulator model.

Table 1 Physical parameters of MCE systems used in MceSim.

Distance between Floors (same as car height)	4 m
Stop time due to hall calls[†]	6 s
Stop time due to pseudo calls	0 s
Max velocity	6 m/s
Acceleration	2 m/s^2
Car Capacity	20 person

[†]The stop time due to hall calls is the total time required for the opening of doors, boarding and descending of passengers, and closing of doors.

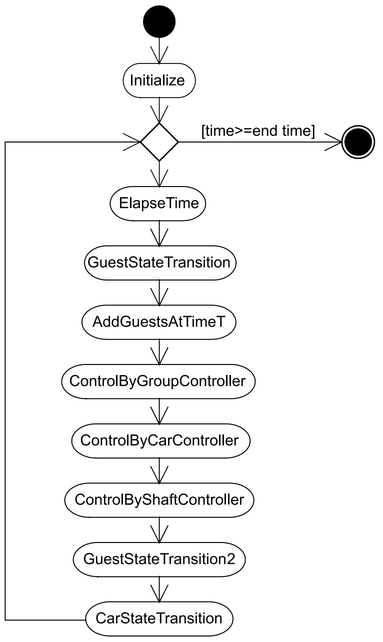


Fig. 4 Flow of MceSim.

3. MceSim

3.1 Static Structure of MceSim

The class diagram in Fig. 3 shows the core classes of MceSim and the relationship between them. A building is composed by four classes: Building, Shaft, Car, and Floor, and one instance of Shaft, Car, and Floor is generated for each shaft, car, and floor, respectively. An instance of Call is generated for each call, and all instances of Call are stored in an instance of class Calls. A call may be held by a floor or a car but not by both at the same time. When a call is held by a floor, this means passengers of this call are waiting at the floor. On the other hand, when a call is held by a car, this means that they are in the car and traveling to their destination floor.

CarContoller, *ShaftController*, and *GroupController* are abstract classes for the car controller, the shaft controller, and the group controller, respectively. The car control algorithm and the shaft control algorithm described in Sect. 3.3 and Sect. 3.4 are implemented in DefaultCarController and DefaultShaftController, respectively.

A group controller can access objects in MceSim through the interface class: MceInterface[†]. We will show the methods provided by the interface class in Sect. 3.5.1, and a sample group controller implementation, DefaultGroupController, in Sect. 3.5.2.

MceSim can run with other controller classes as long as they inherit the corresponding controller abstract classes. Since MceSim calls a method control() for each time step, an algorithm of a new controller must be implemented in this method. In order to initialize a group controller, a method initialize() is also provided. The main function is found in the class MceSim, and the program flow of the simulator is described in Sect. 3.2.

3.2 Flow of MceSim

The activity diagram in Fig. 4 shows how MceSim runs. Af-

[†]MceSim was developed for a competition event of group controllers, and in order to hide internal objects from the group controller programs developed by players, the interface is inserted here.

ter the initialization step, MceSim repeats the same procedure until the termination condition is satisfied. Each cycle in the activity diagram corresponds to a unit time in the simulator, where the unit time is 1 second.

In step GuestStateTransition, passengers descend from cars, and waiting passengers board cars. Here, once the capacity limit of the car has been reached, remained passengers are not able to board. In this case, MceSim generates a new call to transport them to their destination floors.

In step AddGuestsAtTimeT, MceSim selects, from the set of calls in Calls, those calls whose generated time corresponds to the current time and sets each of them in their corresponding departure floor.

In step ControlByGroupController, MceSim calls the method control() of the group controller. The group controller sends the control commands (call assignment, pseudo call assignment, and stop time extension) to cars.

In step ControlByCarController, the car controllers send control commands to cars based on the call assignment done by the group controller. Each car controller decides its control without considering state and control of above and below cars. Thus, overlapping of cars may occur. To prevent the overlapping state, each shaft controller is able to overwrite the control commands performed by the car controllers in step ControlByShaftController.

In step GuestStateTransition2, newly generated passengers board the cars. Finally, in step CarStateTransition, the state of each car is updated based on the control commands received from the group, car, and shaft controllers.

3.3 Car Behavior and Default Car Controller

The main state variables of a car are shown in Table 2. The physical state of the car is described by its elevation, velocity, and door state. Instead of using variables for each attributes, we use the discrete state variable state to describe it.

The state machine in Fig. 5 shows transition relationship between the discrete states. Considering values of the acceleration ($2\,\mathrm{m/s}^2$), max velocity (6 m/s), and unit time length (1 s) of a car, there are seven possible speed states. However, if we have only seven speed states, a problem on elevation control of the car arises. For example, a car cannot stop at 2 m higher position when the speed state of the car is up in 2 m/s. If the car kept the speed for 1 second and then decelerated, it would move 3 m before stopping. If the car began to decelerate immediately, it could move only 1 m. To overcome this problem, auxiliary states up 1m and down 1m are introduced. The time required to load and unload passengers is 6 seconds, and in order to monitor the time the car must remain at a floor, the load/unload states stay for call [1-6] have been included. The loading and unloading of passengers can be either of the load/unload states.

The state transition occurs in step CarStateTransition. The control events (up, down, open) required by the discrete transition states are provided by the car and shaft controllers. The control performed by the car and shaft controllers is stored in the state variable controlEvent. However, transition from the states stay for call 1 to stay for call 6 is independent of the control events.

The variable elevation is updated in step CarStateTransition through the following procedure. If the speed state is the same before and after the state transition, the elevation is updated according to the velocity rate. Otherwise, an intermediate value between the two velocity rates is used. When the transition is to or from the auxiliary speed states the elevation is updated by 1 m. The variables extraTime and extraPseudTime are reduced by 1 in step CarStateTransition.

The guards inherent to all the transitions to the state stop (e.g. [(elevation − 1)%4 = 0]) assure that at the time of stopping the car will do it at a floor; i.e. the car cannot stop between floors.

Whenever a call is assigned, it is included in the set assignedCalls. Then, when the passengers associated with

Table 2 State variables of a car.

Name	Description
elevation	vertical position
state	discrete state (Fig. 5)
assignedCalls	set of assigned calls
onCalls	set of calls whose passengers are in the car
upSchedule	upward schedule
downSchedule	downward schedule
scheduleDir	direction of current schedule
pseudDest	destination floor of pseudo call
extraPseudTime	duration of stay at pseudDest
extraTime	additional time to stay at the current destination floor
controlEvent	control given by controllers

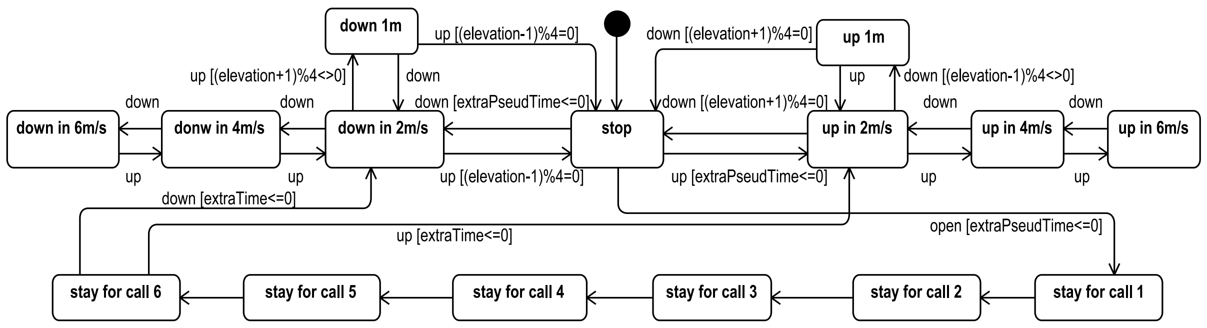


Fig. 5 State transition model for a car.

this call board the car, the call is included in the set onCalls, and when they finally descend the car at their destination floor the call is removed from both sets. The car controller[†] constructs its schedule from these two sets and stores its corresponding floors in upSchedule and downSchedule.

In step ControlByCarController, the car controller decides to which floor it will head to, based on the schedule, whose direction is shown by scheduleDir; and the parameters pseudDest, extraTime, and extraPseudTime. Depending on the relation between the target floor and current position, and speed of the car, the car controller chooses its control event from up, down, open, and keep.

Normally, in the DefaultCarController, the first floor appearing in the schedule is set to the next floor to which the car will be heading to. By this, the **no travel deviation** condition is satisfied. However in the case of allocated pseudo calls and control performed by the shaft controller, which have higher priority than allocated hall calls, violations to the **no travel deviation** condition may occur.

3.4 Shaft Controller

3.4.1 Overlapping Avoidance Condition

The car controller makes the schedule, which is composed by the allocated calls, and controls the car based on this schedule. Due to this, the overlapping between two adjacent cars may occur. In order to prevent overlapping from occurring the shaft controller is incorporated.

We introduce the concept of a virtual floor. The virtual floor of a car is defined as the closest floor in which the car is able to stop. In the case the car is parked at a floor, this floor is considered to be its virtual floor.

Let us consider a given shaft to have $n + 1$ cars $c_0, \cdots, c_n$ built in it, where car c_0 represents the bottom car and c_n the top car. We represent the discrete state and the virtual floor of c_i in time t as $q(t)$ and $f_i(t)$, respectively. With respect to the overlapping of cars, the following proposition holds.

Proposition 1: Let us make all cars stop at time t. In the stop state, a necessary condition for cars not to overlap is the following equation:

$$f_0(t) < f_1(t) < \cdots < f_n(t). \tag{1}$$

When Eq. (1) is satisfied, we can say that an overlapping in the shaft is avoidable at time t. The above condition guarantees that when stopped, there will not be overlapping cars; in other words, that cars will not stop at the same floor. However it does not guarantee that overlapping will not occur during the stopping process.

Consider the control at time t as $e(t)$. We denote the virtual floor of car c_i at time $t + 1$ under the control $e_i(t)$ as $f_i(t + 1)|_{e_i(t)}$.

Proposition 2: Let overlapping be avoidable at time t. Then, Eq. (2) is a necessary condition for overlapping in the shaft to be avoidable at time $t + 1$.

Table 3 Penalty value of the control change performed by the shaft controller. $e_c(t)$ is the control performed by the car controller and $e_s(t)$ the control performed by the shaft controller.

$q(t)$	$e_c(t)$	$e_s(t)$		
		up	keep	down
stay for call [1-5]	-	-	-	-
stay for call 6	up	1	2	3
	keep	1.5	1	1.5
	down	3.1	2.1	1
stop	open	-	-	-
	up	1	2	3
	keep	1.5	1	1.5
	down	3.1	2.1	1
up $*$	up	1	2	3
	keep	-	1	2
	down	-	-	1
down $*$	up	1	-	-
	keep	2.1	1	-
	down	3.1	2.1	1

$$f_0(t + 1)|_{e_0(t)} < f_1(t + 1)|_{e_1(t)} < \cdots < f_n(t + 1)|_{e_n(t)} \tag{2}$$

In addition, because Eq. (2) becomes Eq. (1) at time $t+1$, if Eq. (1) is satisfied at the initial state, we can guarantee that no overlapping will ever occur by making the MCE satisfy Eq. (2).

3.4.2 Shaft Controller's Overlapping Avoidance Control

Let us consider the case in which Eq. (1) is satisfied by the control performed by the car controller but Eq. (2) is not. Because Eq. (1) is being satisfied by making all cars decelerate, we can easily make Eq. (2) be satisfied. However, by doing this, we would soon be making all cars stop. The shaft controller implemented in DefaultShaftController tries to avoid overlapping based on the following principles:

- Upper cars receive priority.
- Cars are moved as much as possible.

The penalty value corresponding to the moment when the control based on the state of the car and the car controller is overridden by the shaft controller is shown in Table 3. The bar (-) in the table represents the case in which the shaft controller does not exert any control action.

From all the possible combinations of control, the shaft controller selects the one that provides the minimum evaluation value and at the same time satisfies Eq. (2). The evaluation value related to the control performed by the shaft controller is represented by the sum of the penalty values associated to the car control actions, which are shown in Table 3.

If the control performed by the car controller is used without change, the evaluation value is 1, which is smaller than one given when the control is overridden. In other words, when the control performed by the car controller satisfies Eq. (2), the shaft controller performs no change in it.

A car moving up can be made to stop at its virtual floor

[†]In MceSim, the scheduling function of the car controller is implemented in the class Car.

Table 4 Implemented methods in MceInterface.

Method	Operation
assign(carID, callID)	Allocate call to car
prolong(carID, dur)	Extend car's stop time
cancelProlong(carID)	Cancel the stop extension command
assignPseudoCall(carID, dest)	Allocate pseudo call to car
clearPseudCall(carID)	Clear the car's allocated pseudo call
prolongPseudCall(carID, dur)	Set the stop time of the pseudo call

$f(t)$ by continuously sending the command down to it. In the same way, a car moving down can be made to stop at its virtual floor $f(t)$ by continuously sending the command up to it. The shaft controller has the option of making all cars stop at their virtual floor. Therefore if Eq. (1) is being satisfied, by all means overlapping can be avoided. However, if the penalty corresponding to the control action of stopping the car at its virtual floor turns out to be the highest, if possible, the shaft controller will control the car without making it stop.

In the control mentioned above, if a shaft consists of 3 or more cars, it is possible for all of them to stop and cease to move again. In this case, it is required for the group controller to make them move appropriately.

3.5 Group Controller

3.5.1 Group Controller Interface

The group controller is able to observe the state of the system and perform its control through the use of MceInterface. Each object in the MCE system (cars, shafts, calls etc.) has an inherent ID, and in the methods offered by MceInterface the objects are referenced by the ID.

The control methods are shown in Table 4. Through these methods the state variables are changed. The calls allocated to a car by using assign are registered in its assignedCalls set. Also, when a call is allocated to a car, its schedule is updated. Through the prolong method the delay time parameter extraTime of a car is set. The destination floor given by the method assignPseudoCall is set to the parameter pseudDest of the car. The parameter pseudDest is not realized as a set, and if assignPseudoCall is called many times, the last invocation will become the valid one. The stop time at the destination floor of a pseudo call is 0 second. In order to extend the stop time at the destination floor of a pseudo call prolongPseudCall can be used. The delay time provided is registered in extraPseudTime.

3.5.2 Sample Group Controller Implementation

The sample implementation of the group controller DefaultGroupController takes the idea of the division by zones. The control is performed based on the following principles:

1. For each shaft, the total number of floors is equally divided into zones, and for each zone a car in its charge is determined.
2. The allocation of a call set in a given zone is determined by the following rules.

   Up-peak time: The call is allocated to the car in charge of the zone containing the destination floor.

   Down-peak time: The call is allocated to the car in charge of the zone containing the departure floor.

   Special Floor time: The call is allocated to the car in charge of the zone containing the floor different from the special floor.

   Ordinary time: The call is allocated to the car in charge of the zone containing the higher floor between the departure and destination floor.
3. The selection of the shaft is done randomly.
4. The ascent/descent of cars in the same shaft is synchronized.

Table 5 Options of MceSim: n is a integer number and d is a kreal number.

Option	Role
-t n	Simulation time (cycle number)
-b n	Beginning time of all data
-e n	Termination time of all data
-g classname	Name of the group controller class
-d datafile	Name of the data file
-r	Generate calls randomly
-s n	Shaft number (only in random mode)
-c n	Number of cars per shaft (only in random mode)
-f n	Number of service floors (only in random mode)
-p d	Call occurrence probability (only in random mode)
-a pattern	Call generation pattern (only in random mode)
-j n	Special floor (only in random mode)
-i n	Simulation time adjustment parameter
-z	Activate suspend state
-m	Open GUI
-h	Display help

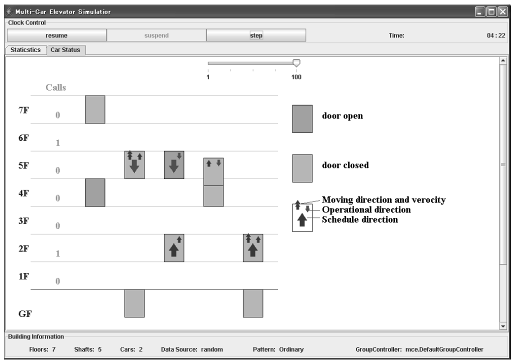
3.6 Operating MceSim

The simulator was developed by using the Java programming language. In order to run it, the J2SE 5.0 or a higher version is required. The operating options are shown in Table 5.

The generation of calls is done by reading a data file or by a random call generation method. When generating calls randomly, options which are indicated as 'only in random mode' in Table 5 can be used to define the MCE configuration. Also, in the random mode a positive value must be provided for the simulation time. In the data file reading mode, if a negative value is provided for the simulation time, the simulation will be performed until all the calls contained in the file are processed.

The default group controller is DefaultGroupController. When a user want to use a different group controller, the option `-g` along with the class name of the desired group controller must be indicated.

To display the GUI of the simulator add the option `-m`.



Time: 04 : 22

Calls: 7F 0, 6F 1, 5F 0, 4F 0, 3F 0, 2F 1, 1F 0

Building Information

Floors: 7 Shafts: 5 Cars: 2 Data Source: random Pattern: Ordinary GroupController: mce.DefaultGroupController

Fig. 6 MceSim screen-shot.

A screen-shot of the GUI is shown in Fig. 6. The GUI contains statistical data panel, time control, and elevator state panel.

4. Computational Experiments

4.1 Performance Evaluation of MCE Systems

Through the use of MceSim, we evaluated 4 MCE systems consisting of 6 cars in total. Figure 7 shows the performance results in terms of the average service completion time[†], when changing the call generation rate (passengers/min). We can see that $6S1C30F$ presented the highest transportation capacity, which is a valid result. Also, we can observe that transportation capacities of $3S2C30F$ and $2S3C30F$ present a high resemblance, a very interesting result.

Figure 8 shows the variation in the average service completion time of 6 MCE systems in a building consisting of 1 shaft when changing the number of cars from 1 to 6. Figure 9 shows the variation in the average service completion time in a single car shaft elevator system when changing the number of shafts from 1 to 6. In Fig. 9 with the increase in the number of shafts the transportation capacity increases linearly, but in Fig. 8 with the increase in cars the transportation capacity tends to decrease.

The above results are consequent as MCE systems. These results show that MceSim is a fully functional MCE simulator.

4.2 CST Solution Competition 2007

MceSim was used in the group controller performance competition CST Solution Competition 2007 [14][††]. Initially 14 teams, from different universities in Japan, nominated the competition, and finally 8 teams performed their proposal. We have reported the evaluation results in [15], please refer the report for more detail.

The proposed solutions used different kinds of methods such as GA, AHP, a branch and bound method, a multi-start local search, zone division, home positioning etc. This shows MceSim as a fully functional testbed for the MCE group control problem.

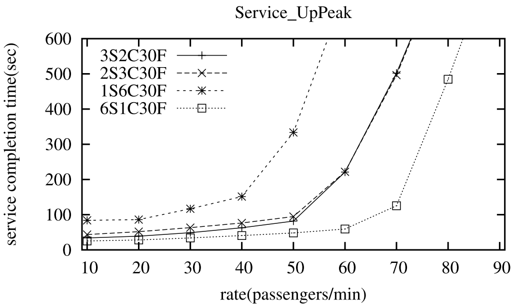


Fig. 7 Service completion time for 6 car buildings.

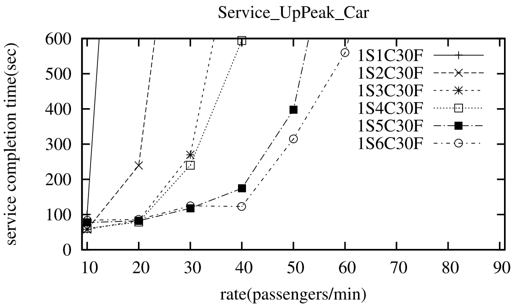


Fig. 8 Service completion time for 1 shaft buildings.

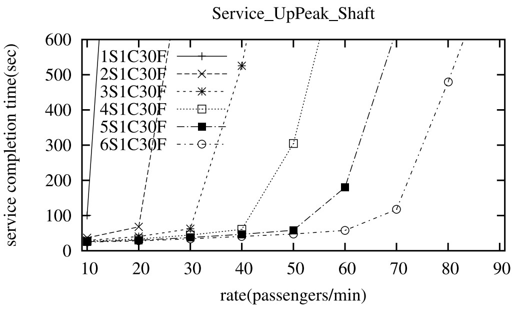


Fig. 9 Service completion time for 1 car/shaft buildings.

5. Conclusion

In this paper, we introduced MceSim. We described the static structure and the behavior of MceSim. In the experiments section, we evaluated MCE systems and group controllers by using MceSim. The group controller problem used in the competition is an optimization problem. However the elevator system is a good target for many researches: discrete event systems, hybrid dynamical systems, non-linear systems, formal methods, etc. [16]–[18]. MceSim is an open source software available for the public. It can be used not only for the optimization of the MCE control, but also for other problems on MCE systems. We hope that MceSim will be used in many investigations.

[†]The service completion time is time since the call was generated until the passenger transportation is completed.

[††]http://www.ieice.org/˜cst/compe07/

Acknowledgments

This competition was adapted as one of the activation projects of the Institute of Electronics, Information and Communication Engineers, Engineering Sciences Society, Japan. We note the use of the activation fund and express our gratitude.

References

[1] T. Yumura, T. Maeda, K. Funai, S. Nozawa, and K. Koyama, "Roped multi-elevator concept and the system performance," Proc. JSME Elevator, Escalator and Amusement Rides Conf., pp.1–4, 1997.

[2] A. Fujino, T. Tobita, and K. Nakagawa, "Basic study on mass transportation systems in buildings by means of multiple-cage elevator," IEEJ Trans. IA, vol.117, no.7, pp.815–822, 1997.

[3] M. Miyatake, T. Koseki, and S. Sone, "A proposal of a ropeless lift system and evaluation of its feasibility," IEEJ Trans. IA, vol.119, no.11, pp.1353–1360, 1999.

[4] T. Sudo and S. Markon, "The perfrmance of multi-car linear motor elevators," Elevator Technology 11 Proc. ELEVCON 2001, pp.141–149, 2001.

[5] J. Gale, "ThyssenKrupp's TWIN lift system," Elevator World, vol.51, no.7, pp.51–53, 2003.

[6] S. Yamashita, M. Iwata, S. Hikita, and K. Koyama, "A study on design method of multi-car elevator system," IEEJ Trans. IA, vol.125, no.9, pp.862–870, 2005.

[7] K. Shiraishi, T. Hamagami, and H. Hirata, "Multi car elevator control by using learning automaton," IEEJ Trans. IA, vol.125, no.1, pp.91–98, 2005.

[8] S. Takahashi, H. Kita, H. Suzuki, T. Sudo, and S. Markon, "Simulation-based optimization of a controller for multi-car elevators using a genetic algorithm for noisy fitness function," Proc. 2003 Congress on Evolutionary Computation, vol.3, pp.1582–1587, 2003.

[9] H. Suzuki, S. Takahashi, Y. Sano, T. Sudo, S. Markon, and H. Kita, "Simulation-based optimization of multi-car elevator controllers using a genetic algorithm," Trans. SICE, vol.40, no.4, pp.466–473, 2004.

[10] S. Markon, K. Ikeda, H. Kita, and H. Suzuki, "Direct control of multi-car elevators with real-time GA," Proc. 11th International Conference on Intelligent Engineering Systems, pp.191–194, 2007.

[11] K. Ikeda, H. Suzuki, S. Markon, and H. Kita, "Designing traffic-sensitive controllers for multi-car elevators through evolutionary multi-objective optimization," Proc. 4-th Intl. Conf. on Evolutionary Multi-Criterion Optimization, LNCS 4403, pp.673–686, 2007.

[12] K. Ikeda, H. Suzuki, S. Markon, and H. Kita, "Traffic-sensitive controllers for multi-car elevators: Design, multi-objective optimization and analysis," Proc. SICE 2007, pp.2655–2662, 2007.

[13] H. Suzuki, "A study on intra-shaft operating method for multi-car elevators," IEICE Technical Report, CST2007-7, 2007.

[14] S. Yamaguchi, T. Miyamoto, N. Uchihira, Q.-W. Ge, and S. Honiden, "CST solution competition 2007 (summary)," Proc. IEICE Gen. Conf. 2008, AI-1-5, pp.SS-31–SS-32, 2008.

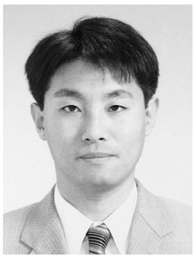
[15] T. Miyamoto, S. Yamaguchi, N. Uchihira, Q.-W. Ge, and S. Honiden, "CST solution competition 2007 (detailed results of computational evaluation)," IEICE Technical Report, CST2008-06, 2008.

[16] T. Hikihara and S. Ueshima, "Emergent synchronization in multi-elevator system and dispatching control," IEICE Trans. Fundamentals, vol.E80-A, no.9, pp.1548–1553, Sept. 1997.

[17] E.-M. Kim, S. Kusumoto, T. Tsuchiya, and T. Kikuno, "An approach to safety verification of object-oriented design specification for an elevator control system," Proc. Third Intl. Workshop on Object-Oriented Real-Time Dependable Systems, pp.256–263, 1997.

[18] J. Hoenicke and P. Maier, "Model-checking of specifications integrating processes, data and time," Proc. Intl. Symp. of Formal Methods Europe, LNCS 3582, pp.465–480, 2005.

Toshiyuki Miyamoto was born in Kobe, Japan, on February 20, 1970. He received his B.E. and M.E. degrees both in electronic engineering from Osaka University, Japan, in 1992 and 1994, respectively, and received Dr. of Eng. degree in electrical engineering from Osaka University, Japan in 1997. From 2000 to 2001, he was a visiting researcher in Department of Electrical and Computer Engineering at Carnegie Mellon University in the United States. He is currently an associate professor of Department of Electrical Electronic and Information Engineering, Osaka University. His area of research interests includes theory and applications of concurrent systems and multi agent systems. He is a member of IEEE, SICE, and ISCIE.



Shingo Yamaguchi received the B.E., M.E. and D.E. degrees from Yamaguchi University, Japan, in 1992, 1994 and 2002, respectively. He was a visiting scholar in the Department of Computer Science at University of Illinois at Chicago, United States, in 2007. He is currently an associate professor in the Graduate School of Science and Engineering, Yamaguchi University, Japan. His research interests are in the area of net theory and its applications. He is a member of IEEE and IPSJ.