A communications environment emulator

by JACK M. PEARLMAN and RICHARD SNYDER

Honeywell, Inc.
Waltham, Massachusetts

and

RICHARD CAPLAN
Consultant
New York, New York

The problem

Communications based systems and software, especially complex systems and software, present an exceptionally difficult checkout and debug problem to the implementor. Conventional techniques require him to check his often times labyrinthine maze of response paths in a plodding, single step by single step, long drawn out and expensive manner. He checks the non-communications or non-real time portions of his program easily enough. The methods of testing standard peripheral functions are well enough known and adequate tools exist for the job. The techniques for checking basic data processing logic are available with every batch oriented system. Communication based system checkout is complicated by the real time nature of the systems. The give and take of on-line terminals, the multiplicity of choice, the need to process data in various stages of disarray, the necessity of leaving some partially complete processing sequence and returning to it at a later time, these are the problem factors. These are the things which make conventional checkout procedures a millstone around the neck of a system builder.

He is required to generate huge quantities of test data, each item of which is keyed to a specific path, expecting a specific response. He simulates as well as he can the action of his operational system with a console or perhaps with a card reader or maybe a single actual terminal. The usual scenario then calls for him to use several actual terminals and operators, with the attendant confusion, logistic difficulties and expense. The final step is a full blown system test sequence using all the equipment and all the operators and he hopes that all the parts are exercised and that all the problems are solved.

This technique works, to a point. It does produce a workable system. It does, eventually, succeed in resolving a large percentage of possible problems. The cost of doing so, however, is extremely high. Devising a specific datum for a specific path test is time consuming, especially since the number of paths may be counted in the hundreds or thousands. Using live terminals and live operators presents a major organizational problem and requires large cash outlays. Such a system too, violates a prime rule of scientific investigation—reproducibility of an experiment. The programmer is rarely positive that a fix works since he can never exactly recreate the conditions causing failure.

The solution

Taking these objections, and others, together, we decided to find a better way to check and debug communications based systems and system software. The result of our efforts, is "The Honeywell Communications Environment Emulator" (HCEE).

HCEE is a multi-purpose communications network simulator whose prime purpose is to aid in the checkout and debugging of communication software, real time or otherwise. A secondary purpose is to permit experimentation with software and remote terminal configurations where the terminal itself is non-existent. A tertiary reason for being is to evaluate communications software by determining its operational limits and its response to unusual stress.
The system may be quantitatively defined in terms of its design goals. It will simulate up to 63 lines, with several distinctly different classes of terminal represented, up to eight terminals per line and generate at least 70,000 100-character messages per hour on an H-1200 computer. It will require a minimum 65K C.P. with about 30K taken up by instructions. It will handle as many message types as a user cares to define and check for as many response types as he wishes. The system will generate as many message variations as a user provides defining entries for, in some cases using a random selection process, in others using a round-robin technique. HCEE will eventually be able to introduce perturbations into a data stream to simulate line control errors and other line associated faults.

The Emulator will reside in its own processor, with the system under test (target) in second C.P. The two computers communicate via standard communications hardware. The target system thinks that it sees a real communication network, with real terminals and operators, in the outside world. Actually HCEE generates all queries and responds to all answers by simulating or emulating the characteristics of the actual terminals and their operators. This approach to checkout is itself complex and by no means inexpensive. But it is far less complex and far less expensive than conventional checkout means and carries with it many distinct advantages. Use of a computer permits reproducibility and creation of exhaustive system evaluation reports. By recording every message transmitted (control and data) and every message received, along with the system time, it becomes possible to “play back” an entire sequence as a way of validating changes to the target system. These same records become the basis for a complete series of reports describing the operation of the system in detail and the success with which various classes of messages were handled, including deliberately created error messages. The computer, too, permits automatic message generation and response analysis based on skeletons defined by the user.

The product

The form which the HCEE design took was partially determined by a number of criteria for use, not the quantitative ones described earlier, but qualitative ones. One major criterion was that the system be modular. By modularity we imply that the component elements be so loosely related to one another (or decoupled) that any element could be readily replaced by a similar component of greater or less complexity. That is, except for some minimum capability, new routines could be added and/or existing ones deleted from the system. This criterion is essential to a system such as HCEE since its utility will be partially dependent upon our ability to adapt it to handle changed operating conditions, and provide capabilities not originally considered. Modularity is achieved by use of elaborate table structures and list processing techniques which insulate processing routines from one another. All contact between routines is through tables, lists or well defined interfaces to small central control programs.

A second important criterion was that the system be able to generate large message volumes within user defined response constraints. This implies that the time limitations associated with program or data storage on external media probably would become intolerable. Consequently it was decided that the entire program and its data base be core resident.

A further goal was to make the program usable internally by Honeywell as a systems checkout tool and eventually externally by our customers, with a third potential user being independent service bureaus. These were some of the factors which led us toward residence in a separate C.P., with independent communications capability. We could now use this tool to debug a target system hundreds of miles away and work with target systems residing in non-Honeywell computers.

The last major criterion considered required that the generated system be independent of any existing operating system. This decision was made to insure usability in a smaller computer than would otherwise be necessary and to insure better control over HCEE operation by not being tied to operating system conventions. We will use operating system MOD 2 to generate the system but, in order to take advantage of the macro assembly and linkage editing and loading facilities of MOD 2.

Phase structure

Functionally, HCEE may be considered as having four discrete phases; Test Generation, Test Initialization, Test Execution and Test Result Analysis.

During Test Generation, the data and environment Specifications which define the test case are integrated into the HCEE execution phase structure. This operation is accomplished through application of the Macro Assembly facility. Macros were selected as the prime configuration medium for several reasons: the facility exists; it is fairly powerful, it is well understood; macros are relatively easy to use; they require minimal manpower to implement; they are, of all alternatives, the cheapest to use. Macros were an expedient choice but more importantly, they do not impose any major limitation upon either the design or the final use of the system. One further consideration is the relative ease of change which macros provide. Certainly as our experience with the Emulator grows, we will find areas for improvement.
The macro approach simplifies at least one such area. The chief alternative to the macro approach, one which is attractive for future use, is an English like language. Unfortunately however, this choice possesses as disadvantages (for the initial system) the negatives of almost every element cited for the macro approach.

Macros defining the particular test (via parameter lists) are assembled into table structures which serve as the data base during test execution and provide system specialization control information for use during test initialization. The macro language is divided into four logical divisions which functionally correspond to those of COBOL. Thus, the Identification Division consists of user specified information for unique identification of a test run or series of runs. The Environment Division defines the virtual terminal population being emulated as well as user specified message volume and scheduling constraints. The Data Division provides the templates necessary for query generation and response analysis. Finally, the Procedure Division defines the information necessary to proceed through a transaction. That is, the decision information necessary to decide "what to do next". For example, what query to send next based upon an analysis of the last response received.

Test initialization takes the linked HCEE programs and tables (on tape in self loading format), performs basic communication startup and housekeeping functions, and dynamically generates certain table entries from the data supplied at test generation time. In addition, the first message to be sent across every active line is generated and scheduled.

During test execution, HCEE generates, transmits, and logs queries and receives, analyzes, error codes, and logs target system responses. Although a limited amount of controlled console operator intervention during test execution is acceptable, the execution phase is capable of running with no outside intervention and with no call on external data storage except for logging purposes.

Following termination of a predetermined test interval supplied by the user, the test result analysis phase is initiated. During this phase the Log Tape is sorted into a variety of sequences (as called for by the user specified reports) and the report generation sub-programs are executed using the log as input, to produce target system performance reports for diagnostic and efficiency measurement purposes.

All four phases of HCEE may be executed contiguously on a load and go basis, or they may be broken into three parts: Generation, Initialization and Execution, and Result Analysis.

**Basic facilities**

HCEE incorporates the following features:

a. **Control**
   1. Specification by the user of desired test interval length, and line transaction volume as a time dependent function.
   2. Specification by the user of both the real and virtual (terminal) hardware environment in terms of the number, line distribution, and type of terminals to be emulated.
   3. Specification by the user of context dependent, scheduled time intervals to be applied in initiating queries that represent the continuation of transactions.
   4. Specification by the user of the syntactic and semantic rules which establish the relationship between target system responses received and the next call generated from the same terminal. These rules are supplied as logical table structures which can be probabilistically weighted to ensure the generation of a prescribed transaction mix, and the production of a statistically viable sample of certain expected operator behavior patterns, as (for example) "think time" to absorb received information.
   5. Specification by the user of errors to be introduced into the transmitted data stream either in accord with statistical measures of type and frequency or continual generation of specific errors.

b. **Query Generation**
   1. Generation of fixed or variable length (format) queries with dynamic selection of key user specified parameter fields.
   2. Dynamic selection of the appropriate query to generate within an interactive context defined by the user in a transaction syntax definition.

c. **Response Analysis**

   Identification of fixed or variable length target system response types fixed keyword or parameter matching search methods.

d. **Error Detection**
   1. Dynamic marking during test execution of all response errors. These errors include invalid (unrecognized) target system responses as well as valid hardware and system error conditions, such as line associated errors.
   2. Dynamic marking during test execution of all calls which are generated after their scheduled transmission time, distinguishing
a deterioration in HCEE performance from a reduction in Target System efficiency.

e. Reporting (and result analysis)
Post mortem production of a comprehensive set of HCEE test performance analysis reports including such critical measures as response time graphically plotted as a function of transaction volume, polling delay (interval measuring operator request to actual transmission) as a function of volume, HCEE service time by message type, etc.

Emulator program components

HCEE incorporates various fixed and generated components (elements). Each element, along with a brief description of its function, is presented below.

1. Executive—The central element. It is responsible for interrupt management and time slice allocation.

2. Terminal Operations Subsystem—The generative and analytic component of HCEE. It is composed of a fixed element, the Diagrammer; and generated elements, the Query-Response Lists, Vocabulary Lists, and Transition Diagrams. The Diagrammer traverses the Transition Diagrams using the Query-Response Lists in conjunction with the Vocabulary Lists for Query generation and the Query-Response Lists alone for Response analysis. The balance of the paper will concentrate on this subsystem.

3. Transaction Scheduler—The element which provides the time increment $\Delta t$ (relative to current time $t$) at which to initiate a transaction for a particular terminal. The decision of what $\Delta t$ to produce is based on user supplied volume constraints and a random number generator. The Transaction Scheduler is activated by the Communications Subsystem.

4. Communications Subsystem—This element is responsible for transmission and reception of all traffic to and from the system under test. The Communications Subsystem is triggered by the Executive Routine following an interrupt from the multiline controller.

5. Path Selector—The module which has the final responsibility for selecting which query to send. It chooses the query on a user specified percentage basis from a population of queries selected as a result of analysis by the Diagrammer.

6. Console Routine—The element which is responsible for outputting system messages to the console and allowing the console operator to input action requests.

7. Buffer Linker—The element which has the responsibility for linking buffers of a message together and activating the Diagrammer in the case of a fully linked Response, or the Communications Subsystem in the case of a fully linked Query.

8. Message Logging Routine—The element which logs to magnetic tape all transmissions between the system under test and HCEE

Figure 1 illustrates the relationships between the elements of HCEE.

The key

Before continuing with the narrative, it is advantageous to diverge slightly and define some essential terms.

1. Query—Any message generated and transmitted by HCEE regardless of initiating impetus. In other words, even if the message in question is an answer to a request from the target system, it is considered a query.

2. Response—Any message received by HCEE regardless of whether solicited by HCEE or not.

3. Transaction—A sequence of messages (made up of queries and responses) which perform an iden-
tifiable application function. For example, a transaction involving a sale might consist of:

a. How many widgets in stock (query)
b. Ten widgets in stock (response)
c. Sell 10 widgets to ABC Co. (query)
d. Confirm 10 widgets to ABC Co. (response)

4. Transition-State-Diagram—A nodular structure providing finite reference points for states of the system. A completed action on an object may be considered as a state of being of that object. There are indicators and conditions associated with that state and there is an expectation of one or more of a finite number of future operations which might occur on the object as a result of what has already taken place. For example, consider a ball at rest on the ground. It's state of being may be defined as "at rest." This state carries with it information, some of which may be indicators, some descriptors, some measures of expectancy of future events. An example of an indicator may be recognition of the fact that the ball is stationary; a descriptor might be that it has zero kinetic energy; expectance, that it will be thrown up in the air or that it will be thrown against a wall. Any other occurrence is an error or failure of the system involved. Further, consider that at the moment anything happens to the ball, it is started on the way to a new state of being regardless whether what happens is expected or not. If we now define the state of being as a "node," the process of determining which of several possible futures has occurred as execution of an action routine, and completion of that process as arrival at a new state of being (node), then we have explained the concept behind the opening description as a "Nodular Structure." All that remains is to emphasize that all possible relationships are predetermined by the user. Should he make a mistake in defining a relationship, or if he leaves out a relationship, the system will be immediately aware of the mistake. In that sense the process is a heuristic one and allows checkout of the logical dependencies between information.

Terminal operations subsystem

The Terminal Operations Subsystem is the heart of HCEE. It contains the prime processing components and is responsible for the generation of queries, checking of responses and maintenance of transaction flow within HCEE. It provides a user with the ability to drive his simulated system over a sustained interval (which may be measured in minutes or hours), without creating an extremely large, highly specific test case library; with surety that he will exercise every path through the target system that he can conceive of and describe, including error paths. Moreover, the user is assured that events within the network will occur in a timely manner, in accord with volume and operator (or system) reaction time constraints defined by him.

The test transaction structure of HCEE is in the form of directed strings of items (Queries and Responses) which permit an exceptionally complex set of relationships to be described. The simplest variation is a one dimensional sequence such as:

```
Q₁ → R₁ → Q₂ → ... → Qₙ
```

Complex networks which introduce choice may be constructed similarly:

```
    Q
     \|  \\
    Q   Q
     \|  \\
      R  R
       \|  \\
        Q
```

For response analysis purposes, it is presumed that the paths emanating from a Query Node represent all possible responses to that Query Node (including a default or error state) and conversely that the paths from a Response Node represent all possible queries generatable as a result of the response. The Query/Response mechanism may be further improved by assigning selection probabilities to various queries so that queries are generated roughly with the frequency that they are expected in the actual system. This ability has the added advantage of facilitating experimentation to determine the effect on a system of different message type loading within a specific overall volume.

Test data base

The Test Data Base consists of a collection of transition diagrams (Transaction Logic Network) similar to those just described, which reference a file of user created skeletal message descriptions known as "Query-Response List". The Query entries within the Q-R Lists may in turn reference a set of key word strings or "Query Vocabulary List" which supplies variable word
values for inclusion in generated Queries. The relationships are illustrated in Figure 2.

- node = state defined by the transmission or reception of a specific message type (nodes are either Q or R type where Q = query and R = response).
- path = implicit execution of an Action Routine (query generator or response analyzer) necessary to move from one node (state) to the next as directed by the path.
- first node for transaction
- transaction termination node which corresponds to a return of the terminal involved to an idle state.
- Format Template for query n which is used by the query generator to produce the actual message.
- Identification template for response m which is used by the response analyzer.
- Variable length word strings (circular linked lists) 1-k which serve as lexical source material for query building, i.e., varying data elements such as part numbers, which are inserted into defined fields in a query template to create a complete query ready for transmission.

The transition diagrams themselves are represented by a set of variable length records, one for each Node in the network. They are created during the test generation process from user supplied information. The Query Node referred to in Figure 3 is the place at which a particular Query is generated and a subsequent response is analyzed. Upon positive identification of the response, control passes to the proper Response Node for preparation of the next Query. Non-identification of a response, indicating either a failure somewhere within the system or a mistake in describing the system by the user, causes control to pass to the error routine associated with the Query Node. HCEE supplies a single common error routine. The user is free to replace it or to add others as he wills. Query path selection using the associated probability measure, and calculation and assignment of the required intermessage interval, are done by the Path Selector subroutine of the Diagrammer.

Figure 2—Relationship of transition diagrams to query generation tables

Figure 3—Record structure for transmission diagrams
Query-response list

The Query-Response List is acted upon by programs called Action Routines, which are executed by the system while it is traversing the Transition Diagrams. There are two basic types of Action Routines; the Response Analyzer and the Query Generator. The Response Analyzer determines which Response Node to activate. That is, it decides which Response type of a number of possible Responses has actually arrived. The Query Generator concatenates the fields specified in the Query-Response List with values extracted from the Vocabulary List so as to create complete messages from the skeletal formats of the Q-R List.

Query-vocabulary list

Creation of a wide variety of queries from a single Query skeleton is made possible by the Query-Vocabulary Lists. Included in a Query skeleton record are a series of field descriptors which define fields as constant or variable. Variable field identifiers contain pointers to vocabulary list records containing a series of possible values for the particular field. Normally the values will be selected sequentially on a round-robin basis. The technique allows the inclusion of deliberate data errors to check data handling characteristics of the Target System. Figure 4 illustrates a vocabulary list field.

Sample network

The diagrams of Figure 5 give a pictorial representation of a very simple one transaction network. In this illustrative example, the first query of the transaction will be “DO YOU HAVE?” and then one of a set of items listed in a Vocabulary List. There are three possible responses, “YES,” “NO,” or “REPEAT MESSAGE.” In the case of the first two responses, the transaction is terminated. In the case of “REPEAT MESSAGE,” 50 percent of the time a new query will be sent and 50 percent of the time the transaction will be terminated.

Diagrammer

The Diagrammer is the central control routine which coordinates the activity of the Response analyzer and Query Generation Action Routines. At any point in time, for a given terminal, a set of transaction continuation possibilities exist. The set consists of those paths emanating from the current node address of that terminal. Earlier it was stated that recognition of a particular Response led to generation of a specific Query. Actually there may exist a number of valid queries. When selection of a single Query from a population is needed, control passes to the Diagrammer, which retrieves the current node address and in turn activates the Path Selector, which may apply a uniformly distributed random number generator and/or path frequency information present in the node record, to select a single query generation path. Responses are processed and identified by a series of comparisons rather than selections.

Path selector

The prime responsibilities of the Path Selector are selection of the next Query to generate within a given transaction context, and calculation of the time at which the message should be transmitted. We must distin-
guish two conditions, each with its own scheduling requirements; initiation of a transaction and generation of a query within an on-going transaction.

Every line or line group (defined as a number of lines with completely homogenous initial transaction requirements) has one or more transaction initiation nodes (Node 0 for Query initiate transactions—Null Node otherwise). Each possible transaction path for a given line or line group emanates from one such unique node. Each potential transaction carries a usage frequency (provided by the user) which the path selector uses as a selection constraint. Scheduling for Transaction Start activity is not done by the Path Scheduler, but rather by the Transaction Scheduler. This latter routine selects inter-transaction intervals based upon transaction volume constraints imposed by the user.

Query selection for on-going transactions has already been described. Scheduling for such queries is done by the Path Selector. This is accomplished by adding a user defined inter-message interval to the arrival time (from system time zero) of the activating response, and posting the result as the scheduled transmission time.

CONCLUSION

We have just described a technique for providing a real time checkout facility to a programmer or software designer writing a communication based system. The design is flexible, modular and expandable. The program is expected to be ready by the fourth quarter of 1969.