Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

SIMULATION BUILDING BLOCKS FOR AIRPORT TERMINAL MODELING

Alexander Verbraeck
Edwin Valentin

Systems Engineering Group
Faculty of Technology, Policy and Management
Delft University of Technology
Jaffalaan 5, 2628BX Delft, THE NETHERLANDS

ABSTRACT

Airports are an ideal application area for simulation. The processes are in a continuous state of change, are complex and stochastic, involve many moving objects, and require a good performance that can be measured in several different performance indicators. Within airports, but also between airports, the same kind of questions are answered over and over again. Often, however, new simulation models are built for each question, if possible copying some parts of previous models. Structured reuse of simulation components is rarely seen. This paper shows an approach for airport terminal modeling that departs from the assumption that reusable simulation building blocks can form the core of a powerful airport modeling tool, which is able to answer different questions at airports better and faster than traditional models. The building blocks have been implemented in the commercially available simulation language eM-Plant. Several studies carried out with this library were very successful.

1 INTRODUCTION

The world-wide airline industry is growing with about 4% per year on average – varying from 2% in the USA to almost 6% in Asia. This growth is hard to accommodate. Building new airports is almost impossible, because they should be built close to large cities to function properly. In the neighborhood of large cities space is scarce, the environmental constraints such as noise are very severe, and the NIMBY – Not In My Back Yard – protests are extreme. Solutions that can increase capacity while not increasing the infrastructure and land use are often considered best. Usually, airport extensions and measures that increase the efficiency of the existing facilities are the only available solution for the growth problem.

This makes processes on and around airports ideal for simulation studies. The systems are complex, the numbers of entities involved are high, and the systems are in a continuous state of change. The boundary conditions for solutions are interesting, because solutions have to comply with strict regulations, financial limits, and environmental conditions. Discrete event simulation is often used to model systems where complex – often logistic – processes are combined with a limited infrastructure capacity. We therefore see that simulation is often used for studying airport processes (Babeliowsky, 1997, Bitauld et al., 1997, Snowdon et al., 1998, Gatersleben and Van der Weij, 1999, Joustra and Van Dijk, 2001). When looking at the issues at and around airports that are covered by simulation studies, we see that there are a lot of different domains covered, each having close links with one or more other domains. Examples are air traffic control, airline schedule optimization, airline crew assignment, airstrip management and taxiing, airport gate planning, airport terminal and passenger modeling, airport baggage handling, air cargo handling, airport road network and parking, interterminal passenger transport, and public transport to and from the airport. As most of these application areas have relations with others, managing the interrelations between modeling studies is difficult.

Several studies have focused on one of the most important customers of the airport, the passenger. Especially in the highly dynamic situations at airports, the interaction between passengers and airport processes – airport access, parking, check-in, customs, shopping, eating and drinking, waiting, boarding, and baggage reclaim – is extremely difficult to control and predict. Passengers have a free will, and do not always behave as intended. Especially when there are delays or when the terminal capacity is near its limit, simulation studies can be a big help to support decision making for changes that will improve the airport's processes. When the passengers are satisfied, the airlines, which is the other important category of customers of the airport, also benefit.

Babeliowsky (1997) and Joustra and Van Dijk (2001) describe a number of simulation models that have been developed for terminal modeling at Amsterdam Airport Schiphol. The models are implemented in a commercial simulation language and tend to become big. The models take, as a result, quite some time to build and are hard to change and maintain. Frequent maintenance is necessary, because airports are in a continuous state of change. Furthermore, it turns out that the same type of questions are asked over and over again at different airports. For each airport new models are usually built, although the questions are quite similar. Even at one airport, new questions need to be answered again and again, due to external changes such as new flight schedules and internal changes such as terminal extensions. Changing the existing models takes quite some time, so answering new questions takes usually more time than desirable. Finally, different models often have to use the same data-sets or scenarios, such as the expected number of passengers, the airlines that use the airport, and the flight schedule.

One possible solution challenges in complex airport modeling for passenger terminal studies is to use a library of pre-configured airport building blocks (Valentin and Verbraeck, 2002). These building blocks speed up the initial modeling process, and they also help in quickly replacing model parts by alternatives. In this paper, we will study a developed library of simulation building blocks from which a model of a passenger handling process at an airport terminal can be quickly modeled. All relevant processes at the terminal are included, such as check-in, customs, shopping, boarding, and baggage reclaim. Section 2 shows some of the design choices, which are worked out in more detail in section 3, where the building blocks are shown. Section 4 demonstrates the use of building blocks by showing an example of a model of Amsterdam Airport Schiphol, and it shows how the user interface and statistical analysis are implemented.

2 DESIGN CHOICES

In our case, building blocks are not pure objects. With object orientation, the emphasis of the modeling process lies on identifying the right classes, finding the inheritance relations between the classes, and defining the attributes and methods of the objects (Joines and Roberts, 1998). With building blocks, the emphasis lies on clear functionality and clear interfaces of the building blocks.

In an iterative process, where several architectures have been proposed by a small design team, we chose an architecture that makes a distinction between four types of building blocks:

1. infrastructure building blocks
2. passenger or group building blocks
3. passenger behavior building blocks
4. control building blocks

The *infrastructure* building blocks capture the static lay-out of the terminal. Infrastructure building blocks can for instance model corridors, waiting rooms, lounges, shops, restaurants, check-in desks, etc. In general, we call this type of building block an *area*. The main characteristic of an area is that it is static in terms of the size and location in the overall graph of the terminal complex. Passengers ask 'permission' to enter an area, which is granted or delayed based on the available and used capacity of the area. A customs area has for instance a capacity of one person, meaning that a person from a neighboring area can not enter the customs area until the current person has left. Because every area has a double function of delaying a passenger and queuing passengers, a natural waiting and delay pattern occurs in the entire terminal complex. Even stairs or a lounge can hold passengers when a neighboring area is full. The areas are connected to each other, and together they form a graph of the terminal complex. Areas can have a walking distance and a size (width), and they are bi-directional or uni-directional. Based on the distance, the number of persons in the area, the uni or bi-directional character, and the autonomous speed of the area (elevator, escalator, or conveyor) an average walking speed can be calculated for a person entering the area, and thereby the time spent to reach the end of the area. At the end of the area, a person has to ask again to enter the next area. When granted, the passenger leaves the current area, and enters the next one. When the next area is full, the person stays in the area and waits in an artificial queue until access is granted. The order of the areas to go through to get from origin to destination is based on a shortest path algorithm that has access to the entire graph. Several alternatives for the shortest path algorithm are available, which are discussed in section 3.3. From the generic concept of an area, several types can be distinguished. The types we incorporated into our airport terminal library, are discussed in section 3.1.

The objects using the infrastructure are the *passengers*. In our case, we decided not to model individual passengers, but *groups* of persons instead. When studying airport processes, it is clear that a family moves in a different way through the airport than a single person. Take check-in for example. The family checks in as a group. They also have the same speed when moving through an area. They only enter the next area if they can all enter. For some individual processes – for example customs or one family member who goes shopping alone – the group might be temporarily split, and recombined later. Please note also that not all persons who are present at an airport are passengers. Crew members and support personnel also use the areas, and can be modeled just like the passengers. Passengers are often accompanied by friends or relatives when they are arriving or departing. These persons also use the infrastructure and should be modeled to get a clear indication of infrastructure use.

Different types of passengers show a different *behavior*. There are many aspects of behavior that should be taken into account when modeling airport passenger processes in detail. The kind of behavior that a group has during the stay at the airport is called a *script*. A script is a kind of small program that tells a group what to do and where to go, in a number of cases using 'if' statements to indicate the conditions under which a group might carry out an action or go to a certain place. Whether a group of persons will go shopping is for instance depending on the type of passenger and the time left till boarding. The most important attributes of a group of persons are group size, average walking speed, number of suitcases, arriving or departing, flight number, type of passenger (business or tourist), and a log of times for important activities for generating statistics.

There are special building blocks called *generators* that generate the passengers based on the flight schedule. Several choices are made for each group that is generated. The group size is based on a stochastic distribution. The type of passenger is based on a probability that is depending on the flight number. The walking speed per passenger is based on a stochastic distribution. Some flights such as charters have more inexperienced passengers than others lowering the walking speed. The group walking speed is calculated as the minimum of the individual walking speeds of the group members. The time of arrival at the airport of the departing passengers is based on a distribution function per airline with the arrival distribution of passengers. The number of accompanying persons per group based on a distribution function. The script of the group is selected from a list of scripts with a chance for giving that script. There could, for instance, be a script BUSINESS with a chance of 0.10 for a flight, and an average group size of 1, a script FAMILY with a chance of 0.45 and an average group size of 2, and a script WELL-WISHER with a chance of 0.30 and an average group seize of 2. For a flight of 300 persons, there are about 300*0.1*1 = 30 business passengers and about 300*0.45*2 = 270 family passengers in about 135 groups on this flight. In addition, there are 300*0.3*2 = 180 well-wishers.

The final category of building blocks are the *control* building blocks. Control building blocks can be control processes of the airport itself, such as allocating a reclaim belt to a certain flight, choosing a check-in desk row, or determining at which gate a flight will be handled. Control building blocks can also be artifacts that one does not see at the airport, but which are present for reduction of the complexity of the simulation model. An example is the 'shortest path algorithm' for passenger movement between the areas. In reality, there is no shortest path algorithm, but signs telling the passengers where to go. In our case, we do not explicitly model the signs, and assume that the airport indicates the shortest route to arrive at a destination.

3 AIRPORT TERMINAL BUILDING BLOCKS

For each of the categories, examples of a number of characteristic building blocks are shown, and discussed in more detail.

3.1 Infrastructure Building Blocks

The whole idea behind the infrastructure or area building blocks is to have a simple mechanism to have passenger groups move through the airport. The mechanism of entering an area, spending time in an area, and leaving the area to enter a next area is simple to understand, yet very powerful. Figure 1 shows some of the areas that can be distinguished in the airport terminal library. The main distinction is between single areas, and compound areas. The compound areas consist of one or more generic areas, which on their turn can again be compound areas. With this hierarchy, the building block concept becomes extremely powerful, as we can now include entire check-in desk rows, reclaim areas, or gate areas as building blocks in our modeling library. The check-in and the gate are shown as examples in figure 1.

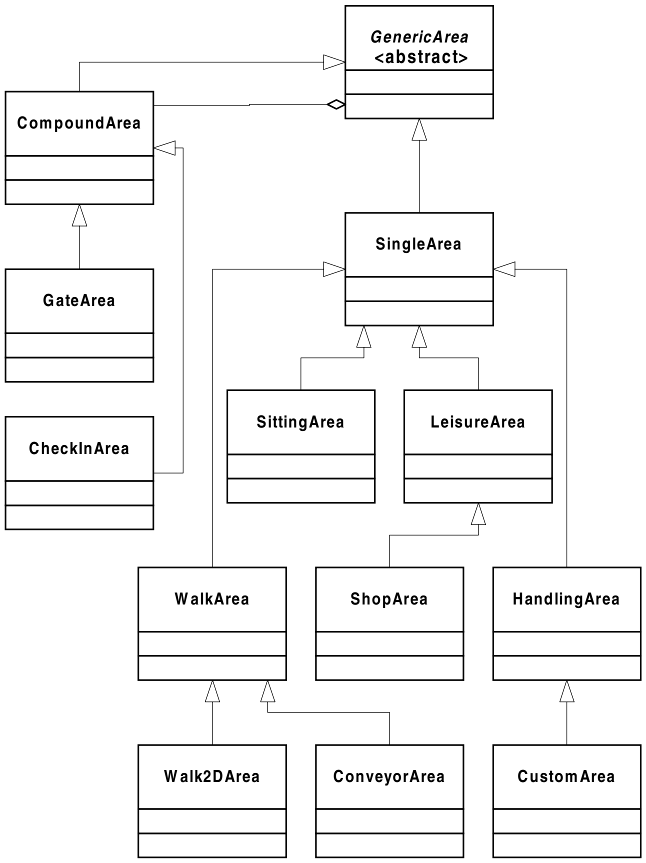

Figure 1: Partial Inheritance Tree of the Area Building Blocks

Each area building block is responsible for gathering statistics about numbers of passengers, waiting times, and capacity use. Furthermore, each building block has a user interface for entering data for its characteristics. Areas might also have methods for showing animation, for setting the parameters for resource behavior, and for determining the time it takes to let a passenger through. To standardize these functionalities, each model building block is built out of so-called *building block elements.* Figure 2 shows the building block elements of a single area. Figure 3 shows that in case of compound areas, the number of building block elements can grow considerably.

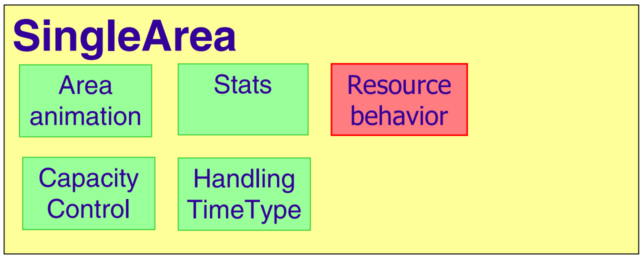

Figure 2: Building Block Element Single Area

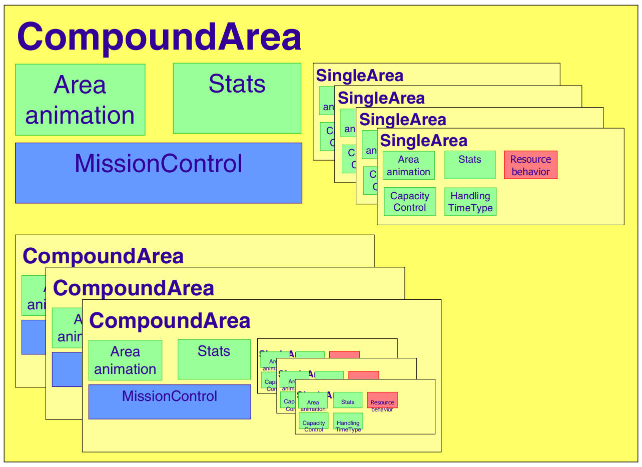

Figure 3: Complex Compound Area Consisting of 4 Single Areas, Mission Control, and 3 Other Compound Areas

Figure 3 shows that a more complex compound area might need control blocks – the mission control – to determine the overall behavior. Here the mission control is shown as one building block element, in reality, there are several building block elements available to model the control functions of the compound area. The statistics building block element is available at all levels, enabling the analyst to drill-down from a high level analysis of area use of an overall terminal or concourse to a detailed analysis of individual areas that cause a problem.

3.2 Group Building Blocks

The group building block is quite simple. It consists of a number of attributes such as group size, walking speed, flight number, and script. Of course a group also calculates a number of statistical parameters. Separate attributes exist for summing the queuing times, waiting times, leisure times, handling times, etc. The only activity of a group is to carry out its script and to update its statistics based on the activity in the area and the waiting time for the next area. A group can have an animation representative that displays itself on the screen. In our case, the icons of the animated groups consist of a number of dots that indicate the group size.

3.3 Group Process Building Blocks

One of the most interesting features of the implemented airport terminal building block library is the fact that each group of passengers carries out its own script that has been given to the group by the group generator control block. A special language has been made for the script, and each group has a script interpreter that carries out the script lines one by one. Each script line consists of two parts: a method name and one or more parameters. The method name is the string representation of the script methods that each group possesses. There can be zero or more parameters that the method expects when carrying out the method. Some methods lead to changes in the destination, impacting the next area the person will go to. Other script statements such as `DetermineRestTime` make a calculation that can later be used. Finally, there are statements such as `WaitForBoard` that wait for a – usually flight related – event to take place.

Table 1: Example of a Group Script

method	**parameter(s)**
SetDestType	Checkin
DetermineRestTime	-1
SpendRestTime	DelayResttime
SetDestType	CheckinArea
DetermineRestTime	-1
DelayFor	min(CurrentGroup.RestTime, Uniform(1,300,600))
SetDestType	Customs
DetermineRestTime	-1
SpendRestTime	DelayResttime
SetDestType	GateArea
WaitForBoard	
SetDestType	Gate
WaitForFlight	
LeaveSystem	

Below, a number of examples of methods that can be used in a script are given:

- `SetDest("CustomsWest")` – the current destination will be changed to an area called `CustomsWest`.
- `SetDestType("Gate")` – the SetDestType statement looks up the instance of a certain category in a corresponding control block and sets the resolved area as destination. For "Checkin" the area

will be the check-in row of the group's flight. "GateArea" is a free space in the neighborhood of the gate. For "Gate" the destination will be the gate of the flight. For "Customs" the destination will be the nearest instance of a customs area.

- `WaitForBoard` – waits for a trigger that the boarding process and final safety check will start. The passengers will try to move to the secure area near the gate.
- `WaitForFlight` – waits for a trigger to occur that the flight will depart. The passengers will try to move to the bridge or bus that connects the gate area to the plane.
- `DetermineRestTime(-1)` – calculate the time that is left before the flight departs, and store it in a group attribute called `CurrentGroup.RestTime`. The "-1" parameter determines the way the rest time will be calculated (e.g. with or without taking the walking time to the gate into account).
- `SpendRestTime("TaxFree")` – look up the nearest area of type "TaxFree" in the corresponding control block, and spend the calculated rest time in one of the tax-free shops.

The scripts can be made as complex as the modeler wants. If necessary, if-then-else or while control statements are available to carry out parts of the script conditionally or repetitive. Because each statement in the script language points to a method to be carried out, the script is easily extendible. When the modeler needs a new script statement, only the corresponding method needs to be added to the generic script interpreter of the group class, and each group can from that moment on use this script statement to carry out the new task. In the script interpreter, easy access is given to a number of generic control building blocks and to the group's own attributes.

3.4 Control Building Blocks

The control building blocks carry out tasks that are outside the observation of the passengers, but that influence their behavior.

The most important control building block is the *flight information*. The flight information building block can read a detailed flight table from a file or database, and make it available for the other control building blocks. Several methods in the flight information building block ease the access to the flight table. Functionality offered by the flight information building block is for instance making available information for the opening and closing of check-in desks, scheduling the actual arrival or departure of flights, and triggering the boarding process based on the real departure time of the flight. Rosenberger et al. (2000) show that the real schedule of departing and arriving planes has a lot of randomness. Disruptions can cause large differences between the ETA and ATA – Expected and Actual Time of Arrival – and the ETD and ATD – Expected and Actual Time of Departure. Therefore, we decided to make an explicit distinction in the flight information building block between the expected, scheduled times and the actual times. Departing passengers, for instance, base their time of arrival in the airport terminal on the ETD, and therefore spend a lot of extra time, and occupy more space than planned, when a delay occurs. It is for these types of delays that the airport terminal capacity needs to be prepared.

A control block that is used by the groups is the *shortest path control block*. Actually, this building block is a simple replacement for group objects to use when they are moving from their current position to a destination. Rather than modeling the signs at the airport in detail, we chose to simplify the models in this respect. Because the areas are connected, the set of areas forms a graph for which the shortest path algorithm can be run once, at the start of the simulation. Two tables are filled that are optimized for speed when a group asks for the next area it should go to for reaching its destination. After some experimentation, it turned out that rather than the lengths of each area, the *resistance* of the area is a better indicator for the attractiveness for the passengers. In our first implementation for Schiphol Airport, passengers went through customs twice – first leaving the international terminal, and afterwards entering the international area at another location – when the distance was shortest. When modeling a customs area with a high resistance, passengers suddenly tried to avoid the customs desks, except when their scripts told them to explicitly go through. Similar actions can be taken for having passengers use escalators rather than stairs, or avoiding infrequently scheduled elevators – which move very efficiently from one floor to another – again, except when the script of a group with at least one disabled person forces the group to use an elevator.

There are also control blocks for *check-in desk assignment*, *baggage belt assignment*, *gate assignment*, and several other flight specific resource allocations. The passenger groups can ask through the `SetDestType` script statements to go to the right check-in desk or transfer desk, baggage belt, or gate. Allocation can be detailed per flight, or more rough per airline. Because the interfaces for the building blocks are standardized, many implementations of these building blocks can be made with different algorithms, or even connecting to databases or systems.

A final control building block is the *statistics control block*, which is responsible for calculating statistics at regular time intervals. The statistics building block can, for instance, ask all areas every hour to report and reset their statistics gathered during the past hour. The building block elements (see figures 2 and 3) help tremendously, because the interfaces of the statistics gathering functions are standardized in each area class. Passengers report all gathered statistics to the statistics control building block when they leave the system. Another item they report to the statistics is whether they caught their flight or not. Especially for transfer passengers, and on airports with large distances, the number of passengers that miss their flight is an important indicator for the quality of the processes.

4 IMPLEMENTATION EXAMPLE

The library building blocks have been implemented in the object oriented simulation language eM-Plant of Tecnomatix (2001). Several tests for different airports have been carried out with the library. In this paper, an example is shown of a full terminal implementation for Amsterdam Airport Schiphol. Figure 5 shows the high-level model of the implementation of the entire airport terminal. Figure 6 shows the generated animation of the F-pier of Amsterdam Airport Schiphol, where passenger groups of different sizes move through the pier. The animation is shown using the animation building block elements in the CompoundArea model building blocks. The building of the initial model could be carried out very fast, the first good working version was ready within two weeks. Later, the model has been extended for new use, for instance for testing the evacuation procedures at the airport. During evacuation, each passenger group is given a new script by a special evacuation control block with one statement: `SetDestType("Exit")`.

Usually, it is quite time consuming to create the user interface for a large number of building blocks. In our case, this turned out to be much simpler, due to the choice for the building block elements (figures 2 and 3). Each building block element was given its *own* user interface that needed to be implemented only once per building block element. The only thing the building block developer needs to do for the user interface of the building block is providing input fields for the specific parameters. Buttons for each building block element which contain the element's own user interface when clicked, are automatically added. Using this functionality, we can ensure that the user interface for similar parts in the tool is always the same. It also reduces the amount of work for developing new building blocks considerably.

The output of the model is of course also very important, because it is the major information on which the decision makers base their decisions. In order to allow for a use of the building blocks in many different decision situations, many different types of statistics are gathered in the model. To avoid an overload of generated information, the user can decide to gather all statistics, or only a subset per building block. It is the responsibility of the *statistics control building block* to turn the gathered data – that is gathered by the statistics building block *elements* in the other building blocks – into information during the run, or into files that can be analyzed later. Again, several different instances of the statistics control building block can exist, of which one can be chosen and inserted into the model. Replacing the way statistics are written away has thereby be reduced to an action of seconds instead of days. We have tested an implementation where all information gathered was immediately written to external files. This produced so many megabytes of information in hundreds of files, that it was unworkable. For several examples, we have therefore chosen to use a statistics block that writes away hourly information to a number of files, which can be studied using e.g. Excel. This worked very well. An example is shown in figure 4, where the number of passengers in the central check-in area is shown during 24 hours of operation on a busy day. A graph like this can be generated in a matter of minutes on the basis of the information on file.

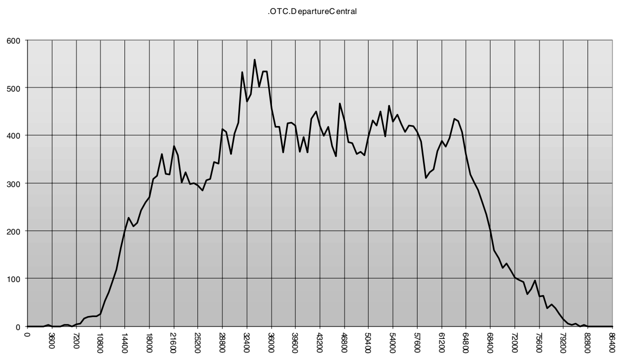

Figure 4: Example of Gathered Statistical Output

5 CONCLUSIONS

The choices we made for the implementation of the airport library resulted in very flexible and powerful models and a well maintainable library. One of the main reasons is the set of well-defined model building blocks with clear responsibilities. The standardization of interfaces helps to quickly replace one implementation of a building block by another version. The separation between infrastructure – static – and passenger behavior – dynamic – made the modeling of complex airports easy. Both the scripts and compound infrastructure areas are now reusable, and can be joined in a model in any combination. The separation between the infrastructure and control building blocks also contributed to the reusability. We expect that faster modeling and easier maintenance will also occur during day-to-day use of these concepts. This has, however, still to be tested in practice.

REFERENCES

Babeliowsky, M. Designing Interorganizational Logistic Networks. Ph.D. Thesis, Delft University of Technology, 1997.

Bitauld, P., K. Burch, S. El-Taji, E. Fanucchi, M. Montevecchi, J. Ohlsson, A. Palella, R. Rushmeier and J.

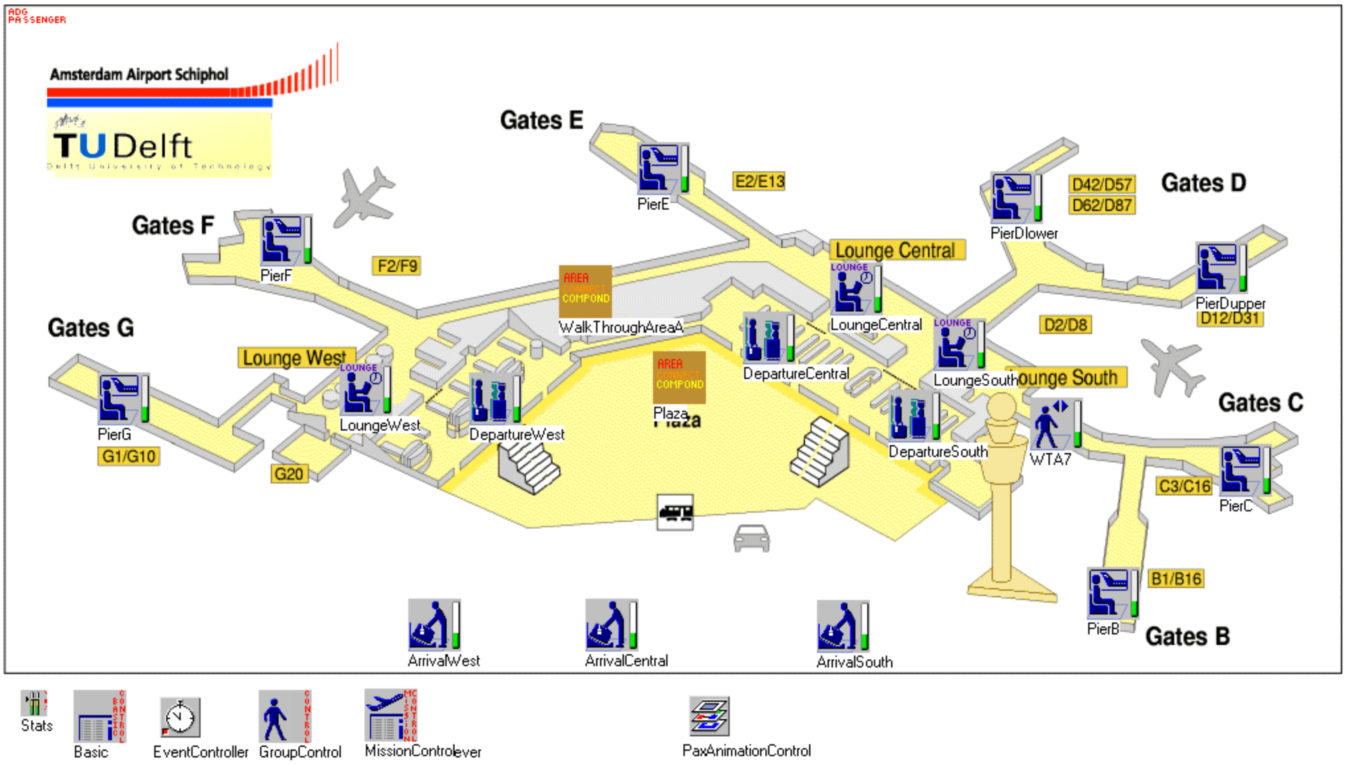

Figure 5: High-Level Building Blocks in the Schiphol Model

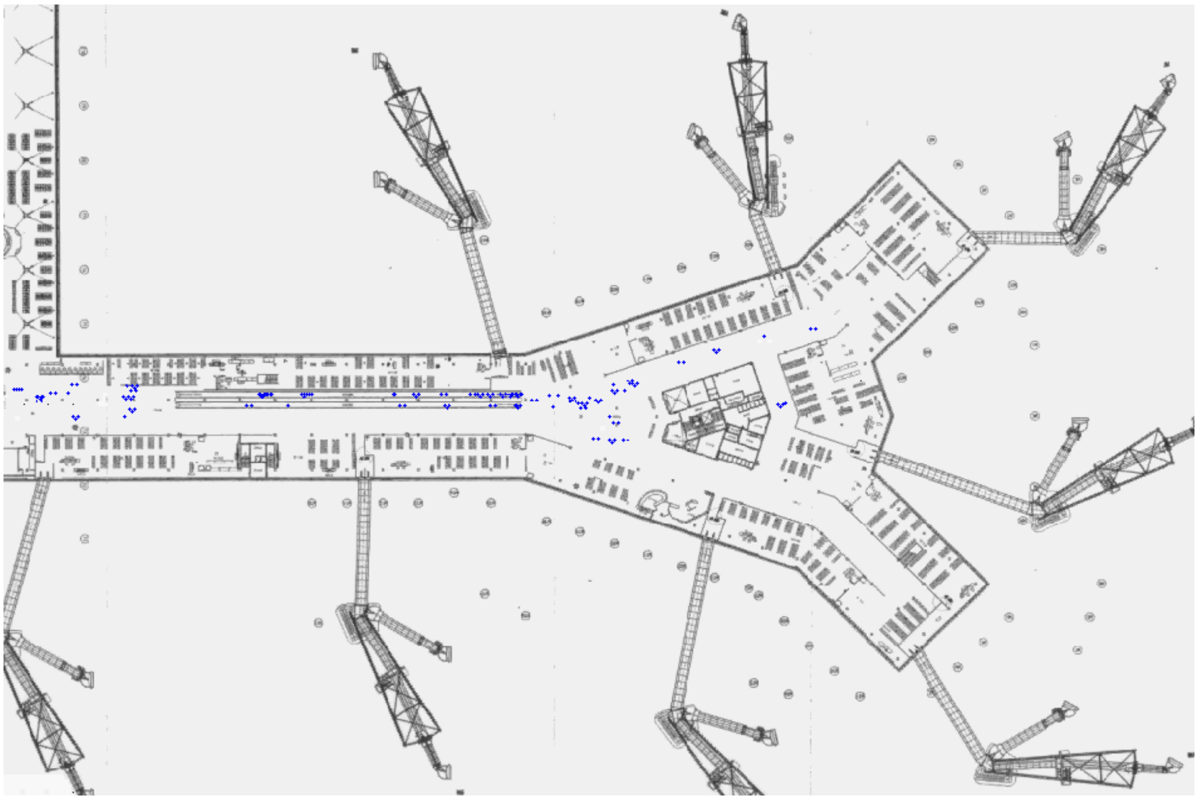

Figure 6: Animation of the F-Pier in the Model

Snowdon. "Journey Management". In: OR/MS Today, October 1997, pp. 30-33.

Gatersleben, M.R. and S. van der Weij. "Analysis and Simulation of Passenger Flows in an Airport Terminal". In: P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds., Proceedings of the 1999 Winter Simulation Conference. Piscataway, NJ: IEEE, 1999, pp. 1226-1231.

Joines, J.A. and S.D. Roberts. "Object-Oriented Simulation". In: J. Banks (ed.), Handbook of Simulation. New York: Wiley, 1998.

Joustra, P.E. and N.M. Van Dijk. Simulation of Check-In at Airports. In: B.A. Peters, J.S. Smith, D.J. Medeiros and M.W. Rohrer, eds., Proceedings of the 2001 Winter Simulation Conference. Piscataway, NJ: IEEE, 2001, pp. 1023-1026.

Rosenberger, J.M., A.J. Schaefer, D. Goldsman, E.L. Johnson, A.J. Kleywegt and G.L. Nemhauser. "SIMAIR: A Stochastic Model of Airline Operations". In: J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, eds., Proceedings of the 2000 Winter Simulation Conference. Piscataway, NJ: IEEE, 2000, pp. 1118-1122.

Snowdon, J., S. El-Taji, M. Montevecchi, E. MacNair, C.A. Callery, and S. Miller, "Avoiding the Blues for Airline Travelers". D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan (eds.), Proceedings of the 1998 Winter Simulation Conference, Piscataway, NJ: IEEE, 1998, pp. 1105-1112.

Tecnomatix. User Manual eM-Plant 4.6. Stuttgart, Germany, 2001.

Valentin, E. and A. Verbraeck. "Simulation Using Building Blocks". In: F. Barros and N. Giambiasi. Proceedings AIS'2002. San Diego: SCS, 2002, pp. 65-70.

AUTHOR BIOGRAPHIES

ALEXANDER VERBRAECK is an associate professor in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology, and a part-time full professor in supply chain management at the R.H. Smith School of Business of the University of Maryland. He is a specialist in discrete event simulation for real-time control of complex transportation systems and for modeling business systems. His current research focus is on development of generic libraries of object oriented simulation building blocks in C++ and Java. Contact information: `<a.verbraeck@tbm.tudelft.nl>` `www.tbm.tudelft.nl/webstaf/alexandv`

EDWIN VALENTIN is a researcher in the Systems Engineering Group of the Faculty of Technology, Policy and Management of Delft University of Technology. His specialty is the development of domain-dependent generic discrete-event simulation libraries. Edwin participates in the BETADE research program on developing new concepts for designing and using building blocks in software engineering, simulation, and organizational modeling. Contact information: `<edwinv@tbm.tudelft.nl>` `www.tbm.tudelft.nl/webstaf/edwinv`