Table 1. Impact analysis: group 1.

Mode	Scenario					
	1[a]	2[b]	3[c]	4[d]	5[e]	6[f]
Automobile	−1%	−2%	−0.3%	−2%	−0.1%	+0.1%
Air	+7%	+5%	−0.3%	−5%	−0.4%	+0.1%
Bus	−2%	+4%	−0.4%	+27%	+2.3%	+0.2%
Rail	−2%	+4%	+7.4%	−4%	−0.5%	−2.4%

[a]Family income rises 10 percent.
[b]Gasoline costs rise 50 percent.
[c]Frequency of rail service outside the northeast doubles.
[d]80-mph speed limit for buses.
[e]Bus fares decline by 10 percent.
[f]Rail fares rise by 10 percent.

Table 2. Impact analysis: Southwest Corridor.

Mode	Scenario			
	1[a]	2[b]	3[c]	4[d]
Automobile	−1%	−7%	+4%	−0.4%
Air	+4%	+19%	−19%	−0.7%
Bus	+3%	+20%	+7%	−1.1%
Rail	+3%	+17%	−9%	+18.2%

[a]Two cents per mile toll for automobiles.
[b]Two-h increase in automobile travel time.
[c]70-mph speed limit for automobiles and buses.
[d]Rail frequency triples.

less true in the Northeast Corridor where service is already competitive. On the other hand, travel time is a more important factor in rail demand in the Northeast.

As noted above, these sensitivity analyses can be performed for any scenario affecting model variables. The model is sensitive to changes in various costs, travel time, frequency, income, access, service availability, and automobile ownership. With little additional effort, the changes in demand can be expressed in policy-relevant terms such as energy saved, tax revenues earned, change in vehicle miles of travel, and change in transportation expenditures.

Various technical improvements, such as sample expansion and sensitivity to total demand, have been undertaken to expand the applicability and usefulness of the model. Even in current form, it can be a useful forecasting and policy tool.

Table 3. Elasticities for group 1 and corridors.

Mode	Cost	Time	Wait	Access
Group 1				
Automobile	−0.076	−0.303	0.0	0.0
Air	−0.618	−0.159	−0.048	−0.107
Bus	−0.321	−1.101	−0.054	−0.061
Rail	−0.373	−0.251	−0.463	−0.100
Northeast Corridor				
Automobile	−0.112	−0.555	0.0	0.0
Air	−0.538	−0.163	−0.031	−0.108
Bus	−0.267	−0.954	−0.022	−0.062
Rail	−0.315	−0.825	−0.050	−0.059
Southwest Corridor				
Automobile	−0.072	−0.292	0.0	0.0
Air	−0.359	−0.164	−0.023	−0.141
Bus	−0.311	−0.209	−0.038	−0.070
Rail	−0.361	−0.335	−0.419	−0.168

ACKNOWLEDGMENT

This research was performed under the sponsorship of the U.S. Department of Transportation. I gratefully acknowledge the guidance of Carl N. Swerdloff of the Office of the Secretary of Transportation.

REFERENCES

1. P. Stopher and J. Prashker. Intercity Passenger Forecasting: The Use of Current Travel Forecasting Procedures. Proc., Transportation Research Forum, 1976, pp. 67-75.
2. P. Stopher, J. Prashker, B. Smith, and E. Pas. Final Report to Amtrak on Intercity Passenger Forecasting from the National Travel Survey Data of 1972. Northwestern Univ., Evanston, IL, 1976.

Publication of this paper sponsored by Committee on Computer Graphics and Interactive Computing.

Interactive UTPS: Implementation Under a Timesharing Environment

JEROME M. LUTIN AND MATTHEW LIOTINE

This paper reports on the development of interactive computer programs for the Urban Mass Transportation Administration's Urban Transportation Planning Systems (UTPS). The programs, originally designed to run under an IBM 360 or 370 OS environment, were executed under a conversational monitor system (CMS) timesharing environment. The aim was to reduce turnaround time and explore future interactive capabilities of the programs. Interactive versions of programs INET, UPATH, UPSUM, ULOAD, UROAD, NAG, UMATRIX, UFIT, and ULOGIT were developed. The paper describes the process involved in creating CMS exec programs to control the program compilation and data set manipulation without any job control steps. Each UTPS program exec is described along with other supporting software that was developed. Finally, a summary of the problems encountered in transforming the software and data files from CS to CMS is presented.

This paper summarizes the development of an interactive version of several Urban Transportation Planning System (UTPS) computer programs. UTPS is a collection of computerized and manual techniques to aid planners in the assessment of urban transportation systems. It was developed and maintained by

the Urban Mass Transportation Administration (UMTA) and the Federal Highway Administration (FHWA). The computerized element of UTPS is a battery of computer programs designed to operate on an IBM 360/370 computer system. These programs were designed to operate under batch processing (1). Part of this research focused on the application of these programs in an interactive timesharing environment. The aim was to facilitate an advanced UTPS training course, where the actual use of the programs would be provided.

The course was held at Princeton University in March 1980 and was taught as an advanced use course for transportation professionals with previous exposure to UTPS programs as a prerequisite. The course emphasized a more systematic analysis of transportation system management (TSM) alternatives through the use of existing and newer UTPS programs. The programs used were ULOGIT, UFIT, INET, UPSUM, UMATRIX, UPATH, ULOAD, NAG, and URCAD (2).

The course described a typical corridor analysis for a large metropolitan area and involved intensive use of the previously mentioned UTPS programs not covered in the existing one-week course. The case study was broken down into three phases:

1. Phase I--Build and update networks and build paths and skims,
2. Phase II--Calibrate and apply mode split model and assign transit trips, and
3. Phase III--Prepare highway subarea and assign all trips.

A brief description of the function of each UTPS program set up on the interactive system is described below:

1. INET--Transit network builder. Takes as input a description of the highway network, transit routes, and links. Produces as output a transit data base, network files, and reports on transit supply parameters.
2. UPATH--Transit path finders. Takes network files from INET as input. Produces weighted impedance paths via transit between zones. Produces file of minimum path trees, and optional interzonal fare and distance matrices.
3. UPSUM--Impedance summarizer. Takes path file produced by UPATH and "skims" impedances. Produces output files containing impedance matrices of transit wait time, transit in-vehicle time, numbers of transfers, and total transit travel time.
4. UMATRIX--Matrix manipulator. Takes as input a variety of data set formats and performs user-specified arithmetic and logical operations. Can be used to factor trip tables and apply demand model formulas to produce transit trip tables.
5. ULOAD--Transit assignment program. Takes as input transit trip tables and interzonal transit paths. Assigns transit trips to system to obtain transit line volumes. Assignment rules can be varied by user.
6. NAG--Network aggregation program. Takes network files and trip tables as input. Allows user to analyze a portion or "window" of the network in detail while reducing detail of network outside the window. Helps simplify output and reduce cost of analysis. Produces aggregated files as output.
7. UROAD--Highway network model. Takes as input an historical record file of the highway network produced by program HR or HNET. Calculates highway travel times between zones. UROAD also assigns automobile trip tables to the highway network to produce link volume reports. Capacity restrained and stochastic traffic assignments can be performed. Output includes reports and updated network files with speeds and traffic volumes.
8. UFIT--Demand model calibration. Takes as input card image files or binary calibration files and performs linear least-squares regressions to fit models. User can establish conditional expressions to screen observations and constrain parameters.
9. ULOGIT--Logit model calibration. Takes as input a binary calibration file of observations of individual user (e.g., disaggregate) mode-choice data. Estimates parameters to fit observed mode choice to an S-shaped (logit) curve. Used to calibrate disaggregate multinomial logit mode split models.

A network for the Shirley Highway Corridor in Virginia, developed at UMTA, was used for the case-study problem. The network data were shipped to Princeton on tape, and the Princeton APL graphics system was used to prepare master maps of the corridor on mylar film. Zone centroids, load nodes, access links, and highway links were plotted with distinguishing graphical conventions. Centroid and node numbers were superimposed. Two maps were produced, one showing the entire Shirley Highway Corridor case-study area, and a second showing downtown southwest Washington at an enlarged scale. Each team received several black-line copies of the maps to lay out transit alternatives.

The course attendees were required to make changes to a transit system network in order to achieve better operational characteristics than its original ones. Within the larger transportation network, a detailed section of the highway network was focused on to evaluate the impacts of the changes previously made. The course emphasized the instruction of newer UTPS programs as opposed to the older programs, specifically the use of INET and NAG in network analysis, the use of ULOGIT and UFIT in the calibration of demand models, and the use of a new version of UMATRIX. The actual interactive use of these programs was featured that had never before been incorporated in any previous UTPS training session. This facilitated actual program use during the session, and quick turnaround time for output.

OBJECTIVES

There were several objectives sought in the development of software to execute the UTPS programs under a time-sharing environment. First, the software would have to allow the user to interact with each program at a CRT terminal and would enable the user to operate each program with a minimum amount of knowledge of job control language and data set manipulation. The turnaround time and output retrieval would be quick. The user would be able to examine the results of a program run at the terminal as well as in the form of a paper printout.

The end result of attaining these objectives was a highly intelligent CMS exec program that would function with a minimum amount of computer knowledge on the part of the user. This would allow more concentration on the use of UTPS for a particular analysis without coding job control language (JCL) steps and little use of the conversational monitor system (CMS) language. The following sections describe in greater detail the execs that were developed.

CMS EXECS

The IBM Virtual Machine Facility 1370 (VM/370) is a system control program that controls "virtual machines". A virtual machine is the functional equivalent of a real computer whereby the user can control its operation from a terminal using a command

language. In effect, the computer simulates for each user an entire computer system, which appears dedicated to the user. The language used to operate UTPS under this environment is called the conversational monitor system, or CMS (3).

CMS operates under yet a higher command language, the Control Program (CP). CP controls the resources of the physical computer machine and also manages the communications among several virtual machines and between a virtual machine and the physical or "real" system. CMS is the conversational operating system designed specifically to run under CP. It can simulate many of the functions of the IBM Operating System (OS).

The file is an essential unit of data in the CMS system. CMS disk files are unique to the CMS system and cannot be read or written using other operating systems. CMS files are named according to a file identifier consisting of three fields: a filename, filetype, and filemode. CMS files are written on disk in 800-byte physical blocks, regardless of whether they have fixed or variable length records.

CMS is a language of statements consisting of active verbs and nouns. An exec is a CMS file that contains many of these executable statements instead of data items. The statements may be CMS or CP commands or exec control statements. The execution can be conditionally controlled, have variables, and may expect arguments to be passed to it. In its most complex form, an exec can contain thousands of records and may resemble a program written in a high-level programming language. It was in this form that the CMS exec was used to operate UTPS.

Under CMS, it is possible to execute many OS language processors: Assembler, VS Basic, CS FORTRAN IV, OS COBOL, and OS PL/1. This enabled the execution of UTPS, whose programs are mostly written in FORTRAN IV, but which does have some subroutines written in Assembler and COBOL. By using CMS, one can assemble and invoke compilers by using special commands. Thus, a typical UTPS program such as UPATH could be implemented in the following fashion. First, a previously compiled object module (a machine language version of the program) is put through a linkage editor loader (software that assigns the program to certain memory addresses in the system) using CMS commands. File definitions must have been previously made for all input and output files, similar to JCL file definitions. This step produces a load module, or load program, which is yet another form of the original program. Once a load program is present in a virtual machine, it could be executed using a CMS command. A CMS exec was written for each UTPS program that was highly robust in that it made this process virtually invisible to the user. The exec performed this operation, overseeing the data file manipulation and program execution.

SPECIFICATIONS AND PROGRAM DESCRIPTIONS

All of the UTPS programs have some features in common. A control card file is needed as input along with other input files. As output, a printout file is produced along with other files. The execs had to allow the user the option of naming all input and output files or use default names supplied by the exec. A default-naming convention was developed and is described in the following section. The execs also had to allow the user to make changes to the control card file interactively, and to view the printout at the terminal screen before deciding whether a hard paper copy is to be printed.

The execs also had to deal with user errors. Typing mistakes at a terminal keyboard are inevitable and thus the execs had to alert the user of the error. In addition, a feature was needed to allow the user to exit the exec at any point and to provide the option of deciding whether to proceed with the program running or to revert back in the exec to make additional changes.

The execs written for each UTPS program were somewhat similar in structure, but were individually designed to accommodate the unique features of each program. Some standard subexecs were written to perform functions required by most of the programs. The following are brief descriptions of each UTPS program exec and the specifications required of it.

1. INET. This program operates in two modes: update and build. In the build mode, the program reads in a historical record (HR) file and, with "NET=F" on the control card, produces a transit data base (TDB) file. In the update mode, the program expects only the TDB file and produces a new or updated TDB file in subsequent runs. In either mode, if NET=F is specified, five new files are produced for use by other programs. Thus, the exec for INET queries the user as to whether he or she wishes to completely build or update a TDB. It also has the capability of examining the NET parameter to see if NET files are to be produced, so that the appropriate file definitions could be called. In addition, the exec accommodates a file of ROUTE cards and allows the user to make changes to these cards if necessary.

2. UPATH. This exec was not very complex in comparison with INET and was designed to resemble a canned process. The exec expects four NET files from a previous INET run, with the default naming of these files unchanged from INET. The exec allows the user to name the output path file and the two non-transit link files. It also examines the control card options "DIST", "FARE", and "IMPED", to see if the user wished to output a distance and/or fare impedance matrix.

3. UPSUM. This exec receives the path file from UPATH with the same default name and produces a skim file.

4. UFIT. This exec, like INET, operates in two modes. Program UFIT reads in a binary calibration file compatible with program ULOGIT and from this file creates a new calibration file as output, which is conditional on the specification of the "FILE: keyword in the control card file. Thus, the exec determines whether this keyword is present. Another mode of operation is the case in which the user wishes to create a new calibration file from card images. These images are usually attached to the control card file and are used only if the option "BUILD=T" is specified. The program then ignores any input calibration file and processes only the card images. This feature was compensated for in the exec by again examining the control card file for the BUILD=T specification.

5. ULOGIT. This exec is structurally the simplest since only one file is input in addition to the control card file and no files are output, except printout files. However, a special FORTRAN program was written to produce a prediction success table as an optional output.

6. UMATRIX. Because of the flexibility involved in the names and numbers of input and output files, this exec must search the control card file for certain characteristics. First, the number of input J-files is determined by examining the specifications of files J1 through J8. A J9 specification alerts the exec of an output file. The exec then queries the user for the exact name of each J-file specified in the control cards. Z-files are handled in the same manner. The use of look-up tables can also be accommodated.

7. ULOAD. This exec was designed as an almost canned process, with an extra provision for an input-loaded legs file from a previous ULOAD run and the file of selected volumes conditional on the specification of the "ALINE/CLINE" criteria in the control card file. In addition, the exec can determine if the user specified the "GENT" parameter, eliminating the need for an input J1 trip matrix file.

8. NAG. The main feature of this exec is the detection of "NET" and "GENT" parameters in the control card file that eliminates the need for the user to input an HR file and trip matrix.

9. UROAD. It was decided that this exec would not implement any of the plotting features of program URCAD and would only address the traffic assignment capabilities of the program. For the purpose of this research, the exec only accommodates the insertion of multiple trip tables. Other features will be incorporated later.

10. ULOG. This exec facilitates the printing of the user's log report after a specified number of UTPS program runs. This exec allows the user to specify this number and maintains the log file on disk for subsequent updating.

GENERAL PURPOSE EXECS AND PROGRAMS

As mentioned earlier, several subexecs were written to perform general purpose tasks used by most of the main execs. These execs are listed below and are briefly described.

1. DEEM. This exec resides on the user's disk and obtains the default filemode for files used in the program. As mentioned earlier, CMS file designations consist of three fields: a filename, filetype, and filemode. A default-naming convention was developed whereby the user's disk mode serves as the default filemode for all files. In most cases, files would maintain a filetype of the name of the UTPS program that produced them. Filenames, in most cases, resembled the original filenames contained on the DD statements in the program's catalogued procedure. Thus, for example, a file designated as TDB INET A is a TDB file produced by program INET and resides on a disk in the user's virtual machine that has been accessed as A, which is the default filemode. Route card files and control card files were handled in a different manner as described in the following section.

2. UTPS. This exec links the user's disk to two software disks. The first is a disk containing the execs and load modules. The second disk contains utility software supported by the Princeton University computer system.

3. GETFID. This exec responds to an input file specification by the user and searches all disks in the user's virtual machine for file and verifies whether it can be used.

4. REPLY. This exec simply checked to see if a user's reply was correct or not, i.e., contained no typing errors for a yes or no reply.

5. CPUNIT. This exec is executed when a program begins running and informs the user that this has occurred.

6. CPUFINL. This exec is invoked at the end of the program run and informs the user of the CPU time of the run and the return code if the program terminates abnormally.

7. DISPLAY. This exec takes the printout file and displays it on the terminal screen. The user can specify the pages and report of the printout he or she wishes to view. It also allows the user the option of printing the file on the system printer.

8. ANTEST. This exec is a subexec of GETFID and, like REPLY, examines the user's entry of a filename to see if it was typed correctly.

9. PRMCLR. This exec clears all the file definitions in the virtual machine after the program run.

10. ODSK. This exec is a subexec of GETFID and determines whether an accessed disk can have information written on.

11. ST. This exec intercepts the print file from the real system printer and spools it to the user's disk. The return code is also read from the print file and passed to CPUFINL.

12. STCLEAR. This exec clears the console stack.

Two general purpose programs were written to supplement the above execs. The first was an Assembler program called FIND. A key element in the overall exec structure, FIND was designed to specifically search the control card file for any character string argument. This enabled the execs to determine the required input/output file based on keywords such as NET in INET or BUILD in UFIT.

The second program was a FORTRAN program called REPS2. This program was used in the exec DSPLY and performed a minor task of determining which UTPS reports are available on the printout file.

FINAL EXEC STRUCTURE

The final versions of the execs conformed to the specifications mentioned earlier. Each was designed to guide the user through the program and demanded little knowledge of CMS from the user. Implicit in the design of each exec was the assumption that the user had at least some knowledge of the UTPS program use. A virtual machine is established for the interactive UTPS environment. A typical user would have read or write access to his or her own personal disk where input and output files could be maintained. All of the necessary software, including the program execs and load modules, reside on another disk from which the user can read only, leaving the software protected and intact. A temporary disk is also formatted on which the user may be able to read and write scratch files and temporary data sets. This disk provides space for scratch files and any extra temporary space if required by the user. Since this is a temporary disk, its contents would be wiped out once the user logs off the system.

The virtual machine configuration is established when the user invokes the exec called UTPS. Once this is done, the user can invoke any exec by simply entering the UTPS program's name.

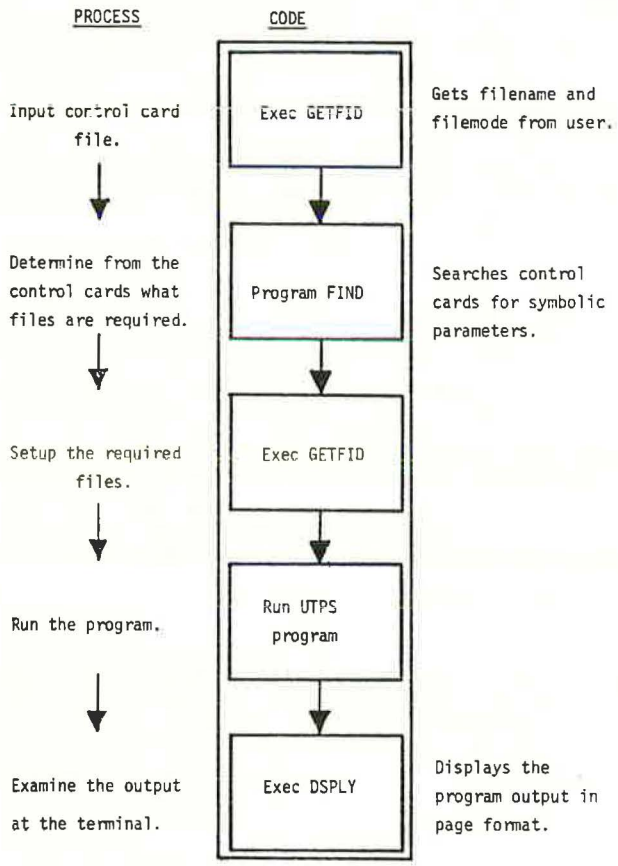
Each exec called several general purpose execs and programs described in the previous section. A general flow-chart for the execs is shown in Figure 1. Each exec follows a process that is summarized as follows:

1. Input Control Card File. Exec GETFID is invoked first, retrieving from the user the name of the control card file specified. The exec then seeks to locate this file in the virtual machine.

2. Determine the Required Files. The control card file is then searched by program FIND for symbolic parameters and keywords that inform the exec what files are to be either read in or written. For example, the presence of the keyword BUILD=T in the INET control card file would inform the exec that five NET files will be written by the program.

3. Set Up the Required Files. Exec GETFID once again is invoked. This time it will ask the user to specify the names of both the input and output files. Input files are then located within the virtual machines. The specified names are inserted in the file definitions so that output files are written with the names given by the user.

Figure 1. General flowchart for UTPS execs.



4. Run the Program. The program's load module is then called and the program is executed.

5. Examine the Output at the Terminal. The output file is retrieved and, through exec DSPLY, is displayed on the terminal screen. The user then has the option of requesting a paper printout of the file.

Mention should be made of the default-naming convention for route card and control card files. Unlike the file convention described earlier, these files assumed a filetype of the UTPS program name by which they were to be used. For control cards, a default file name of CNTRLXX was used. The suffix XX is substituted by a two-digit code from 01 to 99, which is assigned by the user. For example, a control card file might be designated as CNTRL03 UPATH A. The filename of CNTRL03 designates this as a control card file. The filemode A is the default mode of the disk in the user's virtual machine on which this file resides.

The two-digit suffix enables the handling of many sets of control card files on the user's disk and provides the user with a convenient way of identifying them. It also facilitates any easy default way of specifying a particular control card file designation to the exec. By entering "." followed by a two-digit number corresponding to the two-digit numerical suffix in the control card file's filename, the exec will locate this file in the user's virtual machine. For example, if in using program ULOAD, the user responds to the exec's prompting for the name of the control card file by entering ".05", the exec will search for a file designated as CNTRL05 ULOAD A, assuming A is the default filemode. If this file is residing on a disk accessed under a different filemode, say B, then the user needs only to enter ".05 B". Route card files for program INET were handled in a similar manner, with ROUTEXX as the default filename prefix.

UTILITY EXECS

Several utility execs were written during the software development to perform general tasks. One is an exec called SCOPY, which enabled the transfer of files formatted as variable block spanned (VBS) from one disk to another. The normal CMS "COPY" command tends to disrupt the format of these files and thus SCOPY was used in lieu of this command. The original version of SCOPY was written at UMTA.

SUMMARY OF PROBLEMS ENCOUNTERED IN SOFTWARE DEVELOPMENT

Implementing the UTPS programs, which were originally designed to operate in an OS batch environment, in a timesharing environment was not a direct process. Certain inherent OS features of the programs that are not handled by CMS had to be overcome.

A problem arose with respect to data files. It was unclear in the case of files that were originally formatted as VBS as to what the proper data control block (DCB) parameter was required in the CMS file definition. By using the original parameters from the DD card in the catalogued procedure failed in most cases. The problem was resolved by literally guessing at the record format, using a trial-and-error procedure that iterated different permutations of the RECFM parameter until a successful run was obtained. VBS files also presented a problem mentioned earlier, whereby they could not be transferred by using a normal CMS copy command. A special exec was required to copy files.

Problems arose in INET with respect to sorting. Within its internal structure, INET makes several calls to the IBM OS Sort/Merge routine. Under VM/370, this utility is unavailable. A counterpart to this is the CMS SORT utility that is called in lieu of the OS SORT. However, INET uses, in addition to the OS SORT, a UTPS sorting routing called SORT. Problems arose in calls to this routine. It was later discovered that the calls to this routine were identical to the CMS SORT and thus the calls had to be changed.

A problem occurred in ULOAD with respect to the sorting of the loaded legs file. This file is passed through two subroutines, E15 and E35, which are user entries in the OS SORT routine. E15 processes the input loaded legs file for the OS SORT while E35 processes the output file. These routines were designed to process a blocked file. The CMS SORT failed to sort this file due to this characteristic. Thus, E15 and E35 were replaced by two routines, UNBLK and BLK, that, respectively, unblocked and blocked the file.

Program NAG produces an output trip table in a compressed format that deletes rows for which all cells are zero. UROAD did not accept this file. A modification to the file was done by using UMATRIX. UMATRIX will insert a row of zeros where rows are deleted. It was also used to change the file format by using the keyword "OUTPBT".

CONCLUSIONS AND RECOMMENDATIONS

The interactive implementation of UTPS was not a direct process. Before proceeding on this endeavor, a knowledge of the use of the UTPS programs in transportation system planning as well as computer science must be obtained. The transportation planning and UTPS experience enabled the specification

of the exec structures. Implementation of the execs involved a substantial degree of knowledge of the IBM VM/370 system, CMS, several OS compilers, and UTPS. Although this process, in general, did not warrant the entering of the UTPS internal source code, in some instances it was unavoidable. Because the execs were developed with the intention of being used as an instructional tool for a one-week course, many of the accessory features of the programs, such as the plotting capabilities, were not incorporated into the execs. The end result was an exec that required little knowledge of CMS and VM/370 on the part of the user but some knowledge of the use of the UTPS programs in transportation planning.

It is recommended that this process be further crystalized and documented so that typical users may find it easier to implement UTPS in a CMS or any other timesharing environment. Interactive computing is now gaining widespread interest, and any interactive capabilities of software would thus make it more attractive.

The interactive software described in this paper has been turned over to the UMTA Office of Planning Methods and Support and is undergoing further development. It is expected that interactive versions of many UTPS programs will be used in further advanced UTPS training sessions.

ACKNOWLEDGMENT

This research was sponsored by UMTA's Office of University Research and Training. We wish to sincerely thank Philip Hughes, Nathaniel Jasper, and Judy Z. Meade of the Office of Policy Research for their support of this work. A large measure of credit for the short-course development goes to Larry Quillian of UMTA's Office of Planning Methods and Support. Ed DeLong, Chief of the Software Support Division, played a major role in helping debug the system. Most of the programming was done by William Collins, a 1981 graduate of Princeton University.

REFERENCES

1. M. Liotine and J.M. Lutin. Report on the Development of the Workshop on Interactive Applications of UMTA/FHWA Planning Tools. Urban Mass Transportation Administration, U.S. Department of Transportation, Sept. 1980.
2. Urban Transportation Planning System Introduction. Urban Mass Transit Administration, U.S. Department of Transportation, Feb. 1980.
3. IBM Virtual Machine Facility 1370: CMS User's Guide, 3rd ed. IBM Corp., White Plains, NY, March 1979.

Publication of this paper sponsored by Committee on Computer Graphics and Interactive Computing.

Interactive Model for Estimating Effects of Housing Policies on Transit Ridership

JEROME M. LUTIN AND BERNARD P. MARKOWICZ

This paper reports on computer graphics developed as part of an interactive computer model designed to assess the impact of housing policies on transit ridership in urban transit corridors. A set of programs was written in APL to implement the model in an interactive computer environment, with computer graphics used for both input and model output. A mode-split model that uses U.S. Bureau of the Census data predicts ridership for the transit line, based on discrete combinations of mode and access mode including walk-and-ride, park-and-ride, kiss-and-ride, and feeder bus. The program permits the analyst to input alternative residential patterns, with respect to location and density, in the transit corridor and to evaluate the effects on transit ridership by comparing various alternative housing policies. Computer graphics are used at two levels. First, as an input mode, graphics allow the planner to create new transit route alignments and station locations by using a screen cursor. The program then models station choice from the zones, on the basis of a number of variables, including driving or walking times to stations, transit fares, line-haul travel times, etc. As an output mode, graphics are used to display socioeconomic data, mode-split results, or any algebraic combinations of input or output data. Different types of graphic displays are used for data presentation at the zone level or station level. Throughout the development of the graphics, special attention was given to the readability of the output. The paper reflects the general effort to produce more visually attractive and commonly understandable outputs. Included in the paper are a description of the program design and organization, examples of graphic output, and a discussion of the ability of the model to provide useful output to policymakers.

Planners and urban policymakers have long recognized that a strong relationship exists between urban development forms and the existence of rapid transit systems in cities. In recent years, new transit systems have not led to significant positive changes in urban development. It is believed that the existing high level of automobile accessibility tends to obscure the increases in mobility achieved by transit. Many planners and policymakers believe that transit systems can be more effective in meeting the travel needs of the public, more energy efficient, and require less subsidy if land use planning in transit corridors can be coordinated with the planning of the transit system itself.

To achieve better coordination between transit planning and land use planning, the Urban Mass Transportation Administration (UMTA) has been making grants to cities to encourage urban development in transit corridors. However, there are major questions that need to be answered about the kinds of policies to be implemented. Planners need to know, for example, what kinds of housing should be encouraged in transit corridors. Should land close to transit stations be reserved for high-density apartments or be kept open to provide large lots for park-and-ride patrons? Given that land use regulations are difficult to enact and enforce, how does noncompliance with the plan affect the desired result? Because of the many unanswered questions, this research was directed toward the development of some quantitative tools that would provide planners