Interactive Airport Landside Simulation: An Object-Oriented Approach

S. A. MUMAYIZ AND R. K. JAIN

Airport landside simulations have been developed for more than two decades by the analytical-mathematical modeling approach in which conventional general purpose languages (e.g., FORTRAN BASIC) or simulation systems [e.g., GPSS (general purpose simulation system)] are used. Consensus within technical circles currently recommends the use of object-oriented programming simulation as an appropriate approach to designing the next generation of airport simulation models, along with the use of techniques developed in artificial intelligence and computer-aided design. The objective of this paper is to explore the different aspects in the design and implementation of an airport landside simulation model. Major features and properties of objects in the object-oriented programming environment are discussed and related to the simulation environment with an attempt to find analogs between the two to represent the airport landside environment in an object-oriented environment. The preliminary effort using an object-oriented approach to design and implement a simulation environment is presented and discussed.

There has been considerable discussion recently about implementing object-oriented programming for airport landside simulation. Object-oriented programming has been around for a while and its main applications have been in artificial intelligence and expert systems. However, modern object-oriented extensions of popular programming languages have not yet gained a foothold in mainstream scientific programming. The intent of this paper is to analyze the airport landside simulation system from a conceptual view compared with the conventional mathematical modeling approach, and explore various aspects of simulating this complex system through the adoption of an object-oriented approach. A brief description of the major features of object-oriented language elements is followed by a presentation of preliminary work to simulate airport landside using object-oriented programming. Drawbacks of the conventional approach are also discussed and compared with the present programming approach.

CURRENT SIMULATION APPROACH: DESCRIPTION AND LIMITATIONS

The airport terminal can be modeled on two scales: macroscopic, as one integral system from landside to airside, or microscopic, in which individual facilities are considered as separate entities. Each approach has its own merits and shortcomings. A brief description of two simulation approaches taken to design an airport landside simulation model is presented. They are the Federal Aviation Administration's (FAA's) Airport Landside Simulation Model (ALSIM), and the Canadian Airport Planning Model. A more detailed description of airport landside simulation models can be found elsewhere (*1*).

S. A. Mumayiz, TAMS Consultants Inc., 2101 Wilson Boulevard, Suite 300, Arlington, Virginia 22201. R. K. Jain, TAMS Consultants Inc., The TAMS Building, 655 Third Avenue, New York, N.Y. 10017.

FAA Airport Landside Simulation Model (ALSIM)

ALSIM is a macroscopic, discrete-event, fast-time computer simulation model capable of producing flow and congestion parameters and respective statistics at simulation facilities. The structure of the model consists of main and auxiliary programs. The ALSIM simulation model is programmed on the IBM/370. The operating system on IBM/370 is essentially a batch processing type that requires the program, data, and appropriate system commands to be submitted together as a single job. There is no user interaction between the users and the executing programs (Figure 1*a*). ALSIM exhibited a fast execution time of approximately 7 min because of batch processing and the absence of run-time interaction between the user and the simulation environment. Debugging is a difficult task—an inherent drawback of a batch-oriented operating system. Several different languages are used to design ALSIM. The operating system commands are executed with the use of job control language (JCL), the main simulation environment is designed in GPSS-V, all input-output, data and report formatting are conducted in FORTRAN. This is tied together as a single module by the IBM/370 assembly language. In a single run, ALSIM can compute and analyze a number of parameters, statistics, and transactions. The model creates transactions representing the processing of passengers and visitors. Data sets, which are grouped into four major categories: flight schedule, passenger characteristics, airport geometry, and facility information, are supplied to the simulation environment before the execution of the simulation process. Reporting includes passenger statistics, processing facility statistics, occupancy rate, and flow characteristics of passengers through the system (*1–3*).

Canadian Airport Planning Models

Contrary to ALSIM, the model developed by Transport Canada represents an independent module-based simulation model to assist in the planning, design, demand assessment, and capacity-service evaluation levels of airport terminals and ground access facilities. It consists of a number of interactive, interrelated, and mutually compatible separate models that run on an IBM-AT or compatible microcomputer. Each one

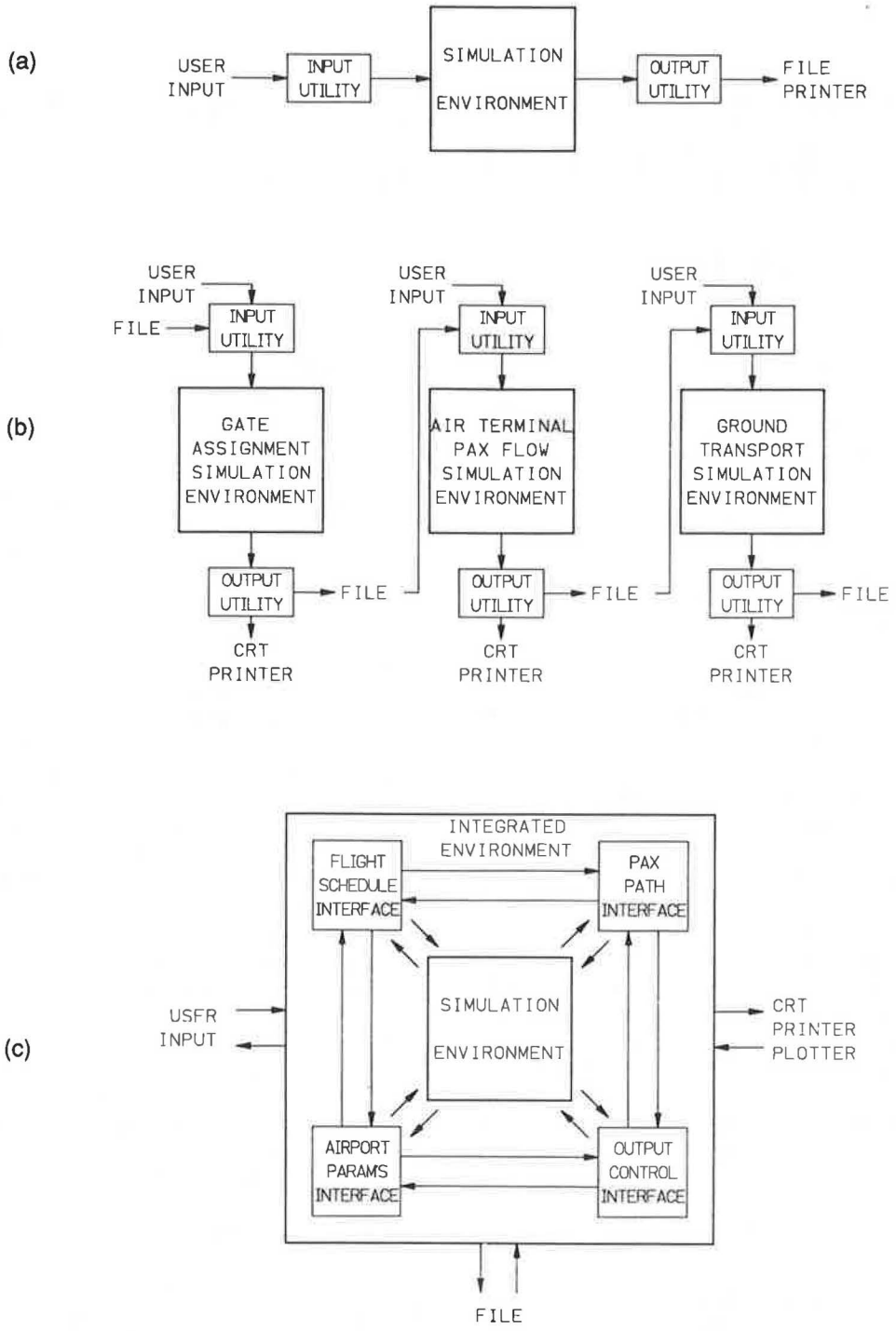

FIGURE 1 (a) ALSIM simulation environment, (b) Transport Canada's simulation environment, (c) interactive simulation environment.

of these individual models nas front, middle, and end segments. Front and end segments, programmed in BASIC and Pascal, serve as input data editor and report generator. The middle segment is programmed in FORTRAN IV, and represents the simulation engine of the model. These three models are the gate assignment model, the air terminal passenger flow simulation model, and the ground transportation simulation model (Figure 1*b*).

The gate assignment model is a multi-channel queueing type operating in time intervals of 5 min. The output generated includes listing of two-way passenger flow through each gate, a Gantt Chart, statistics of gate use, and aircraft-passenger delays.

The air terminal passenger flow model is an interactive event-oriented simulation model that simulates flow of passengers along predefined paths from curb to the airport gates

and vice versa. The output includes number of users entered and exited, baggage device assignment, queueing, and delay.

The ground transportation model is an interactive model that simulates the flow of vehicular traffic in the roadway system of the airport. The input is derived from survey statistics and flow path information. Its output includes statistical information on occupancy time interval; vehicular characteristics (private, taxi, rental, mass transit); and total number of vehicles entering and exiting in a given time interval (*1,4*).

THE CURRENT APPROACH TO LANDSIDE SIMULATION

The difficulty in representing the landside of an airport is increasing with the growing complexity of the modern methods of passenger handling at airports. A number of proposals have been put forward in recent years for the development of a general airport landside simulation model.

The complexities of the airport landside system are exhibited by the variations in the handling processes that a passenger has to undertake at an airport under different situations. Combined with the human element, these processes become stochastic in nature. The technology and organizational context of each processing unit differ from one situation to another.

A clearly defined simulation process for the landside is needed. The conceptual task faced is to understand and represent the airport landside so that a simulation model can be generated with the capability of effectively helping in the design of airport service areas.

From a theoretical standpoint, a simulation model should ensure the following:

1. Suitability for a wide range of situations,
2. Effectiveness for the process of airport service design,
3. Ability to evaluate present states and also support evaluation and selection through judgment and predictions,
4. Easy modification to accept newer technological and organizational approaches, and
5. Effective analysis of threshold and performance measurement standards.

AIRPORT LANDSIDE AND COMPUTATIONAL MODELING

Advanced programs that support numeric calculations for structures, highways, energy, finance, defense, and other quantitative aspects have grown steadily in the past two decades. These tools have helped in the development of solutions and alternatives in their specific fields, particularly in cases in which there is standardization. These tools have successfully augmented and substituted the traditional manual operation but have actually provided very little support to the representative, generative, and evaluative tasks of a simulation process, states, and interaction.

The computational techniques and tools developed to date for other disciplines do not support the requirements of airport landside computer simulation models. They suffer from limitations that are inherent to the concept of the word "program" itself. In this context, a program deals with the execution of a sequence of a predetermined set of instructions to obtain an output defined by the formulation of the problem and the data provided (*5*). This "programming" approach works well for applications and situations in which the problem can be perfectly formulated, but fails when applied to complex and less formulated systems like landside simulation. The landside design problems are neither well understood nor can they be strictly formulated. Instead, they tend to be formulated in reaction and in response to parameters that emerge gradually during run time. These parameters vary according to case and vary even within the same case at different times. This clearly reflects a complex system made up of a large number of parts that interact in an uncomplicated way in which the whole is more than the sum of the parts. Given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer, evaluate, or generate the properties of the whole. In the face of such complexities, the principles of reduction and modulation of the parts along with their properties and laws can be a pragmatic method (*6*).

The failure of a systematic method has nevertheless produced a series of computational operations that make up the present simulation models and utilities. Thus the designers of landside simulation software have to use precise, systematic, and easily manageable models, following the logic of methodology. This situation can be overcome by the hierarchic system. It is based on techniques developed in the domain of artificial intelligence for representation, inheritance, demons, and perspective. Various procedures can be constructed that can make assumptions, report what is relevant, and also look for information on the basis of hierarchy (*7*).

HIERARCHIC SYSTEMS AND OBJECT-ORIENTED PROGRAMMING

A hierarchic system or hierarchy is composed of interrelated subsystems, each of the latter being in turn hierarchic in structure until some elementary subsystem is reached. Frames used as a means of representation in artificial intelligence can best represent hierarchy. Apart from a simple class or subclass relationship, a frame can also represent complex subordination among subsystems and specialized concepts. These frames can report common situations (prototype), make assumptions, assign default values, decide what to look for depending on the situation, and also where to look for it (*7*). This concept of frames can be easily codified into binary representation with the use of object-oriented programming languages.

Object-oriented languages are not new in concept, first appearing in the 1960s. The first object-oriented language, Simula-67, was a tool designed to model the operation of mechanisms or objects. Previously, object-oriented languages were largely restricted to specialized or experimental applications such as artificial intelligence and expert systems appearing in the form of an interpreted language such as Lisp, Smalltalk, Flavors, or Loops. The development of computer languages has kept pace with the realities of various applications. The increasing technological breakthrough in computer hardware and software has made conceptual object-oriented programming a reality. The emphasis on structured programming imposed order on a situation that had been

approaching chaos. Data abstraction and hiding of data has given object-oriented programming a new level of modularity. A conventional programming language essentially stores, recovers, and manipulates data mathematically, and treats data and procedures and functions separately. In object-oriented programming, not only are the data and procedures viewed as one entity, they can be viewed conceptually. The representation of "concept" is a unique characteristic of object-oriented programming. With object-oriented techniques hierarchy, relationship and abstract concepts can be conveniently and neatly represented with compact and reduced code. The designer does not have to worry about data types, procedures, and variables and can just deal with the representational and conceptual issues (*8–10*).

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming can be done either by an object-oriented language such as Lisp, Prolog, Smalltalk, or object-based programming languages like Object-Pascal and C++, which are supersets of programming languages Pascal and C, respectively. These programming language extensions define object as a data structure. An object type of data structure is similar to a record type in Pascal or a struct type in C, with the provision to include fields that are procedures or functions called methods of an object within itself; they can also have an established ancestor-descendent hierarchy. The principal properties of an object-oriented language are inheritance, encapsulation, and polymorphism.

- *Inheritance* is a property that allows creation of a hierarchy of objects and descendants of objects inheriting the characteristics and properties of the ancestor. The relationships between ancestors and descendants can provide predictive information and represent knowledge. Inheritance facilitates code sharing, reuse, and extension of the current object. The data fields are inherited by the descendent objects from the ancestors. The descendent objects can also have data fields of their own. Descendants can inherit functions of their ancestors and can have additional functions of their own.
- *Encapsulation* allows the orderly, structural arrangement of the internal data of an object and the associated action on the manipulation of these data. It combines records or structs with functions. The data elements of the object are accessible through the functions connected with the object. In addition to these benefits, data encapsulation also enables data hiding—access to the data of an object is restricted to the methods associated with the respective object. The data-hiding feature makes debugging easier than conventional programming.
- *Polymorphism* means "many shapes." In object-oriented programming this concept refers to sharing action (and action names) throughout an object hierarchy. Each object in the hierarchy implements the action in a way appropriate to its specific requirements (*8*). The same function name can be assigned to objects in the same hierarchy. Though the functions carry the same name, they can be coded differently depending on their individual requirements.

A good interactive user interface has a major impact on the operation of the software. A good graphical user interface could be easily designed through the modular feature of object-oriented programming. The design, construction, and debugging of the program are relatively straightforward and simple. The programs can be written for a wide range of hardware platforms (computer systems) without major modifications. A change in the platform would require only the replacement of hardware-dependent modules. It would also be easy to upgrade or modify the procedures without changing the major source code.

Because concepts can be easily represented, a simulation model of a generic airport can be formulated. This generic model can be tailored to any specific situation through a user interface without having to change the source code. The simulation model can also be expanded to evaluate and make judgments on a rule-based or predictive expert system.

THE OBJECT: BASIC ENTITY OF THE SIMULATION PROCESS

The object is the basic unit of the simulation system. It holds the data base and its operators, the inter-object relationship, and operators for prototypical and specific cases. The simulation system consists of three major types of objects:

1. Generic objects that are the prototypes,
2. Specific objects that are tailored generic objects placed in a specific context, and
3. Relational objects that establish inter-object relationships.

The objects include both descriptive and functional characteristics. The descriptive characteristics represent the physical or name-value component, whereas the functional characteristics represent the nonphysical, conceptual characteristics. Descriptive characteristics provide the system with a data base and functional characteristics provide it with knowledge, establish relationships, and represent concepts.

Examples of some of the objects that are used in the prototypical, preliminary simulation model are listed in the following paragraphs.

Event_Object

An object that updates the clock, the event_object looks out for changes of states in the simulation environment and reacts to any changes according to the parameters of the simulation environment. The event_object also checks for user interruption and halts the simulation process for user input; for example, if the system clock time equals the arrival time of the plane less the time the first meeter-greeter arrives at the airport, it initiates the meeter-greeter logic of that specific plane object.

Pax_Object

This is the passenger object, which is the subclass of the plane object. Its main parameters of arrival and departure time are held within the plane object. The pax_object is assigned a route, which it undertakes at arrival and departure. The

pax_object updates its position on its own if it is "linked" to a transition_object; or if the passenger object is linked to a transaction_object, it queues and waits to be processed by the transaction_object. The pax_object keeps track of its position. It also relates itself to other passengers to represent group relationship, moves through the list of process_objects, and keeps statistics of its own time at each processing unit.

Transaction_Object

The transaction_object represents the processing units such as check-in desks, baggage handling, immigration counters, and so on, in which a passenger has to queue and undergo the necessary processing. This object holds the logic of its process and actions, which can be mathematical, statistical, or heuristic representations or models. It is in reality a sub-simulation of the main simulation process, in which each transaction_object simulates the scenario at a check-in desk or security center. It holds the processing variables, statistics of itself, processes passengers, and keeps track of processing time.

Transition_Objects

The transition_object represents the transition the passenger has to undertake to connect from one processing unit to another. The transition_object connects the different transaction_objects into a finite set of possible processing paths that represent the situations at a specific airport. Each arriving, departing, or transfer pax_object is assigned a processing list on creation. Apart from establishing relationships between transaction_objects, it also holds values of distances between the transaction_objects.

Process_Object

This is the superclass of the transition_ and transaction_objects. It defines the logic of traversing within the processing network. The processing_object is an abstract object that contains generic procedures for the use of its descendent objects (e.g., access interface object).

Demon_Object

The simulation system consists of a number of demons who "reside dormant" in the simulation environment. They are evoked when they are needed to accomplish some task. A demon assigns logical or default values to other objects in the simulation process. These values are dependent on a specified range of mathematical, statistical, or heuristic sub-modeling of an individual action represented by the demon. A demon can help in modeling the randomness of a simulation process and also give it a realistic touch. An example of a demon is the stochastic_deplaning_demon, which is evoked when the plane_object is attached to a gate. This demon assigns the deplaning passengers or group of passengers a processing path for a set of predefined processing paths.

DESIGN OF USER INTERFACE AND IMPACT ON SIMULATION PROCESS

User interaction has become an important feature in software design. Computer graphics have become a standard means of communicating between the user and the machine. A good user interface reduces the visible complexities of the model and guides the user through the whole process, making the operation of the model very simple. Interactive graphics can also be used to reflect the state of the simulation environment to the user, thus confirming whether it is a true representation.

Through interactive computer graphics, the user can control and manipulate the simulation process. This can be accomplished with the use of interactive devices such as keyboards, a mouse, a digitizer tablet and puck, or a light pen. With interactive devices, the user can either type in alphanumeric information or select from menus existing on the digitizer or on the CRT screen. Interactive interface is used to specify operations or components of the simulation process on which operations are to be executed. This interactive interface technique does not require the user to be a programmer. The user simply has to make choices, answer questions, and assign place values on the CRT when prompted (*11*).

The interactive graphics capability of the simulation model achieves a strong man-machine communication. A combination of text and static or dynamic graphics or animation makes a significant difference in the user's ability to understand data, perceive trends, and visualize the simulation in totality. The user interaction is designed to

1. Provide a consistent interaction sequence,
2. Limit and control the number of options required for direct communication with the program and exhibit available options in a hierarchical manner,
3. Prompt the user at each stage of the interaction process,
4. Provide appropriate feedback to help the user evaluate the trends of the simulation process, and
5. Make the user interface crash-proof and recover gracefully from mistakes. It should issue warnings by beeping and provide a feedback on the error and possible solutions.

The simulation environment communicates back to the user through a CRT, a dot matrix printer, or through an output file. Additional facilities like capturing the screen and producing a hard copy can be helpful in documenting the process and subprocesses.

REPRESENTATIVE HIERARCHY OF AN AIRPORT TERMINAL SYSTEM

The passenger terminal system can be represented as a hierarchy based on the activities. The major subsystems of a passenger terminal system are the access interface, the process interface, and the flight interface.

- *Access interface* is a situation in which the passengers transfer from the ground transportation access mode to the terminal complex. The representative activities of this subsystem are curb-side loading and unloading of passengers, circulation, and parking. The facilities that constitute this subsystem are the roads, arrival and departure curbs, long- and

short-term parking spaces, walkways, and public ground transport facilities.

- *Processing interface* is the center of the passenger terminal system. Here the passenger is processed at the start or end of the trip. The primary activities that constitute this subsystem are ticketing, baggage check in, baggage claim, seat assignment, federal inspections, and security. This subsystem consists of facilities such as ticket counters and baggage check-in facilities, public and nonpublic concessions, lobbies, circulation spaces (such as corridors, stairways, escalators, and elevators), baggage-handling facilities and devices, security facilities and devices, and federal inspection facilities.
- *Flight interface* is the situation in which the "processed" passenger transfers to the aircraft. The activities conducted in this subsystem are assembly, conveyance to and from the aircraft, and security checks. The facilities provided for these activities are the concourse, departure lounge or gate hold-room passenger boarding devices, airline operations, and space- and security-checking equipment and facilities (*12*). The components of a prototypical passenger terminal system and the hierarchy of the components are shown in Figure 2.

Each node in the representative tree of the prototypical airport can be considered as an object in the same hierarchy as shown in Figure 3. The lines connecting these nodes represent the relationship of ancestor-descendants in the direction from top to bottom. The root of this hierarchical representation is the generic airport object. The first generation objects are the three subsystems, their descendants are the individual facilities, and the end node of this system is the queue_object. The basic components of each object are the variables that hold the data and statistics of the subsystem they parent and the procedures needed to process passengers within a subsystem or a specific node. The processing procedures are more general near the root of the tree and get increasingly specific and complex lower down the tree. This is helpful in establishing default procedures and overall statistics of a subsystem. The property of polymorphism of objects is exploited to achieve the hierarchical complexity of the object-processing procedures that act on the pax_object.

DESCRIPTION OF INTERACTIVE AIRPORT LANDSIDE SIMULATION MODEL

Currently, the model consists of the following five interfaces integrated in a common environment:

1. *Flight Schedule Interface* is a window-based utility used to generate flight schedules interactively. The user can create, retrieve, store, and manipulate originating, terminating, or transit flight schedules interactively. The schedules are maintained in separate files in the binary format and are merged together at run time. Through interactive windows and screen menus, the user can operate on these schedules. Through this interface, the user can also assign passenger characteristics to a specific flight.
2. *Passenger Path Interface* is a menu-driven interactive graphic interface used to represent passenger processing paths required in the simulation environment. The prototypical flow diagrams representing the typical processing facilities and processing paths are selected from the IATA manual (*13*). These

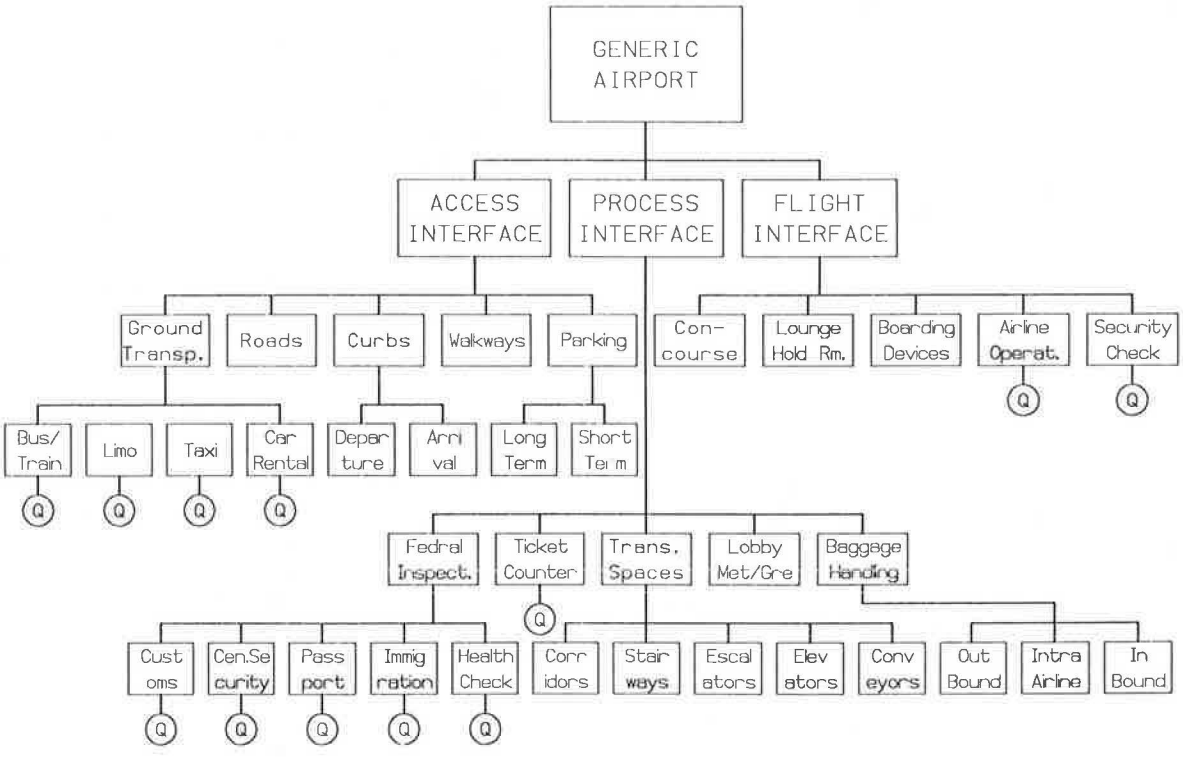

FIGURE 2 Hierarchic arrangement of landside processing facilities.

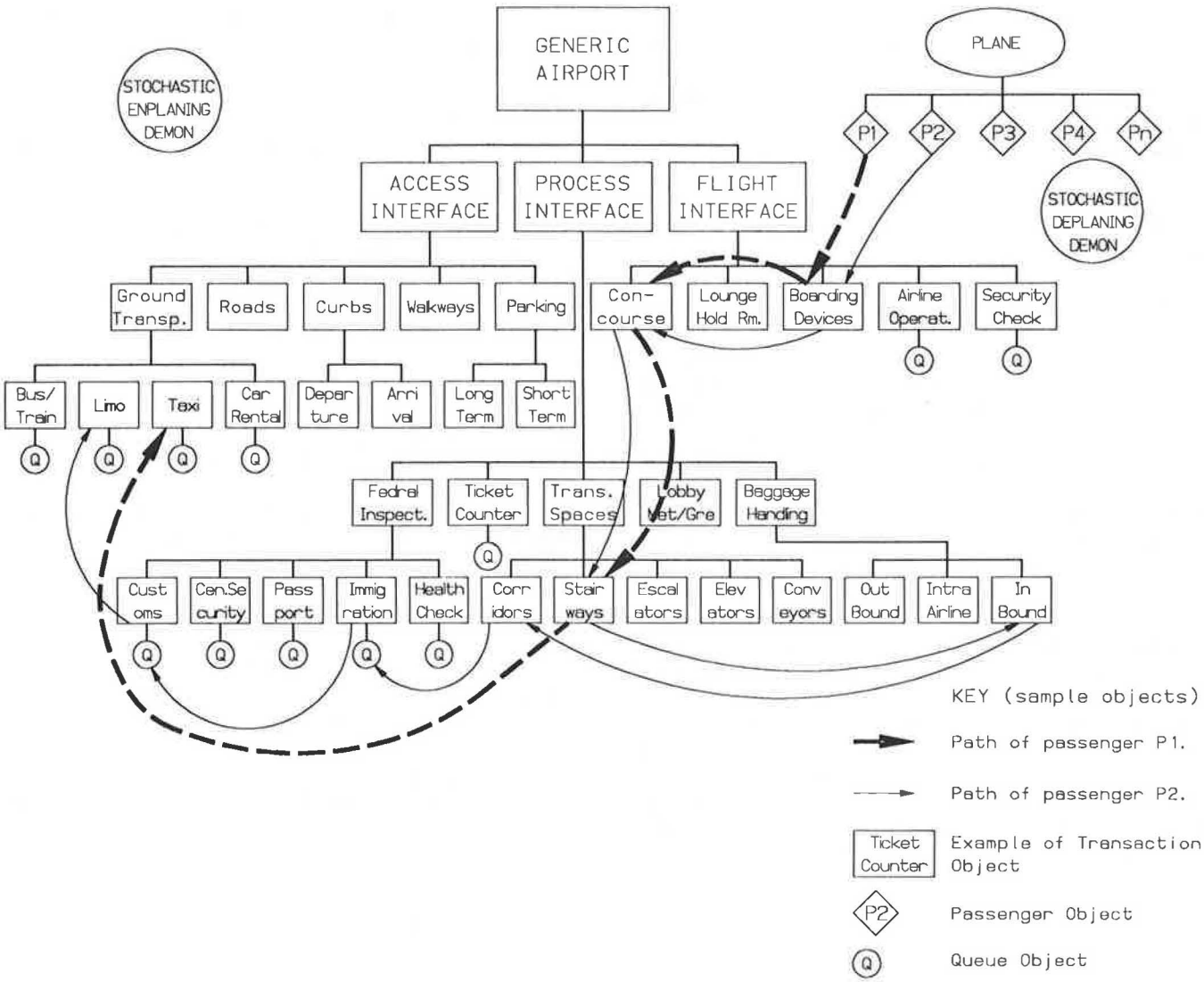

FIGURE 3 Hierarchic arrangement of objects in the simulation environment.

flow diagrams are displayed graphically on the screen. From an interactive mouse-driven menu, the user can choose from four flow diagrams. Further details on the flow charts are available in the IATA manual (*13*) (pp. 3–7 to 3–10). The selected diagram is displayed on the screen and the user has five selection choices from the menu:

- Add a new path to the processing path_list (by interactively linking the displayed processing facilities);
- Assign parameters like traveling distances, mode of travel (corridor, stairway escalator, etc.), or the percentage of passengers on a specific path (either to an existing path or a new path);
- Assign or alter the default parameters of the processing facilities (objects);
- Edit an existing path; or
- Delete an existing path. These paths can be stored in a file and retrieved for future use.

3. *Airport Parameter Interface* is used to assign global parameters to the simulation environment. Default passenger characteristics can be assigned globally to be used by passengers of flights that do not have a specific passenger characteristic assigned to them. Other global parameters like service time distribution, passenger loading and unloading at arrival and departure curb, baggage transport time to baggage claim devices, and so on, can be provided to the simulation environment to override default values. These input requirements are based on the requirements of ALSIM and Transport Canada's simulation model. Further details can be found in the references (*2–4*).

4. *Simulation Environment* is a graphics-based, user-interactive, menu-driven, mouse-assisted interface of the whole simulation process. It is a probabilistic, multiple event-based simulation environment that operates on the parameters and inputs determined by the other systems. The model is designed to run in three modes: representative, evaluative, and optimization.

- In the *representative* mode, graphical representation of the airport landside scenario is generated and displayed on

the CRT. Representative assembly of the different components of the simulation environment and the interaction among them are shown in Figure 1*c*. In this mode, no evaluation is made and the simulation environment graphically reflects the execution of the process. The representative mode is the basic simulation process; it is a "state" generative-representative environment that forms the deterministic parameters for the other two modes.

- The *evaluative* mode is the working of the simulation process in the dynamic threshold evaluator shell. This is shown diagrammatically in Figure 4. At run time the threshold evaluator will check for the performance levels of the processing facilities and periodically indicate the performance of these facilities through an output channel (screen or printer). At the end of the run, the statistics of individual facilities are generated. In the evaluative mode the simulation system can (a) generate transactions of the different processes, (b) keep track of the passengers in the system and the time spent at individual facilities, (c) evaluate operational performances at the individual processing facilities of the whole subsystem or the whole airport, and (d) represent queues and compute waiting times for individual passengers or give an overall perspective on the level of service.
- In the *optimization* mode (Figure 5), the dynamic facility optimization is an integral part and works with the threshold evaluator. The threshold evaluator triggers the dynamic facility optimization manager if a processing facility is overburdened or underused. The facility optimization manager either increases or decreases capacity at run time. A statistical result helps to establish capacities for a new design or evaluates capacities of an existing situation. In this mode, the simulation system, apart from what it can do in the evaluative mode, can also (a) recognize points of bottlenecks or underuse, and (b) increase or decrease the number of facilities dependent on the demand and need based on the level of service established for that run. Actions are indicated graphically. The simulation process can be interrupted at any moment during its run to alter parameters or request status reports through the output subsystem.

5. *Output Control Interface*, as the name implies, is a utility to generate outputs of the simulation environment. These outputs can be directed to the CRT or filed on the disk or the printer. The output can be obtained at run time, user interrupts, and at the end of the simulation run. It can also generate outputs of the schedule files and passenger flow paths.

These systems are interactive and can be selected at any time during the simulation process (Figure 1*c*). At present, the simulation is based on process relationship and is not related to the geometry of the airport. There is a provision in the design of the data structure to incorporate the geometric relationship of an airport terminal floor plan.

The sequence of logic of the simulation process is as follows:

- At the start of each cycle the event_handler_object checks the status, parameters, and flags to establish existence of events such as arrival of planes or user interrupt.
- If a plane is attached to the gate, it evokes the demon_object to deplane the pax_objects from the plane_object and puts the pax_objects into the terminal network.
- Each generated pax_object is assigned a processing path by the stochastic_deplaning_demon. The pax_object travels through the processing path until it reaches the end of the access interface.

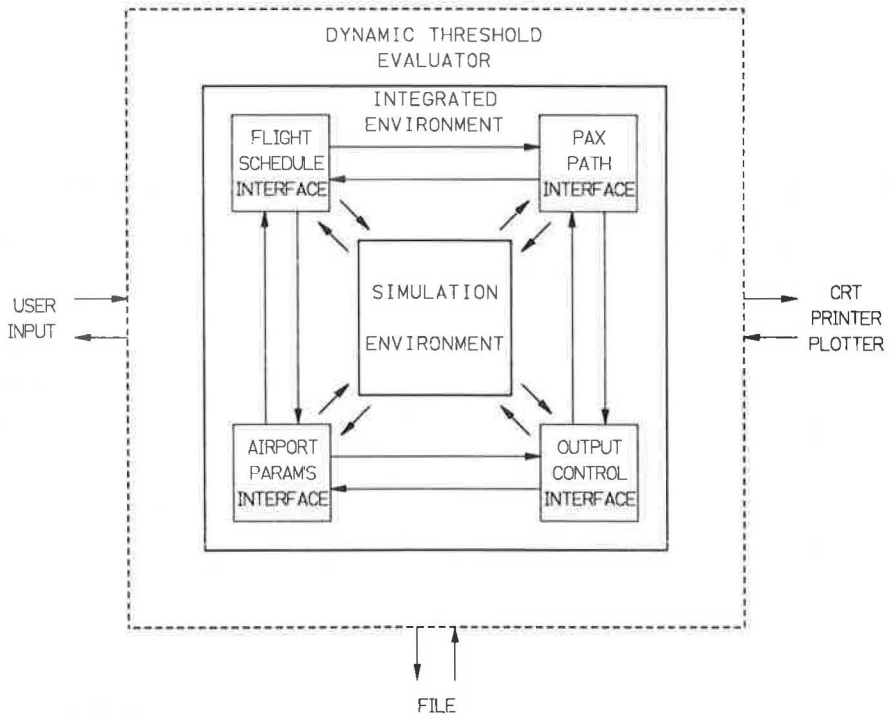

FIGURE 4 Evaluative mode environment.

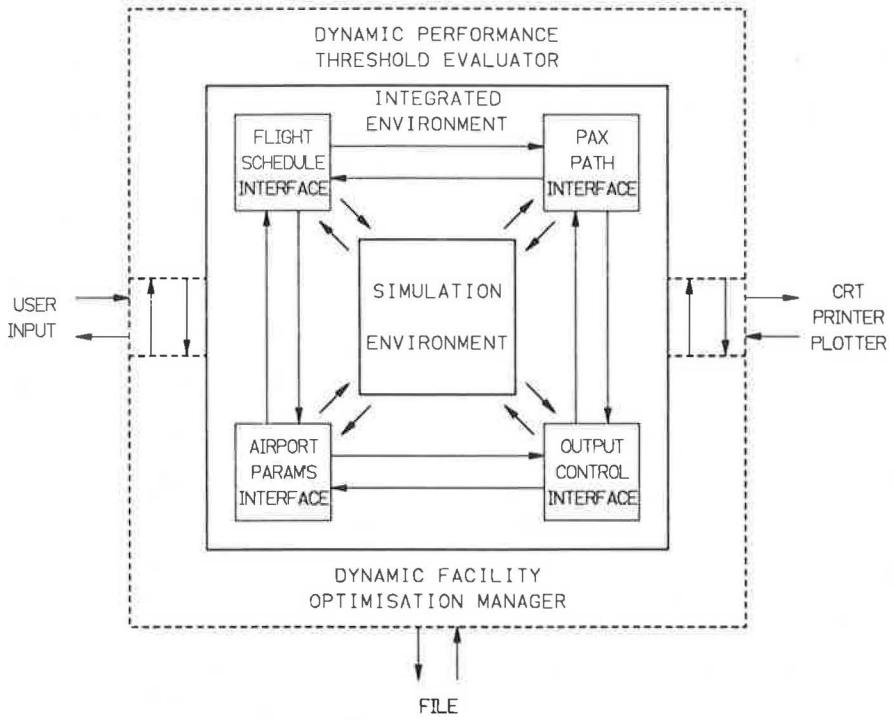

FIGURE 5 Optimization mode environment.

- In the end, destructor_demons update the necessary statistics and destroy the pax_object to make room in the limited memory resource.
- The evaluation of performance of a state is achieved by computing the parameters at that instance; for example, if the queue length at time t at facility f was needed, then the physical count of the pax_objects at that facility f had to be taken.

SUMMARY AND CONCLUSIONS

Presented in this paper is a new approach to airport landside simulation using the concepts from artificial intelligence and computer-aided design. Comparison of the present approach with the conventional approach indicates many drawbacks that inhibited efficient use of earlier simulation systems. Those systems had high "number crunching" capabilities, but exhibited limited and ineffective use of logic-based computation and great difficulty in programming and debugging. In particular, the object-oriented approach is capable of providing immense programming flexibility, greater reduction of input requirements, is easier to operate and user friendly, and offers fully interactive execution with a high degree of animation-graphics capability. Preliminary findings of the performance of the object-oriented landside simulation model indicated the following:

1. The ability to represent concepts in program code and other properties of object-oriented programming greatly enhances the flexibility, versatility, and use of landside simulation.

2. A prototypical (generic) simulation model can be represented and extended for a wide variation of airport types, sizes, and complexity without the need to change the source code.

3. Animation, graphics, run-time user interface, and interaction can increase the capabilities of the model and offer wide possibilities for manipulation and checks that can make landside simulation closer to reality.

As an ending note, the question remains: Can simulation be the ultimate in the process of design of airport terminals? The computer has come a long way in assisting in daily requirements. In the last decade its role has been more computational; hence its use as a simulation tool was restricted to evaluate or derive parameters that influence the quantitative aspects required for the design of airport terminals. The objective is to develop a tool that can assist in the design process that should not be limited to achieving quantitative results only. It should also represent qualitative properties and values and establish ways to evaluate them. This would help in the optimization of performance of facilities, evaluate, make selections and judgments, and predict the quality of performance standards. The computer should not be treated as another computational tool because its potential can be used in a decision-making capacity [i.e., Expert Systems, in the process of airport terminal design (*14*)]. Ultimately, the computer may become an essential element in the airport design process or even play the role of an equal partner. Some of the areas in which simulation and computers can assist are related to human factor issues of "way finding" at airport terminals and "distress reaction" to emergencies. Some pioneering research in the domain of architecture and computer-

aided design (*15–17*) can make a strong and significant impact on the modeling and simulation of airport landside. In the opinion of the authors, this can be achieved through object-oriented tools and the use of artificial intelligence techniques of representation, semantic network, and expert systems (*14*). This paper provides a discussion of a preliminary ongoing study; more details will be presented in subsequent publications.

REFERENCES

1. S. Mumayiz. Overview of Airport Terminal Simulation Models. In *Transportation Research Record 1273*, TRB, National Research Council, Washington, D.C., Jan. 1990, pp. 11–20.
2. *Collection of Calibration and Validation Data for an Airport Landside Dynamic Simulation Model.* Prepared for Federal Aviation Administration by Research and Special Programs Administration and Transportation Systems Center, FAA-EM-80-2, Washington, D.C., Jan. 1980.
3. L. McCabe and M. Gorstein. *Airport Landside. Vol. II. The Airport Landside Simulation Model (ALSIM), Description and Users Guide.* Federal Aviation Administration, Washington, D.C., June 1982.
4. *Summary of Airport Planning Models—Background Paper.* Professional and Technical Services, Transport Canada, Ottawa, Ontario, Canada, Jan. 1988.
5. G. Carrara, Y. E. Kalay, and G. Novembri. *A Conceptual Framework for Supporting Creative Architectural Design. Evaluating and Predicting Design Performance.* John Wiley & Sons, New York, N.Y., 1990, pp. 303–322.
6. H. A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, Mass., 1989.
7. P. H. Winston. *Artificial Intelligence.* Addison-Wesley, New York, N.Y., 1984.
8. B. Ezzell. *Object-Oriented Programming in Turbo Pascal 5.5.* Addison-Wesley, New York, N.Y., Nov. 1989.
9. D. Hu. *Object-Oriented Environment in C++: A User-Friendly Interface.* MIS Press, Portland, Oreg., 1990.
10. J. D. Smith. *Reusability and Software Construction C and C++.* John Wiley & Sons, New York, N.Y., 1990.
11. A. Van Dam and J. D. Folley. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, New York, N.Y., 1984.
12. R. Horonjeff and F. X. McKelvey. *Planning and Design of Airports.* McGraw-Hill, New York, N.Y., 1982.
13. *Airport Terminal Reference Manual.* 7th ed. International Air Transport Association, Geneva, Switzerland, Jan. 1989.
14. S. Mumayiz. Development of Airport Terminal Design Concepts: A New Perspective. *Transportation Planning & Technology Journal*, Vol. 13, pp. 303–320.
15. M. D. Gross and C. Zimring. Predicting Wayfinding Behavior in Buildings: A Schema-Based Approach. In *Evaluating and Predicting Design Performance* (Y. Kalay, ed.). John Wiley & Sons, New York, N.Y., 1990, pp. 293–302.
16. F. Nichols. Design for Pedestrians: a CAD-Network Analysis Approach. In *Evaluating and Predicting Design Performance* (Y. Kalay, ed.) John Wiley & Sons, New York, N.Y., 1990, pp. 303–322.
17. M. O'Neill. A Neural Network Simulation as a Computer-Aided Design Tool for Predicting Wayfinding Performance. In *Evaluating and Predicting Design Performance.* John Wiley & Sons, New York, N.Y., 1990, pp. 275–291.

Publication of this paper sponsored by Committee on Airport Landside Operations.