Simulation of Building Traffic

and

Evacuation by Elevators

Henri Hakonen

Licentiate Thesis

Helsinki University of Technology
Department of Engineering Physics and Mathematics
Systems Analysis Laboratory

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF LICENTIATE THESIS
DEPARTMENT OF ENGINEERING PHYSICS AND MATHEMATICS
April 29, 2003

Author:	Henri Hakonen
Department:	Department of Engineering Physics and Mathematics
Major subject:	Applied Mathematics
Minor subject:	Computer and Information Science
Title:	Simulation of Building Traffic and Evacuation by Elevators
Title in Finnish:	Taloliikenteen simulointi ja evakuointi hisseillä
Chair:	Mat-2 Applied Mathematics, Professor Raimo P. Hämäläinen
Supervisor:	Professor Raimo P. Hämäläinen
Instructor:	Dr. Tech. Marja-Liisa Siikonen

Abstract:

This thesis describes the structure, functionality, and use of Building Traffic Simulator (BTS). The evacuation of tall buildings was studied by simulations. The simulator has versatile possibilities to define traffic situations, elevator groups, escalators and staircases. Simulations are visualized through 2- and 3-dimensional displays. The statistics of performance measures for passengers and elevator groups are calculated based on the simulation results. The main use of BTS is in elevator planning. Kone corporation uses BTS also for testing new group control algorithms.

According to the current safety standards, buildings are evacuated using stairs. Stairs are an inefficient means to empty a tall building, since stairways will become congested if the whole population leaves at the same time. Evacuation is also slowed down because of the long distance from the highest floors down to ground level. Furthermore, elderly and handicapped persons cannot use the stairs at all. There has been reconsideration of the evacuation standards of tall buildings and discussion on improving the fire safety of elevators. According to the simulations in this thesis, the elevators speed up the evacuation of tall buildings considerably.

Number of pages: 117	**Keywords:** simulation, building, traffic, simulator, elevator, lift, planning, evacuation, emergency
Department fills **Approved:**	**Library code:**

TEKNILLINEN KORKEAKOULU LISENSIAATTITYÖN TIIVISTELMÄ
TEKNILLISEN FYSIIKAN JA MATEMATIIKAN OSASTO
29. huhtikuuta 2003

Tekijä: **Osasto:** **Pääaine:** **Sivuaine:**	Henri Hakonen Teknillisen fysiikan ja matematiikan osasto Sovellettu matematiikka Informaatiotekniikka
Työn nimi: **Title in English:** **Professorin koodi ja nimi:** **Työn valvoja:** **Työn ohjaaja:**	Taloliikenteen simulointi ja evakuointi hisseillä Simulation of Building Traffic and Evacuation by Elevators Mat-2 Sovellettu matematiikka, professori Raimo P. Hämäläinen Professori Raimo P. Hämäläinen TkT Marja-Liisa Siikonen

Tiivistelmä:

Työssä kuvataan taloliikennesimulaattorin (Building Traffic Simulator) rakennetta, toimintaa, ja käyttöä. Työssä on simuloitu korkeiden talojen tyhjentämistä. Simulaattorissa on monipuoliset mahdollisuudet määritellä erilaisia liikennetilanteita, hissiryhmiä, liukuportaita ja portaikkoja. Simulaatioita havainnollistetaan 2- ja 3-ulotteisilla näkymillä. Simulaatiotuloksesta voidaan laskea erilaisia matkustajiin ja hissiryhmiin liittyviä tunnuslukuja. Taloliikennesimulaattorin pääasiallinen käyttökohde on talon hissiryhmien mitoitus. Kone Oyj käyttää simulaattoria myös uusien hissiryhmäohjauksien testaamiseen.

Nykyisten ohjeiden mukaan talo evakuoidaan portaita käyttäen. Portaat ovat hidas tapa tyhjentää korkea talo, sillä portaikossa syntyy tungosta, jos kaikki poistuvat yhtä aikaa. Syynä hitaaseen evakuointiin on myös se, että kävelymatka ylimmistä kerroksista on pitkä. Lisäksi iäkkäät tai liikuntavammaiset eivät välttämättä pysty käyttämään portaita ollenkaan. Korkeiden talojen evakuointisuunnitelmia ollaankin harkitsemassa uudelleen, ja keskustelua on ollut myös hissien paloturvallisuuden parantamisesta. Tämän työn simulointien perusteella hissit tehostavat huomattavasti korkeiden talojen tyhjentämistä.

Sivumäärä: 117	**Avainsanat:** simulaatio, rakennus, liikenne, simulaattori, hissi, suunnittelu, evakuointi, hätätilanne
Taytetään osastolla **Hyväksytty:**	**Työn sijaintipaikka:**

Preface

Building Traffic Simulator (BTS) is based on the ALTS simulator (Advanced Lift Traffic Simulator) developed in Kone (Siikonen 1989). ALTS was completed by the end of 80s and it was made for the PC environment and DOS operating system. A Windows version of ALTS was developed after the mid-90s. ALTS simulators can be used to simulate one elevator group. At the end of the 90s, the simulator was extended so that it could simulate traffic of an entire building containing several elevator groups (Leinonen 1999). Escalators were also designed but not fully implemented. At the same time, the user interface and simulator displays were completely renewed and the name was changed to BTS. In 2001, the old ALTS simulation core was replaced with an event-driven simulator. In 2002, a statistics application, escalator and stair models were completed. A building parameter editor will be completed in 2003.

Acknowledgements

This thesis study was carried out at the Systems Analysis Laboratory of Helsinki University of Technology. The work was financed by Kone and Tekes. The work on the simulator core, statistics program and smaller improvements of BTS were part of the PNET project in the years 2000 and 2001. The work continued as the Schedule-project in 2002. During the project, Tuomas Susi and the author developed passenger models, a testing environment for destination control and a statistics program for BTS, while Dr. Marja-Liisa Siikonen supervised the work. My sincere thanks to Tuomas Susi, Dr. Marja-Liisa Siikonen, Rasko Leinonen and Timo Harjunen, who participated in the development and testing of BTS, and to Aki Ruotsalainen and the staff of Cybercube Oy who developed the 3-D graphics. Thanks to Jari Ylinen and Toni Rintala, BTS now uses the latest group control algorithms.

The articles were written in co-operation with Dr. Marja-Liisa Siikonen and Tuomas Susi. Dr. Siikonen has given much advice during the course of this thesis. I would also like to thank Professor Sampo Ruuth and Professor Raimo Hämäläinen for their interest and support during the work.

Henri Hakonen
Espoo, April 29, 2003.

Contents

Articles

[1] Siikonen M-L., Susi T., Hakonen H. (2001). Passenger Traffic Flow Simulation in Tall Buildings, Elevator World, August, pp. 117-123.

[2] Siikonen M-L., Hakonen, H. (2002). Efficient Evacuation Methods in Tall Buildings, Elevator Technology 12, Proceedings of Elevcon 2002, pp. 237-246, ISBN 965-90338-1-8.

[3] Hakonen H., Susi T., Siikonen M-L. (2003). Evacuation Simulation of Tall Buildings. Proceedings of CIB-CTBUH International Conference on Tall Buildings. Kuala Lumpur Malaysia 21-23 October 2003 (to be presented).

[4] Hakonen H. (2003). Building Traffic Simulator Software Document. Kone, internal report.

1 Building Traffic Simulator

Building Traffic Simulator (BTS) is a software system for simulating passenger traffic in an arbitrary building. BTS has been designed with three main purposes: 1) to analyze the performance of an elevator system; 2) to demonstrate elevator systems for customers, and 3) to test elevator group control software.

Analysis of the performance of an elevator system is the most important use of BTS. Nowadays simulators are often used for elevator planning, since the simulations are more flexible than analytical methods, which can be applied to basic cases only. In addition to basic elevator systems and the usual traffic situations, BTS is used to study complex buildings such as skyscrapers and systems including escalators and stairs. BTS can also be used to study special traffic situations including evacuations. Elevators have been regarded as an unsafe means of evacuation, but especially after September 11, 2001, there has been debate on whether evacuation practices should be changed. Evacuation can be considered as a very intense down-peak traffic situation, which can be simulated using BTS.

It is important that an elevator company can clearly demonstrate to their customers how the planned system works in realistic cases. BTS offers high-quality 3-D graphics tools to visualize the properties and behavior of an elevator system without going into details. Even though simple 2-D graphics show all the necessary information and is sufficient for traffic planner or algorithm designer, schematic displays do not give the same sense of reality as 3-D graphics.

BTS is also an important tool for testing elevator group control software. Group control algorithms that are under development may contain errors that cause rare malfunctions or decrease elevator performance. Problems that are difficult to detect just by running simple test cases can be revealed by simulating different traffic situations.

Figure 1 shows the basic tasks that can be executed by the user. The user can create and edit buildings, define simulation cases, run simulations, playback old simulations, and display statistics. The parameters of the simulation models are stored in building databases. Simulations produce log files for playback or statistics generation.

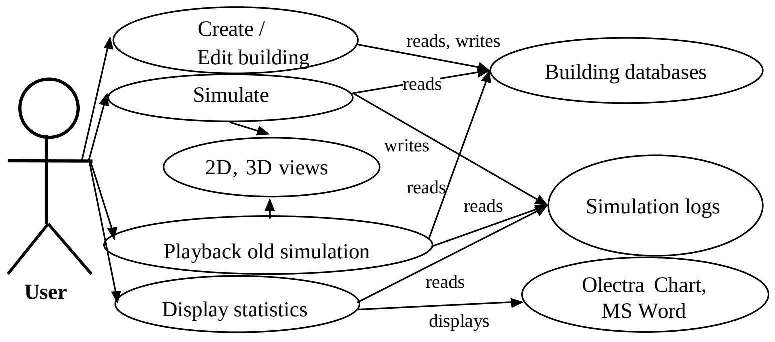


Figure 1. Usage of BTS.

2 Simulation

2.1 Existing Simulation Software

Central problems when developing a simulator are how to manage concurrency, how to collect statistics, and how to interact with the user. The main approaches to support simulator development are 1) through dedicated *simulation languages*, 2) *simulation libraries,* and 3) *general-purpose simulation environments*. The first programming language that was designed particularly for developing simulators was *Simula* (Mitrani 1982; Pooley 1987). Simula supports the process-oriented approach of modeling concurrency through a pseudo-parallel execution mechanism based on co-routines. Simula was also the first object-oriented programming language. Object-orientation is useful because it supports modular system development and makes it easy to provide additional support through class-libraries. However, dedicated simulation languages never gained an important role in simulator development. The other approaches were able to develop more rapidly owing to advances in general software technology. Many of the novel features of simulation languages are now incorporated in general-purpose programming languages, such as C++ or Java. An efficient way to implement simulators in generic programming languages is to use special simulation libraries, such as Sim++ and SimJava (Howell & McNab 1998). Libraries can support statistical computations and the building of graphical user interfaces reasonably well, but concurrency management is not as straightforward. *General-purpose simulation environments* often provide a graphical user interface and animations for visualization.

Typically, simulation environments compute automatically generic statistics of the simulation, such as number of entities, average waiting times and processing times. Some environments also provide statistical tests. Arena is an example of such system (Kelton, et al. 1998). Other environments include AutoMod, Extend, ProModel and Simul8 (Schriber & Brunner 2001; Swain 2001). Simulation environments make it easy to create simple simulations without programming, but these environments are not as flexible and efficient as simulation libraries. Some simulation environments, for example Arena, include a built-in simulation language that can be used for more detailed customization of the simulator. A simulation environment is a good choice when the system can be modeled using the usual discrete event simulation components such as queuing systems. The application areas include manufacturing, material handling, transportation, health care and communications.

2.2 Tools for Building Traffic Simulator

Implementing a simulation for a particular case is easier than implementing a simulator that can handle a wide variety of cases. The latter has requirements that are difficult to satisfy, whether a simulation library or general-purpose simulation environment is used. If a simulation environment was chosen, it should be flexible and computationally efficient, since it should be able handle hundreds of queues, behavior rules of thousands of passengers and the dynamics of tens of elevators. Moreover, the reliability should be tested carefully, since BTS is an uncommon application and it would probably use many rare features.

For some users BTS is not primarily a simulator, but an elevator planning tool. These users are not interested in learning about discrete event simulation or how to use a general-purpose simulation environment. If the simulator was based on a general-purpose simulation environment, the simulator should be made to look like a traffic planning tool instead of a generic simulator. This could be achieved by hiding the simulator and creating a custom user interface. The build-in animations of the simulation environment could be difficult to adjust for elevator simulations. Especially 3-D graphics requires a lot of work in any case. Statistics of a general-purpose simulation environment provide some important functions, but do not include most of the necessary performance measures. If the graphics, user interface and statistics were customized, a simulation environment would not have more to offer than a simulation library.

The basics of simulators are described in the literature (Banks 2000; Kelton, et al. 1998; Law & Kelton 2000). There are many ready-made libraries. On the other hand, a simulation library is only a few percent of the whole application. Therefore, the decision was made to base BTS on a new implementation of a simulation library. The event-oriented approach was chosen, since the memory requirements of a process-oriented simulator would be greater. C++ was chosen as the implementation language because it is efficient and widely used.

2.3 Discrete Event Simulation

Simulation can be defined as imitation of the operation of a real-world process or system over time (Banks 2000). A good *simulation model* is equivalent to the real system in important aspects, but simpler and easier to study. *Discrete event simulation* is a type of simulation where time advances in steps and *events* cause stepwise changes to the system state. Discrete event simulation is applied to problems where the timing of activities is the main interest. Uncertain and imprecisely known factors are modeled using randomness. The system is divided into entities rather than trying to model it as one big finite state machine. An *entity* is something that flows through the system or something that stays. The former is called a temporary entity and the latter a permanent entity. Temporary entities can, for example, be parts that arrive according to a stochastic distribution, flow through the system and exit the system after processing. Permanent entities can be, for instance, machines processing the parts with stochastically distributed processing times. *Attributes* are used for defining the states and properties of individual entities. Each machine can, for example, contain as attributes the mean and standard deviation of the stochastic processing time.

The *simulation timer* maintains the simulation time and sends timer events to the entities. The simulation timer contains a *simulation clock* and a set of scheduled future events. The timer seeks the scheduled event that has the smallest time stamp, advances the simulation time and then sends a timer event to the corresponding entity. This is called the next-time advance approach (Law & Kelton 2000).

The usual concepts in discrete event simulation are queues and arrival processes. Queuing is an important part of discrete event simulation. A *queue* is an object responsible for ordering when the entities have to wait for something. Queues usually behave according to the FIFO (First-In, First-Out) principle, which means that the first-arriving entity leaves first. For example, if a machine that processes parts can take only one part at a time, then the parts to be processed will form a queue. The usual way to model the arrival of entities is with an *arrival process* – a permanent entity that creates temporary entities according to some random distribution. The most common case is that the arrivals are independent of each other, which is modeled using a Poisson process.

Discrete event simulation can be classified into event-oriented and process-oriented simulation (or event-scheduling approach and process approach (Law & Kelton 2000)). *Event-oriented* simulation concentrates on handling and sending events while *process-oriented* simulation has the entity's point of view. There are differences in the event-handling mechanism and delays are handled differently, which yield major implementation differences. The performance of an event-oriented simulator is usually better.

Events control everything in *event-oriented* simulation. The simulation timer sends timer events to entities, which respond to them. The response includes state change and sending events to other entities. Event procedures are executed in this way one at a time until the simulation ends. The *event-dispatching* system, which delivers events from one entity to another, can be implemented efficiently. Unfortunately, since the events control everything, the program needs many event procedures. The event-dispatching system also obscures the sender of an event and the state of the entities at the sending moment. This means that the program is sometimes difficult to follow because even a simple operation may require the execution of several event procedures and the entities require many additional state variables in order to control the execution path.

Typically, an entity in *process-oriented* simulation has a main program that starts execution when the entity is created and ends when the entity exits the system. *Waiting*, *deactivation* and *activation* of another entity are the basic operations. The difference is that, while an event-oriented simulation executes an event procedure and exits, a process-oriented simulation executes a piece of code and waits. The process-oriented approach yields often simple and clear programs, since the program state, i.e. the execution line and local variables, contains part of the state information so that there is no need for as many state variables.

For example, when the algorithm contains several wait-commands and a wait operation finishes, we know trivially that the next line after the wait-command will be executed next in process-oriented simulation. In event-oriented simulation, the wait-command does not pause the execution of the algorithm. Instead, an end-wait event is scheduled and the timer calls the event procedure after the wait has been ended. The mechanism does not tell which of the waits was carried out and this has to be explicitly checked from somewhere, which requires an additional state variable and if-clause. There is an example in Table 1. The process-oriented case has the entity's point of view and it has a single pass execution. The event-oriented case handles each event in a separate event procedure call.

Table 1. Example of process-oriented and event-oriented simulation.

Process-oriented way	Event-oriented way
`hold, wind, bend, finish: The methods of the entity.`	
`deactivate``: holds the execution of entity until another entity calls activate.` **`wait`**`: holds the execution for the specified time.`	**`wait`**`: schedules timer event.` `activate: activation of entity by another entity.` `end_wait: called by timer when specified time has been elapsed.`

```
void execute() {
begin:
    deactivate();
    wait(2);
    hold();
    if(random(0, 1) == 0) {
        wait(10);
        wind();
    }else{
        wait(5);
        bend();
    }
    wait(1);
    finish();
    goto begin;
}
```

```
enum { ACTIVATE, HOLD, WIND, BEND, FINISH
     } nextOperation; // state variable
void activate() {
    if(nextOperation == ACTIVATE) {
        nextOperation = HOLD; wait(2);
    }
}
void end_wait() {
    if(nextOperation == HOLD) {
        hold();
        if(random(0, 1) == 0) {
            nextOperation = WIND; wait(10);
        }else{
            nextOperation = BEND; wait(5);
        }
    }else if(nextOperation == WIND) {
        wind();
        nextOperation = FINISH; wait(1);
    }else if(nextOperation == BEND) {
        bend();
        nextOperation = FINISH; wait(1);
    }else if(nextOperation == FINISH) {
        finish();
        nextOperation = ACTIVATE; }
}
```

Process-oriented simulation is implemented by allocating a *stack* for each entity. There must be a lot of stack space because the need is difficult to predict and the overflow causes a program crash. This is no problem if the number of concurrent entities is small, but otherwise the approach wastes memory.

Object-oriented programming is a natural way to implement entities. Each entity is an *object* maintaining its own state information. It behaves and interacts with other objects according to rules, which are implemented as *methods* of the objects. A simulation entity can also be seen as a *finite state machine*. A finite state machine has a finite number of states, an initial state and state transition function. The state transition function takes the current state, input event and returns the next state and a set of output events. A finite state machine is a useful point of view, since the modeling often leads to a design where the objects have more than one state. There is an established notation for *state diagrams*, which makes it easier to specify and understand the objects.

3 Elevatoring of Tall Buildings

3.1 Elevator Planning

The objective of *elevator planning* is to find suitable elevators for a building. Elevator planning uses both simulation and analytical methods. The following *performance measures* are used at the planning stage: (Barney & dos Santos 1985; Siikonen 1997)

- *Handling capacity* is the maximum number of passengers that can be transported with an 80% load factor in up-peak within a certain time. The value is usually given in persons per five minutes or percentage of population per five minutes.
- *Round trip time* for up-peak is the time required for an elevator to serve car calls given from the entrance floor and the return back to the lobby. The round trip time starts from the door opening at lobby and ends with door opening.
- *Interval* is the average time between elevator departures from the entrance floor during up-peak. It equals round-trip time divided by the number of elevators in the group.
- *Call time* is the time required to serve a landing call, which is time between call registration and cancellation. A call is cancelled when the elevator starts to decelerate to the landing call floor.
- *Waiting time* is time between passenger arrival at the landing call floor and entrance to the elevator.
- *Travel time* is the elevator running time from the lowest served floor to the highest

served floor with nominal speed.

Up-peak means an intense traffic situation, where passengers arrive at the entrance floor and head to populated floors.

At the elevator planning stage, often the building does not yet exist and there is no accurate information of the traffic. The required handling capacity can be calculated according to the estimated population, number of floors, floor heights and type of building. If the population is not known in advance, a rough estimate in a typical office building is 10-25 m^2 per person of the net area. The required handling capacity is 12-20 % / 5 minutes for an office, and 5-7.5 % / 5 minutes for a residential building (Kone 2000). In addition to the requirement for up-peak handling capacity, there is also a requirement for the interval, to restrict waiting times. In an office building, the interval is good if it is between 10-30 seconds. Typically, a group having few big elevators has a long interval, but a group having many small elevators has a short interval, even though the two groups might have the same handling capacity.

3.2 Elevator Arrangements

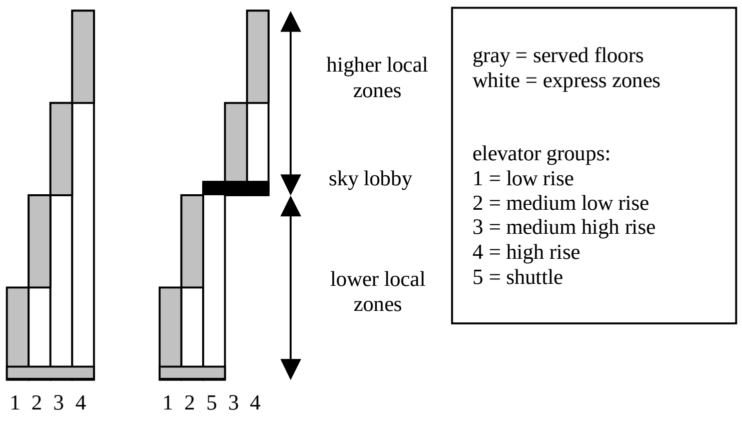


Figure 2. A tall building with four zones without a sky lobby, and with a sky lobby**.**

Elevators require a lot of space in tall buildings. The reason is that when the height of the building is doubled, also the population and travel distances are about doubled. Using faster elevators partly compensates longer distances, but there have to be more elevators to handle the increased population. Since there is only one elevator in each shaft and the shafts are long, rentable space is reduced considerably. There are ways to improve the efficiency, as shown in Figure 2. Elevator groups can be arranged into *zones* that are served by low-rise, mid-rise and high-rise groups. The low-rise group serves the lowest floors and the high-rise the highest. The number of stops will be smaller compared to an arrangement where all the elevator groups serve all floors. The low-rise elevator group does not have to be fast. *Sky lobbies* can be used in buildings with more than 40 floors. This arrangement has a fast *shuttle* elevator group, which transports passengers between the ground lobby and sky lobby. Local elevator groups take passengers to their destination floors. The drawback is that passengers have to change elevators in order to reach the upper floors. Elevator systems for high-rise buildings are discussed by Barney (Barney 2003) and Schröder (Schröder 1984).

Another way to improve space efficiency is a *double-deck elevator*. A double-deck elevator has two cars on top of each other, which can take twice as many passengers. Both cars can serve calls simultaneously at sequential floors. The transportation capacity is increased by 50% - 100% compared to a single-deck elevator with the same shaft. Two–storey, high-entrance lobbies are necessary, so that both cars can be loaded at the same time.

A *Destination Control System* (DCS) has destination call stations (terminals) on landing floors. A passenger dials the destination call for a landing floor and the call station shows the number or the letter of the serving elevator. There is no need for conventional landing call or car call buttons. The group control can utilize the destination floor information by arranging the passengers who have the same destination floor into the same car. Therefore, the number of stops is reduced, the round-trip time becomes shorter and the handling capacity increases. The best possibility to utilize the information, and thus the biggest performance improvement, is achieved during up-peak traffic. There is also a *hybrid* alternative, where there are terminals at the main entrance, but conventional buttons elsewhere.

4 Group controls

4.1 Elevator Groups

Elevators are located close to each other whenever possible, so that a passenger can enter any of them from a common *waiting area*. A passenger calls an elevator by pressing a *landing call* button. There are usually two buttons, one for each direction, which means that they are common to all the elevators and any elevator can serve the call. This kind of arrangement, where elevators serve a common set of landing calls, is called an *elevator group*. Landing calls are stored into memory and served by elevators in a suitable order. An elevator serves a call by moving in the shaft, decelerating and opening doors. When the passenger is inside the *car*, he can choose the destination floor by pressing a *car call* button.

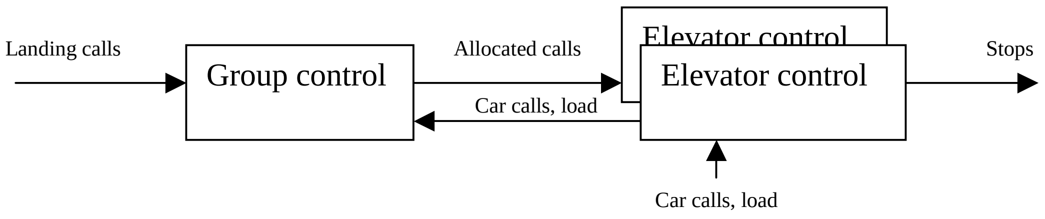


Figure 3. Elevator group overview.

Each elevator has its own *elevator control*, which handles the drive, doors and equipment inside the car. A single elevator is capable of functioning without group control, since the elevator control orders the elevator to operate according to the calls, open and close the doors and so on. However, an elevator group consisting of several elevators has a common *group control* (Figure 3), which increases the efficiency of the elevators. The main purpose of group control is to allocate each landing call to the most suitable elevator. The result of the allocation is sent to the elevator control, which plans the drive and door commands accordingly. *Parking* means that a vacant elevator is sent to wait on a floor where the traffic is greater. For example, it is often useful to have an elevator waiting at the main entrance. Group control also collects the statistics of traffic, which is utilized in call allocation.

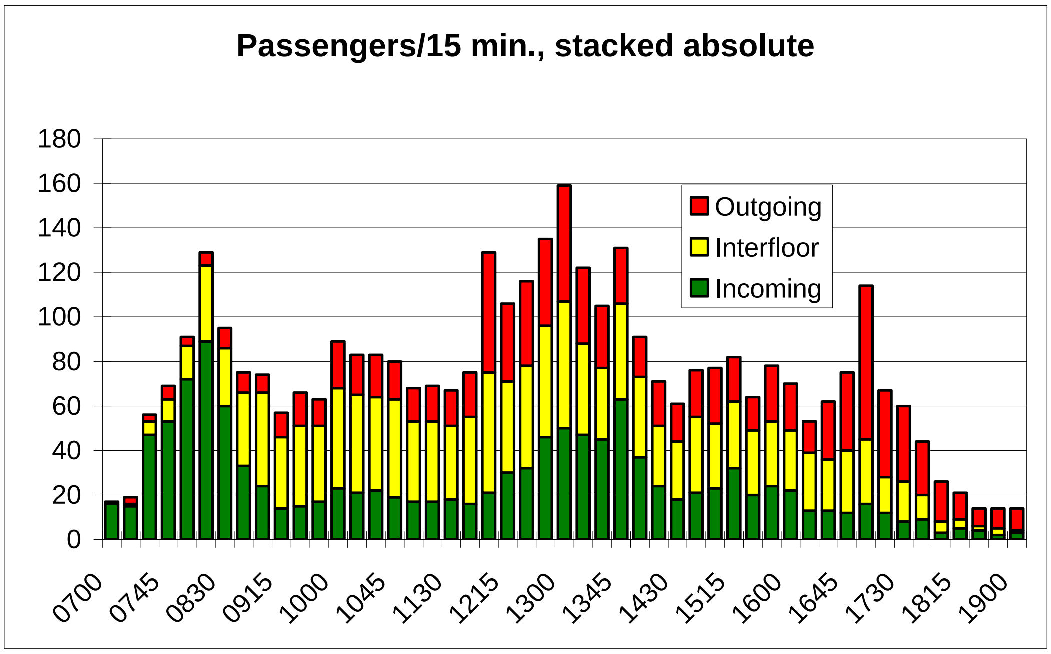


Figure 4. A typical traffic profile of an office building (Siikonen 1997).

Figure 4 represents the traffic profile of an office building. There is an *up-peak* in the morning when people come to work. The morning up-peak traffic is usually the most difficult with respect to handling capacity, so it is often adequate to plan the elevators according to that. Often employees have a lunch break, during which they go down and come back afterwards. *Lunchtime* intensifies the traffic at midday. After the workday, people exit the building, which causes a *down-peak*.

4.2 Landing Call Allocation

The elevator group controls can be classified into immediate and continuous allocation. *Immediate allocation* means that the allocated landing call is fixed to a car and the result is shown to the user immediately by lantern and gong. Immediate allocation is essential with DCS, but the principle is used also with conventional controls especially in Japan. A system with *continuous allocation* does not fix the serving car or tell the allocation to the user until the elevator starts decelerating to the floor. The continuous allocation is executed repeatedly using, for example, a 500 ms interval and when a call is registered or canceled. The times required to unload and load the elevator vary considerably from stop to stop. When the delay is longer than expected, the system can change the allocation, which makes continuous allocation somewhat more efficient than immediate allocation.

Elevator group control allocates the best elevator for each landing call. There are many heuristic methods and optimization algorithms for achieving this goal. *Collective control* is a simple allocation principle, where an elevator serves a landing call having equal direction every time when the elevator arrives at the floor. In other words, an elevator collects the calls systematically in its direction. Unfortunately, this principle *bunches* elevators when the traffic is intense, because every time an elevator advances ahead of the others, it gets a call and falls behind the front. The bunching lengthens the average waiting times, since after the front has passed a floor, it takes time before the front has made another round trip to serve that floor again (Al-Sharif 1993).

Modern elevator group controls use industrial PC hardware. The improved computing capacity has made it possible to use *optimization algorithms*. A modern system can provide more information for call allocation than just the calls. An elevator measures the load changes and a door can count the breaks of the photocell beam. The system estimates the number of incoming and outgoing passengers in this way. A typical *traffic profile* of a day can be found by collecting the data over a long period. The information of traffic intensity can be used to *predict* the number of passengers per call, to find the best parking floors or to estimate how many new calls there will be during a time interval. The forecasts serve call allocation among other things by enabling the optimization of average waiting times, since the number of waiting passengers must be estimated for that purpose.

In last few years, KONE has developed a *genetic algorithm* for group control (Tyni & Ylinen 1999). Genetic algorithms are metaheuristic algorithms that mimic the evolution of species. Nowadays, they are widely used to compute an approximate solution to hard optimization problems. The optimization objective of a genetic group control algorithm can be set to minimize call times, waiting times, journey times or some mixed function. There are algorithms for both conventional and DCS controls. Double-deck elevators also have their own allocation algorithm (Sorsa 2000)

The problem in continuous call allocation is to allocate K landing calls to M elevators. There exist M^K possible allocations. The input consists of elevators, their positions, loads, states, nominal speeds, accelerations, heights of floors, car calls, landing calls, elevators available for the calls and traffic intensities. The problem can be formulated as a *deterministic* combinatorial optimization problem, and what the best way is to take account of the intensity information is an open problem. A *stochastic* model could be formulated, which utilizes the distribution of incoming calls and the distribution of destinations, but since group control is a real-time application there should also be an efficient method for solving this kind of problem.

5 The Main Components of BTS

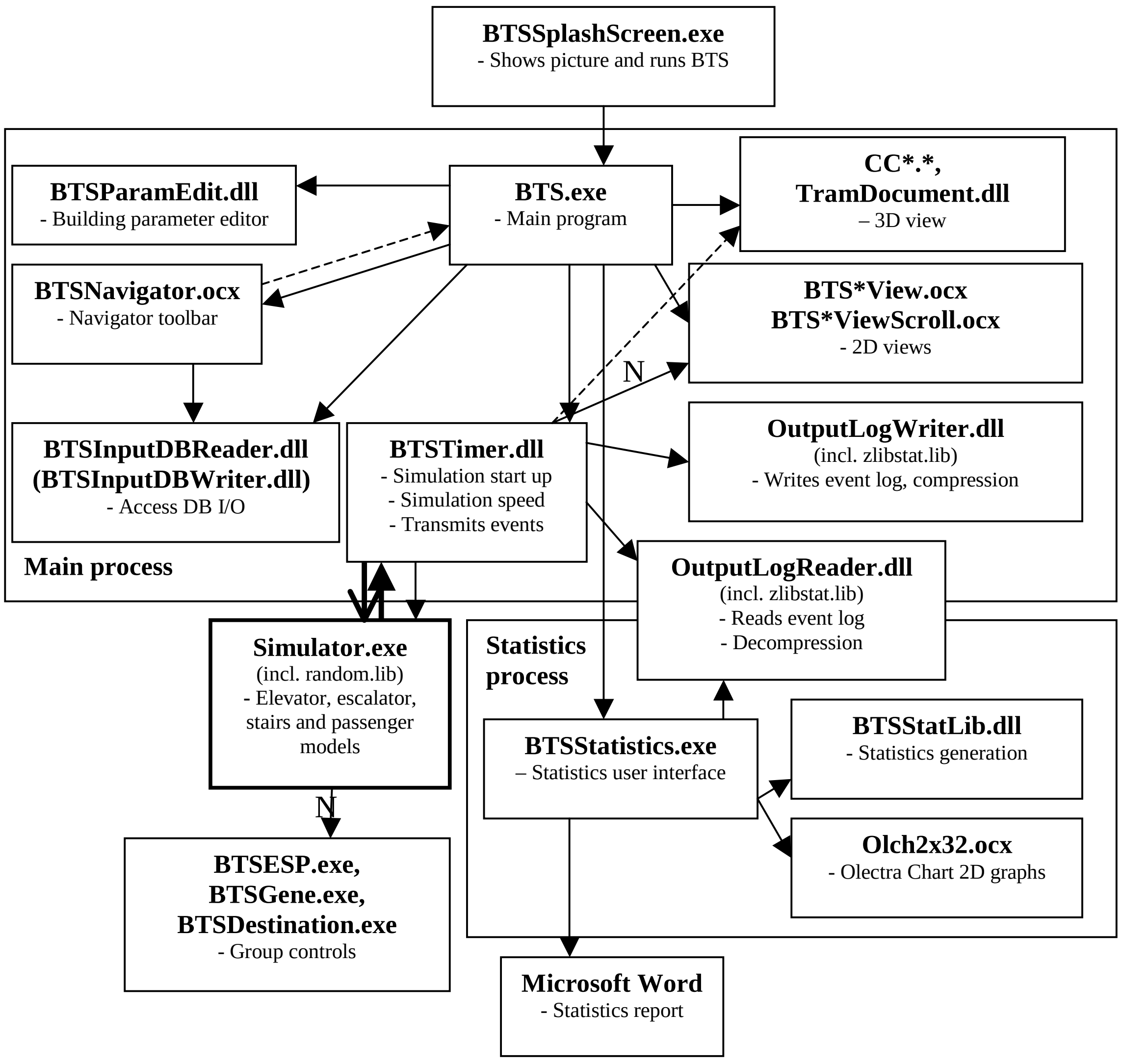


Figure 5. BTS components [4].

According to Figure 5, BTS consists of the main executable (bts.exe) and components that are either executables or utilize Windows Component Object Model (COM) technology (Grimes 1998; Bates & Tompkins 1998). The main components of BTS are the building parameter editor, database components for storing the data, simulator executable (simulator.exe) and group control components. There are displays for showing the ongoing simulation and a statistics program for displaying simulation results. The simulator executable is perhaps the most important and complex part, since it contains the main simulation loop and all the simulation models, including passenger and elevator models.

5.1 BTS Main Program

The main program (bts.exe) controls the main window and menu bars. One building database is open at a time. The opening and closing is managed by the main program. Its duty is also to launch other programs, transmit data, display error messages and execute several kinds of miscellaneous tasks. It maintains simulation speed in coordination with the simulation timer (btstimer.dll).

In the beginning of the simulation, the main program

- launches the simulator executable,
- creates a pipe between the processes,
- fetches simulation parameters using SQL-queries,
- writes the parameters into the pipe and
- starts reading events from the pipe.

The main program forwards the events to the displays and the event log file (Figure 6).

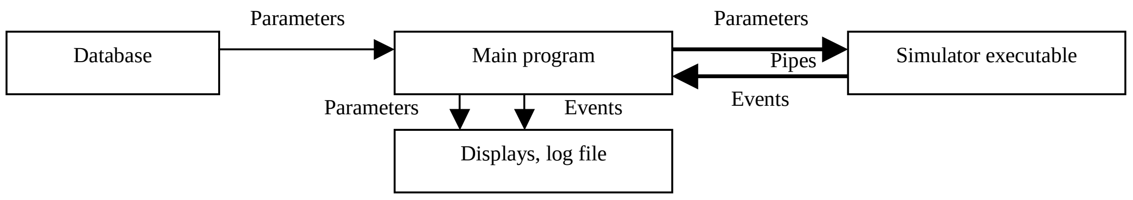


Figure 6. Data flow between the main program and the simulator executable.

5.2 Simulator Executable

The *simulator executable* reads the parameters from the input stream. In the beginning of simulation, it constructs the simulation objects, which include elevator groups, escalator groups, the passenger generator and many others. It takes care of the actual simulation. The main program simply keeps getting the next event from the event queue and delivering it until the simulation time has elapsed. The objects react on incoming events and the reactions include sending events to other objects. Since the simulation core is event-driven, there is a need for many events. Some of the events are converted to output stream format and logged, while other events are internal.

The main parts of the simulator executable are the simulator core, passenger simulator, elevator simulator, escalator simulator and stair simulator. The *simulator core* library implements the basic components of the event-driven simulator: timer, queue and event dispatching mechanism. It also contains smart pointer classes, which simplify memory management.

The *passenger simulator* generates passengers randomly according to traffic parameters (Susi 2001; Leinonen 1999). A newly generated passenger chooses the route to its destination and travels the route. There can be several elevator, escalator and stair groups. Passengers have the possibility to change transport device at certain places. This means that there are many possible routes. Some network optimizations and heuristics are necessary to choose a good route. The passenger journey consists of walking, pressing call buttons, waiting in queues and riding in transports. Simulation is focused to vertical traffic. Therefore, the walking model is not detailed: passengers walk always straight on the floor with a defined speed. Congestion can occur only at transportation devices.

The *elevator simulator* simulates door movements, elevator positions, velocities and accelerations, call buttons and waiting area lanterns. The elevator simulator is linked with a real group control algorithm, which takes care of call allocation. The *escalator simulator* models the escalators, which simply run in the same direction and never stop. The *stair simulator* models staircases. Even though the stairs are not active devices, the stair shaft dimensions and passenger queues on stairs need to be modeled. Details of the simulator models are presented in the Software Document [4].

The program design is quite efficient. The maximum simulation speed is about 10-100 times real time on a 700 MHz Pentium PC. The speed is useful, since it is a common task to make several hour-long simulations using different elevator groups and traffic situations.

5.3 Database

All the data required in the simulation and in the displays are stored in relational *database*. There is one database for each building. There is no need for a separate database server, since the amounts of data are not that big and there is no need to share the data between several simultaneous users. The choice to use a database instead of a custom file format was made because the relational model is a clear way to define the data structure and because the SQL-language is a convenient way to access the data. BTS uses Microsoft Access databases. However, it is not necessary to have Microsoft Access on every computer, since running BTS requires only the freely re-distributable database drivers.

5.4 Group Controls

Group controls algorithms take care of call allocation. The elevator group parameters include the type of group control, so that the user can choose the control group-wise. The group controls include *collective control*, *genetic allocation* and *DCS allocation*. The group controls of BTS consist of the call allocation code that is used in the elevator groups, interface code and some services of the lift operating system. During the simulation, the state of the elevator group is passed to group control, allocation is executed and the results are read back to the simulator. This takes place once in every 500 ms of simulation time. Group controls are wrapped inside *COM*-components. They act as plug-ins: in the beginning of the simulation, the system seeks among the installed group controls the COM-component that implements the specified group control and launches it.

5.5 Parameter Input

The *building parameter editor* provides a user-friendly way to define the parameters. The basic use is to create a new building or edit an existing one. The user can define floors, elevators groups, escalators, stairs, population, traffic and miscellaneous simulation parameters. The data is stored in the building database. The user interface is an ActiveX control written in Visual C++ and Visual Basic. It interacts mainly with the database using the SQL-language instead of communicating directly with the other parts of the simulator.

5.6 Simulation Displays

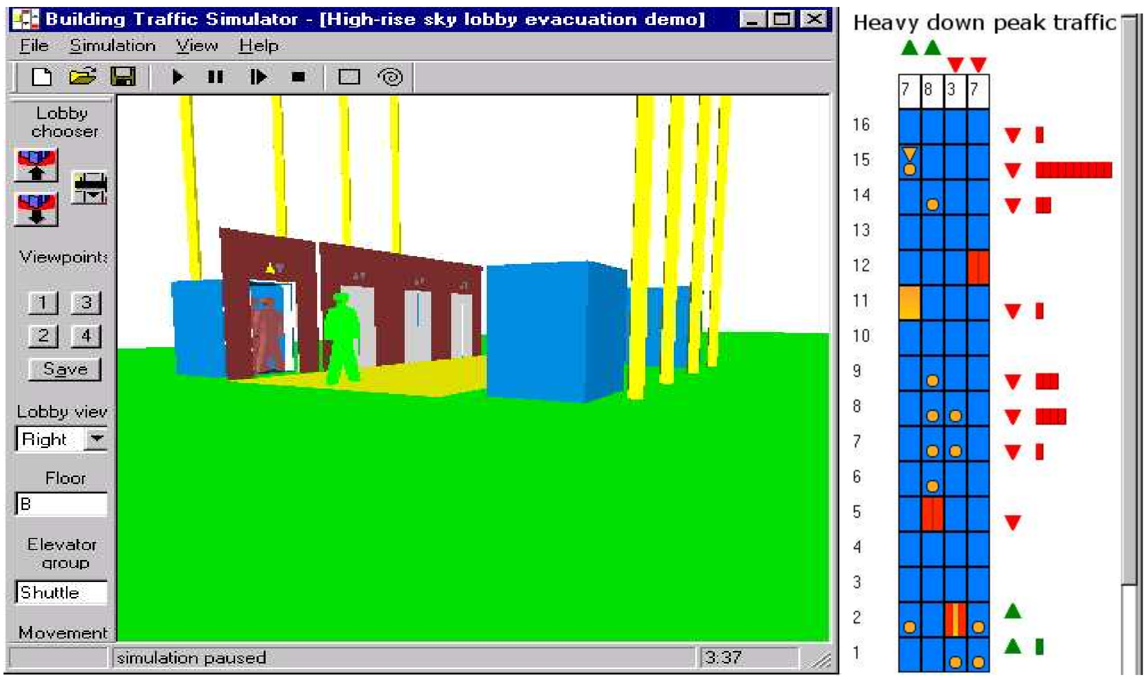


Figure 7. The main window of BTS showing a 3-D view [1] and 2-D view of an elevator group.

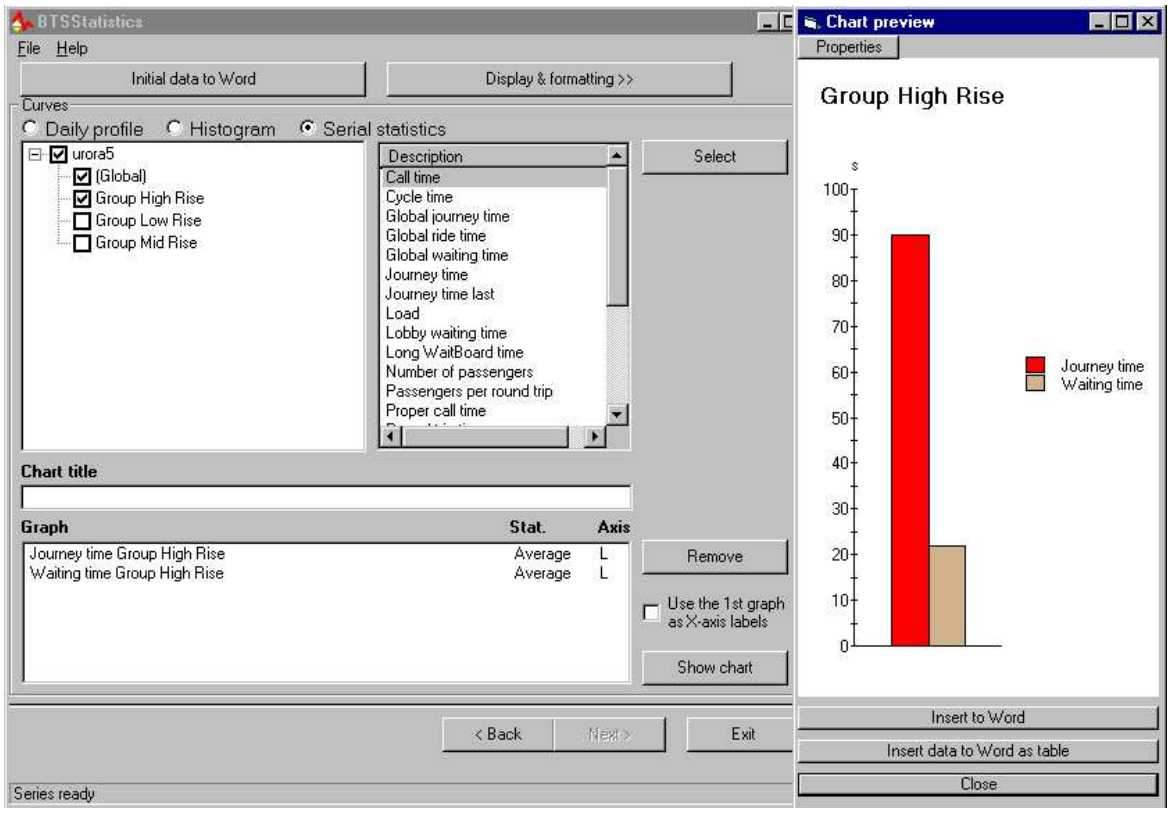
BTS has two kinds of visualizations (Figure 7). One is a *two-dimensional traffic display* that shows the traffic for an elevator, escalator or stair group. The screen updates the positions of elevators, active calls and passenger queues. The other is a *three-dimensional display* that shows the whole building. In 3-D view, the user can zoom in on certain elevator groups and floors and change the angle of view. The animation shows the shape of building, floors, elevators and passengers. The 3-D view is a component by Cybercube Oy.

There is also a *graphics designer* for 3-D views by Cybercube Oy. The distances of devices affect walking times, but otherwise the purpose of the graphics designer is to define the appearance of the building.

5.7 Statistics

The statistics program provides numerical data for evaluating the performance of the system more accurately than by watching the displays. Statistics give answers to questions like how long on average a passenger has to wait for an elevator, what the average queue length is, how fast the building is filled or evacuated, and what the estimated energy consumption is. Based on the statistics, it can be decided if the current elevator arrangement is sufficient for the assumed population. It is also easy to compare two elevator arrangements or two group control algorithms in order to choose the more appropriate one.

The statistics are generated from *simulator log files*, which contain events like "door opens", "elevator stops", and "passenger exits elevator". This data is processed into performance measure items, e.g. call times of single calls, which are then grouped into data sets and the statistics are computed from each set according to the user's choices. The *summary statistics* include averages, minimums, maximums, sums and numbers of items. The application computes also *histograms* with user-definable scales and *daily profiles* with 5-minute (or other period) averages. However, the application does not compute more complex statistics such as standard deviations, confidence intervals or t-tests. The results are displayed as *charts* or *tables* on the screen (Figure 8) or inserted automatically into a Microsoft Word document. The design decision was to separate the statistics program from the simulator itself. The benefits of the separation are simpler simulator architecture and the possibility to generate new statistics from old simulation log files. The downside is the processing time. The processing may take a few minutes, because a long simulation produces a log, which can be several megabytes in compressed form.



Curves: ( ) Daily profile ( ) Histogram [✓ Serial statistics]

- ☑ urora5
  - ☑ (Global)
  - ☑ Group High Rise
  - ☐ Group Low Rise
  - ☐ Group Mid Rise

Description:
Call time
Cycle time
Global journey time
Global ride time
Global waiting time
Journey time
Journey time last
Load
Lobby waiting time
Long WaitBoard time
Number of passengers
Passengers per round trip
Proper call time

Chart title

Graph	Stat.	Axis
Journey time Group High Rise	Average	L
Waiting time Group High Rise	Average	L

☐ Use the 1st graph as X-axis labels

Series ready

Figure 8. Statistics program.

6 Evacuation Studies

Stairs have been regarded as the only means of evacuations. The argument against elevators is that they are not built to operate in extreme conditions. In case of fire, ordinary elevators are brought down and shut down. *Fire-fighting lifts* are used to bring firemen up. If all elevators were protected from fire and water in the same way as fire-fighting lifts are, evacuation by elevators would be safer during fire. The elevators would probably also need a special evacuation mode and smoke detectors that would prevent them from stopping on floors with fire. Lobbies should be fire resistant and protected against smoke. The technical difficulties can be solved (Howkins 2000; So, et al. 2002). However, in practice the improvements will not be realized without changes in safety regulations, since they increase costs.

Fire is not the only reason for evacuation. The existing elevators can be used in evacuation, if the conditions are good. Sometimes it is important to leave the building before a given situation gets worse. A bomb threat is one such case. In addition, if the fire is far from the elevator group in large plated buildings, it is possible to use the elevators. The problem is also in evacuation instructions, which easily become complicated.

Elevators are the most efficient means of transportation in tall buildings, since they are planned to be able to handle normal rush hour traffic. People seldom use stairs and everybody does not necessary know the locations of stairs. According to evacuation drills, people tend to exit the same way they entered even if emergency exits are nearby. Handicapped or elderly people may not be able to use the stairs at all, and consequently, according to current evacuation plans, they cannot exit without assistance.

Stairs are efficient in lower buildings, since walking distances are short and the handling capacity is sufficient for the whole population. In tall buildings, (over 40 floors) stairs are efficient, when it is necessary to evacuate one or a few floors to neighboring floors. However, when it is necessary to evacuate the entire building, people have a long way to walk down from the upper floors. The staircases are planned according to the population of one floor, not the population of a whole building. For that reason, there will be overcrowding, if the whole population tries to leave at the same time (Lo, et al. 2001).

The *evacuation time* of a building is the time from the beginning of the emergency until the last person exits. The evacuation time of a person consists of reaction time and egress time. *Reaction time* is the time from the beginning of the emergency until the person starts moving towards the exits. Alarms and the people’s situation have a great effect on the reaction time. For example, reaction time is usually long for a sleeping person, and it can be even longer than egress time. In the World Trade Center explosion in 1993, the reaction times ranged from a couple of minutes to four hours (Fahy & Proulx 1997). Thus, in reality, the reaction times may differ a lot from person to person. It is possible that the simulation models underestimate the variations.

The enclosed articles focus on *egress time* because it is easier to calculate and simulate. There are flow measurements of evacuation drills (Graat, et al. 1999) which can be used for calculating handling capacities of stairs and thereby egress time (Weckman 1997). The effect of smoke is studied in the literature and there are simulators, ASERI and Building EXODUS, which are capable of simulating the effect of smoke, visibility and gases on walking speed (Schneider 2001; Weckman 1998). BTS is missing two features compared to the mentioned simulators. Firstly, it does not simulate fire. Secondly, since BTS focuses to transportation devices and vertical traffic, it is not possible to define inside walls or obstacles. Therefore, the horizontal walking model is not detailed. For these reasons, the simulations consider ideal evacuation situations.

From the simulator's point of view, the evacuation situation resembles very heavy down-peak traffic. In fact the simulator is configured to generate *out-going traffic* with an intensity of 100% of population / 5 minutes during the first five minutes, when the reaction time is five minutes. This means that the reaction time is uniformly distributed and all the people walk to the elevator lobby or staircase door in five minutes. The down-peak handling capacity with an efficient control system is about 1.6 - 2.2 times greater than the up-peak handling capacity. If an office building is planned according to 13 % up-peak handling capacity, the building can be emptied within 17-24 minutes in intense down-peak situation. A call allocation algorithm that can handle the down-peak well, can also handle the evacuation situation. In an evacuation situation, group control does not have much effect, since the elevators are filled at one stop and emptied at another stop. Several stops can be necessary to fill an elevator, if there are not enough waiting passengers.

If the elevator makes only two stops during a round trip, the following evacuation time can be calculated from equation (3). Let

- floor 0 be the exit floor,
- $P_{k,0}$ be the initial population on floor k,
- $P_{k,i}$ be the remaining population on floor k after round trip i,
- C be the car size,
- t_v be time to travel one floor distance at nominal speed,
- t_s be door opening and closing time during a stop plus elevator acceleration and deceleration time, and
- t_m be the average time for a passenger to enter or leave the car.

Round trip i consists of driving H_i floors upwards, picking M_i passengers and driving H_i floors downwards. M_i is the minimum of the remaining population on the floor and the car size:

$$M_i = \min(P_{H_i,i-1}, C) \tag{1}$$

The remaining population on floor H_i after round trip i is

$$P_{H_i,i} = P_{H_i,i-1} - M_i \,. \tag{2}$$

The procedure continues for round trips i = 1, 2, …, until after round trip N_{RT} the building is empty. In the case of one elevator, the evacuation time, $T^1_{evacuation}$, is the sum of round trip times:

$$T^1_{evacuation} = \sum_{i=1}^{N_{RT}} (2H_i t_v + 2t_s + 2M_i t_m)\,, \tag{3}$$

For a group of N elevators, the evacuation time $T_{evacuation}$ is

$$T_{evacuation} = T^1_{evacuation} / N \,. \tag{4}$$

The enclosed articles introduce BTS and discuss evacuation cases focusing on tall buildings. They compare egress times in cases where stairs and elevators are used. The recommendation is to use elevators (together with stairs) whenever it is safe, since the egress times will improve significantly.

7 Summary of the Articles

Article [1] focuses on the Building Traffic Simulator. It explains the use of simulations in elevator planning. It discusses the BTS architecture, and explains the parameters and construction of the simulation model. Some parts are now outdated, for instance the Runge-Kutta method for computing elevator dynamics has been replaced with an analytical solution of differential equations. The passenger generation and routing algorithm is explained. The possibility to choose the group control algorithm is mentioned. Two call allocation algorithms: the Enhanced Spacing Principle and Genetic Algorithm are explained in brief. The 2-D and 3-D views are presented. The most important performance measures are explained. Article [1] has an evacuation case of a 400m-high building. Floors 54-88 containing about 4,000 people are evacuated to the sky-lobby using local groups and from there to the ground floor within 30 minutes. The bottleneck in the simulation was the shuttle group operating between the sky-lobby and the ground floor. The reason is that the common practice is to plan the groups according to up-peak traffic. The down-peak handling capacity is greater than up-peak handling capacity for local elevator groups, but not for a shuttle group that has only two stops.

Article [2] focuses on evacuation by stairs and elevators, and compares the evacuation methods for buildings of different sizes. It starts by evacuation terms and practices and then goes on to study the relationship between up-peak handling capacity and egress times. The article states that modern group controls do no benefit from zoning in an evacuation situation, nor does having every 3rd floor as a *refugee floor* speed up evacuation considerably. Refugee floors (Fortune 2002) are temporary gathering places before transportation down. According to the article, mega high-rise buildings are problematic, since shuttle elevators become a bottleneck in evacuation and the stairs are inefficient. The article explains the Melinek and Booth flow model (Melinek & Booth 1975) for calculating egress time by stairs. Egress times are then computed for buildings with different numbers of population and floors. The egress times by stairs are compared to the egress time by a usual elevator system. The emergency exit staircases of the buildings are planned according to the *building codes* (Allen & Iano 1989). The codes define the maximum walking distances to an exit, minimum number of staircases and the relationship between staircase width and population. By studying typically planned office buildings, it is seen that, for example, an office building occupancy of 50 persons per floor and with over 50 floors is faster to evacuate using elevators than using stairs. There is a case where half of the population uses elevators and the other half stairs. The results show that, for example, in an office building with 30 floors and with 100 persons per floor, the egress time by the stairs is 26 minutes and by elevators 22 minutes. If half of the population uses stairs and the other half elevators, the egress time is 13 minutes. The end of article discusses the evacuation plans and the need for changing them.

Article [3] discusses evacuation by elevators and stairs, analytical formulas, BTS simulation models, and compares evacuation methods of buildings with different numbers of stairs. The article explains the passenger flow model in stairs (Barney 2003) and presents an evacuation time formula for elevators, which is based on round-trip times. The simulation models of BTS, especially the passenger generation model and passenger behavior in elevators and stairs is explained. An office building with 36 floors and a hotel with 50 floors are used as examples. The hotel has a shuttle elevator group from the ground floor to the sky lobby on floor 20, and the floors above the sky lobby are evacuated to the ground. Evacuation times by the stairs are simulated using different numbers of 1.2m-wide staircases and one case is calculated analytically for comparison. According to the results, two staircases for the studied office and three staircases for the studied hotel are required to compete with elevators. The article also compares egress times and journey times in cases where different portions of passengers use stairs and elevators. The article emphasizes the role of shuttle groups in the evacuation of a mega high-rise building.

Article [4] describes the structure and functionality of the Building Traffic Simulator. The main emphasis is on the simulation models of the simulator executable and the algorithms of the statistics program.

8 References

Al-Sharif (1993). Bunching in Lift Systems, Elevator Technology 5, *Proceedings of ELEVCON 93*, The International Association of Elevator Engineers.

Allen E., Iano J (1989). *The Architects Studio Companion, Technical Guidelines for Preliminary Design*, John Wiley & Sons, Inc, USA.

Banks J. (2000). *Discrete-Event System Simulation 3rd edition*, Prentice Hall.

Barney G. C., dos Santos S.M. (1985*). Elevator Traffic Analysis, Design and Control*, Peter Peregrinus Ltd., London.

Barney G. C. (2003). *Elevator Traffic Handbook: Theory and practice*, Spon Press, London and New York.

Bates J., Tompkins T. (1998). *Using Visual C++*, Que Corporation, USA, ISBN 0-7897-1635-6.

Fahy R.F., Proulx G. (1997). Human Behavior in the World Trade Center Evacuation, *Fire Safety Science – Proceedings of the Fifth International Symposium* pp.713-724.

Fortune J. W. (2002). New Thoughts on the Use of Elevators for Emergency Evacuations of High-Rise Buildings, *Elevator World*, November 2002, pp. 110-113.

Graat E., Midden C., Bockholts P. (1999). Complex evacuation; effects of motivation level and slope of stairs on emergency egress time in a sports stadium, *Safety Science* 31 1999, p. 127-141.

Grimes R. (1998). *Professional Atl Com Programming*, Wrox Press Inc, ISBN 1-861001-4-01.

Howell F., McNab R. (1998). Simjava: A Discrete Event Simulation Library For Java, *International Conference on Web-Based Modeling and Simulation*. Society for Computer Simulation International (SCS), January 1998.

Howkins R.E. (2000). In the event of fire - Use the Elevators, Elevator Technology 10, *Proceedings of ELEVCON 2000*. The International Association of Elevator Engineers. Also in *Elevator World*, December 2000, pp. 140-142.

Kelton W. D., Sadowski R. P., Sadowski D. A. (1998). *Simulation with Arena*, McGraw-Hill.

Kone (2000). Alta planning guide.

Law A.M., Kelton W.D. (2000). *Simulation Modeling and Analysis 3rd edition*, McGrawHill.

Leinonen R. (1999). *Taloliikennesimulaattori*, Master's Thesis, Helsinki University of Technology, Laboratory of Computer and Information Science.

Lo S.M., Fang Z., Chen D. (2001). Use of a Modified Network Model for Analyzing Evacuation Patterns in High-Rise Buildings, *Journal of Architectural Engineering*, June 2001.

Melinek S.J., Booth, S. (1975). An Analysis of Evacuation Times and the Movement of Crowds in Buildings. *Borehamwood, GB: Building Research Establishment, Fire Research Station*. (BRE Current Paper CP 96/75 FRS)

Mitrani I. (1982). *Simulation Techniques for discrete event systems*. Cambridge University Press.

Pooley R.J. (1987). *An Introduction to Programming in SIMULA*, Oxford, Blackwell Scientific Publications. (http://www.cee.hw.ac.uk/~rjp/bookhtml/)

Ruotsalainen A., Brummer A., Susi T., Leinonen R., Hakonen H. (2002). *Database Specification for Building Traffic Simulator*. Kone, internal report.

Schneider V. (2001). Application of the Individual-Based Evacuation Model ASERI in Designing Safety Concepts, 2nd *International Symposium on Human Behaviour in Fire*.

Schriber T. J., Brunner D.T. (2001). Inside Discrete Event Simulation Software: How it works and why it matters, *Proceedings of the 2001 Winter Simulation Conference*.

Schröder J. (1984). Pocket Computer Elevatoring – to Shuttle or not to Shuttle, *Elevator World*, March 1984. Pocket Computer Elevatoring – the Economy of Double Deckers, *Elevator World*, April 1984.

Siikonen M-L. (1989). *Hissiliikenteen ja ohjauksen mallintaminen tietokoneella*, Licentiate thesis, Helsinki University of Technology, Systems Analysis Laboratory.

Siikonen M-L. (1997). *Planning and Control Models for Elevators in High-Rise Buildings*, Doctoral thesis, Helsinki University of Technology, Systems Analysis Laboratory, Research Reports A68 October 1997.

So A., Lai T., Yu J. (2002). On the Development of Emergency Escape Lifts, Elevator Technology 12, *Proceedings of ELEVCON 2002*. The International Association of Elevator Engineers.

Sorsa J. (2002). *Kaksikoristen hissien optimaalinen ryhmäohjaus*, Master's Thesis, Helsinki University of Technology, Systems Analysis Laboratory.

Susi T. (2001). *Traffic Generation in BTS*. Kone, internal report.

Swain J. (2001). Simulation Software Survey. *OR/MS Today*. February 2001.

Tyni T., Ylinen J. (1999). Geneettiset algoritmit ohjausjärjestelmissä, *Sytyke ry, systeemityö* 2/99.

Weckman H. (1997). Calculational estimation of evacuation from buildings. *Research Notes* 1846, Technical Research Centre of Finland. 61 p.

Weckman H (1998). Rakennuksista poistumisen laskeminen ja simulointi, Sovellusesimerkki, *VTT Tiedotteita* 1890, Technical Research Centre of Finland. 63 p.