ACKNOWLEDGMENTS

The authors wish to acknowledge the support of this work from the National Science Foundation Grant ENG 76-06221, Walter P. Murphy Fund, and the DuPont Foundation.

NOTATION

A = area, m^2
b = bottoms flow rate, kg-mole/s
d = distillate flow rate, kg-mole/s
F_j = flow rate of feed to stage j, kg-mole/s
F_{ij} = flow rate of component i in the feed to stage j, kg-mole/s
H_j = enthalpy of the vapor on stage j, J/kg-mole
H_j^* = enthalpy of the vapor entering the rectifying section after compression, J/kg-mole
h_j = enthalpy of the liquid on stage j, J/kg-mole
h_j^* = enthalpy of the liquid entering the stripping section after expansion, J/kg-mole
h_{Fj} = enthalpy of the feed stream into stage j, J/kg-mole
I = number of components
K_{ij} = vaporization equilibrium ratio for component i on stage j
L_j = liquid flow rate out of stage j, kg-mole/s
n = number of stages; stages 1 and n denote the condenser and reboiler, respectively
P = pressure, N/m^2
Q_j = heat removal rate from stage j, J/s
T = temperature, °K
U_j = liquid sidestream withdraw rate from stage j, kg-mole/s
U_{ov} = overall heat transfer coefficient, $J/m^2 \cdot s \cdot °K$
V_j = vapor flow rate out of stage j, kg-mole/s
W_j = vapor sidestream withdraw rate from stage j, kg-mole/s
x_{ij} = liquid mole fraction of component i on stage j
y_{ij} = vapor mole fraction of component i on stage j
α = relative volatility

LITERATURE CITED

Brugma, A. J., "Fractional Distillation of Liquid Mixtures, Especially Petroleum," *Dutch Patent No. 41850* (Oct. 15, 1937).

Freshwater, D. C., "Thermal Economy in Distillation," *Trans. Inst. Chem. Engrs.*, 29, 149 (1951).

———, "The Heat Pump in Multi-Component Distillation," *Brit. Chem. Eng.*, 6, 388 (1961).

Guillaume, M., See Mariller reference.

Haselden, G. G., "An Approach to Minimum Power Consumption in Low Temperature Gas Separation," *Trans. Inst. Chem. Engrs.*, 36, 123 (1958).

Mah, R. S. H., ed., "Innovative Design Techniques for Energy-Efficient Processes," Summary Report to the National Science Foundation, NTIS Accession No. PB243651/AS (May 30, 1975).

Mariller, Charles, "Distillation et rectification des liquides industriels," 3. èd Dunod, Paris (1943).

Perry, R. H., and C. H. Chilton, ed., *Chemical Engineers' Handbook*, pp. 3-190 and 3-195, McGraw-Hill, New York (1973).

Petlyuk, F. B., V. M. Platonov, and I. V. Girsanov, "The Design of Optimal Rectification Cascades," *Intern. Chem. Eng.*, 5, 309 (1965*a*).

Petlyuk, F. B., V. M. Platonov, and D. M. Slavinskii, "Thermodynamically Optimal Method for Separating Multicomponent Mixtures," *ibid.*, 555 (1965*b*).

Petlyuk, F. B., V. M. Platonov, and V. S. Avet'yan, "Optimal Rectification Schemes for Multicomponent Mixtures," *Khim. Prom.*, 42, No. 11, 865 (1966).

Prengle, H. W., J. R. Crump, C. S. Fang, M. Frupa, Henley, and T. Wooley, "Potential for Energy Conservation in Industrial Operations in Texas," Final Report on Project S/D-10 Governor's Energy Advisory Council, The State of Texas (1974).

Robinson, C. S., Clark Shove, Edwin Richard and E. R. Gilliland, *Elements of Fractional Distillation*, 4 ed., McGraw Hill, New York (1950).

Seader, J. D., "Multicomponent Distillation by the Wang-Henke Method," in *CACHE Computer Programs for Chemical Engineering Education, Vol. VI. Stagewise Computations*, J. H. Christensen, ed., National Academy of Engineering (1973).

Soave, G., "Equilibrium Constants from a Modified Redlich-Kwong Equation of State," *Chem. Eng. Sci.*, 27, 1197 (1972).

Stupin, W. J., and F. J. Lockhart, "Thermally Coupled Distillation—A Case History," *Chem. Eng. Progr.*, 68, No. 10, 71 (1972).

Wang, J. C., and G. E. Henke, "Tridiagonal Matrix for Distillation," *Hydrocarbon Processing*, 45, No. 8, 155 (1966).

West, E. H., and J. H. Erbar. "An Evaluation of Four Methods of Predicting the Thermodynamic Properties of Light Hydrocarbon Systems," Proc. 52nd Annual Convention of Natural Gas Processors Assoc. pp. 50-61, Dallas, Tex. (Mar. 26-28, 1973).

Wodnik, R. B., M.S. thesis, Northwestern Univ., Evanston, Ill. (1976).

Manuscript received March 30, 1977; revision received June 13, and accepted June 22, 1977.

System Structures for Process Simulation

LAWRENCE B. EVANS
BABU JOSEPH
Department of Chemical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts

and

WARREN D. SEIDER
Department of Chemical Engineering
University of Pennsylvania
Philadelphia, Pennsylvania

The plex data structure is proposed for use in an advanced computing system to model chemical processes. The plex is shown to permit increased modularity and flexibility over systems with dimensioned array structures. Two methods are suggested to create and operate upon the plex: the problem oriented language and the problem oriented calling programs. The concept of a routing plex is introduced as a means for specifying the path of calculations when building block routines call upon other routines and choices exist at each level.

SCOPE

This paper presents concepts for design of an advanced computing system to meet the needs of process engineering in the 1980's. The requirements for such a system were discussed earlier (Evans and Seider, 1976). In this paper, we consider trends in process engineering and digital computation and their impact on system design. Two aspects to design of the system are examined: data structure and programs.

A limitation of present day process simulators is their use of dimensioned arrays, usually residing in COMMON in FORTRAN programs, to store variables describing a process. This leads to inflexibility, because the designer of the system must know all variables of interest at the time the system is designed. It becomes difficult to incorporate new types of variables without altering the layout of data in COMMON and changing every routine that was defined on the basis of the original layout.

An alternative type of data structure is suggested based on the plex instead of on fixed arrays. In a plex structure, first proposed by Ross (1961), information is stored in blocks of contiguous storage locations known as beads. Beads of any length are created dynamically from a pool of free storage as needed during execution of the program. They are referenced and linked together by means of pointers. Beads may contain integer values, real values, Boolean values, character strings, and pointers intermixed as needed to describe an element of the model.

The problem of interfacing building block routines with the plex data structure is considered. Two methods are proposed to create and operate on the plex: the problem oriented language and the problem oriented calling programs. The use of a routing plex to specify the computational path is described for situations where many options exist.

CONCLUSIONS AND SIGNIFICANCE

The plex provides an elegant way to represent the data describing a process model. Beads are the primitive elements from which the plex data structure is built. The ability to intermix real numbers, integer numbers, Boolean values, and character strings within a bead permits a more natural description of the attributes of process models. Pointers show relations among information in the data structure.

From a programming viewpoint, the plex data structure has many advantages over fixed array structures. It promotes modularity: the building block routines require only a single argument, the pointer to the portion of the plex upon which they are to operate. It provides flexibility: for example, stream classes for new types of streams are easily represented. It allows easy modification to the process configuration by deleting beads and creating new ones. It does not waste storage for large numbers of unused entries in predimensioned arrays. It does not limit problem size as occurs with predimensioned arrays.

The principal disadvantage is that execution time may be increased by the time required to access information from the plex. This limitation is not serious, however (Joseph et. al., 1977).

Two methods have been described for creating and operating on the plex data structure: the problem oriented language and the problem oriented calling programs. It is easier for the user to read and write programs written with the problem oriented language, but a translator or interpreter is required to process the language. The language processor adds complexity and may not fit onto smaller computers. The problem oriented calling programs require no special processors but are more difficult to use. They consist of calls on subprograms, and they focus upon the individual building blocks and modularity. A system that works with the problem oriented calling program can be adopted later for use with a problem oriented language.

The plex data structure permits easy specification of the path of computations when building block routines call upon other routines and choices exist at each level. Only a single pointer to the routing plex need be passed as an argument from routine to routine.

As an improved representation of information, the plex could have a significant effect upon those who develop modular computer programs for modeling chemical processes. Just as the use of vector and tensor notation stimulated theoretical work in transport phenomena, and the use of state variables promoted new developments in automatic control, the use of plex data structures could provide a stimulus to developing modular programs for computer aided chemical process analysis.

Despite the considerable effort expended by the chemical and petroleum companies on process simulation systems, there exists a need for an advanced computing system for the analysis of process flow sheets. The requirements for such a system were presented in an earlier paper (Evans and Seider, 1976). This paper discusses some concepts for design of the system.

The next generation of computing system for flow sheet analysis must address trends in both process engineering and digital computing.

TRENDS IN PROCESS ENGINEERING

Process engineering is being called upon to solve new problems. Processes are needed to convert coal into clean fuels, to use new raw materials such as oil shale and tar sands, to store the energy from nuclear power plants in the form of fuels such as hydrogen or methanol, and to reprocess waste materials such as trash, sewage, and agricultural wastes. Systems must be optimized to reduce both capital and operating costs while having less pollution and greater plant safety and reliability. Techniques of process simulation now used successfully in the chemical and petroleum industries need to be applied in other industries such as paper, food, and metals processing.

Present day simulation systems handle a limited class of processes (mostly hydrocarbon processes). They lack the ability to model diverse types of streams (such as solid-liquid and solid-vapor) characterized by different sets of variables (size distributions, particle shape parameters, porosities, etc.). They lack unit operation subroutines for process units that involve solids, for example, cyclone separators, kilns, fluidized-bed heat exchangers, filters, centrifuges, driers, electrostatic precipitators, crystallizers, screens, and equipment for crushing and grinding.

New types of flow sheet analysis are becoming important, including dynamic simulation for the study of

PROPERTY	STREAM 1	STREAM 2		STREAM N
TEMPERATURE	T1	T2		TN
PRESSURE	P1	P2		PN
ENTHALPY	E1	E2		EN
ETC.				

Fig. 1. Array of stream variables.

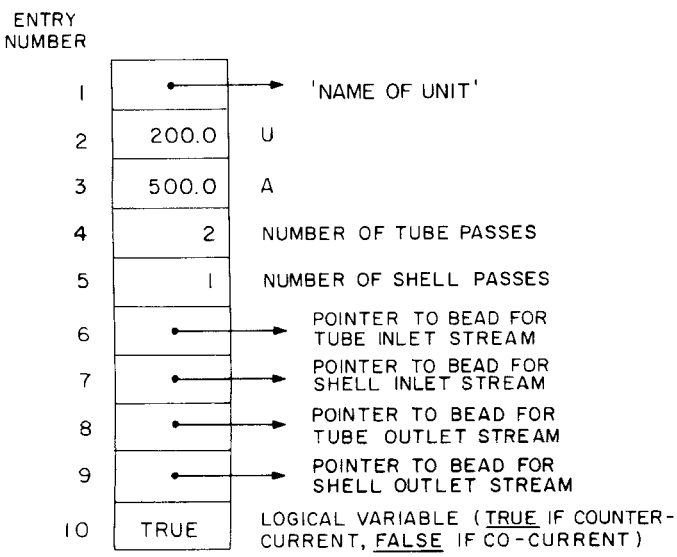


Fig. 2. Example bead to contain the equipment parameters for a heat exchanger model.

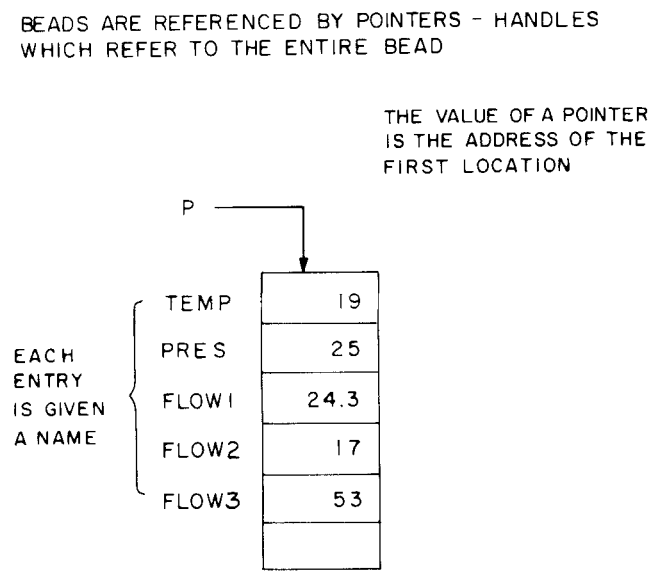


Fig. 3. Typical bead.

start-up, shutdown, emergency transients, and control; balances on thermodynamic availability for studies of process energy efficiency; and fault-tree analyses for studies of system safety and reliability.

It is essential that an advanced computing system for process analysis be flexible. It must be easy to add new types of process units, new types of streams, new types of chemical compounds, and new types of flow sheet analysis.

TRENDS IN DIGITAL COMPUTATION

The revolution in digital computer hardware has resulted in smaller computers with increased capabilities at decreasing cost. Today's minicomputer is as powerful as the IBM-7094 which was the mainstay of scientific computing in the 1960's, when many of today's simulators were designed. By the 1980's, equivalent capabilities will exist in desk-top computers based upon microcomputer technology. Developers of an advanced computing system must recognize that engineers of the future are likely to have very powerful, compact, personal computers at their fingertips.

CONSIDERATIONS FOR SYSTEM DESIGN

The challenge is to design a computing system that is flexible so that one may extract portions of the system, modify them, and tailor them to new applications. At the same time, the system should be decomposable into subsets to be used on smaller computers.

We believe that such a system must be built around a set of modular building blocks of data and programs that can, in effect, be taken off the shelf and assembled into a wide range of applications. The collection of building blocks could be augmented as new needs emerged. A capability would be provided to simplify their use through a problem oriented language. This approach is opposed to the design of a massive system to solve all of the problems presently envisioned. Such a system would probably be obsolete by the time it was completed.

There are two aspects to the design of the system: data structure and programs. The data structure contains the numerical values and alphanumeric information required during solution of the problem. The programs operate on the data structure. This paper focuses upon the data structure and its implications for building block routines, and the use of a problem oriented language.

THE PLEX DATA STRUCTURE

Most present day process simulators are written in a high level language such as FORTRAN. The data structure consists of a set of dimensioned arrays to store the variables describing a process. Figure 1 shows a typical two-dimensional array to store the stream variables for a process. Each column in the array contains the values for the variables of a single stream. The arrays are dimen-

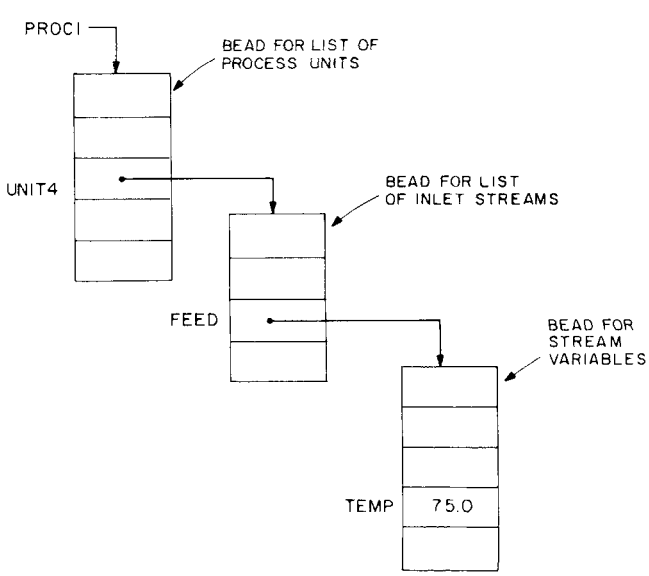


Fig. 4. Use of nested pointers.

sioned to contain all of the variables for each stream in the flow sheet and usually reside in COMMON storage so different subprograms may access them.

The problem with fixed array data structures is their inflexibility. For example, one must know at the time the system is designed all variables of interest in a given stream. It becomes very difficult to alter the layout of data in COMMON storage without changing every routine that was defined on the basis of the original layout.

To gain the flexibility and modularity required for an advanced computing system, we propose an alternative type of data structure based on the plex instead of the fixed array. The plex structure (from the Greek word plexus meaning an interwoven combination of parts or elements in a structure or system) was first proposed by Ross (1961) and suggested for use in computer aided process design by Porter (1969). Fay (1971) used such a structure in a prototype system for computer aided design of heat exchange networks.

The basic element of the plex data structure is a bead, which is a block of contiguous storage locations; the number of locations may vary from one bead to the next. A bead may contain different types of entries, including integer values, real values, Boolean values, pointers to other beads, and pointers to alphanumeric strings. An example bead to contain the equipment parameters for a heat exchanger model is illustrated in Figure 2.

Beads are referenced by pointers, which may be thought of as handles that refer to the entire bead. The value of a pointer is the address of the first storage location of the bead.

A typical bead is shown in Figure 3. Each entry is given a name such as TEMP, PRES, FLOW1, etc. These names are descriptive of their meanings.

Data in the bead shown in Figure 3 might be accessed as follows. To retrieve the contents of the second entry of the bead, the statement

$$R = VAL(P, 'TEMP')$$

would place the contents of the second location into R. To store a value in the fourth entry, the statement

CALL STOR(P, 'FLOW2', 17.)

would place the value 17. in the fourth location. In this example, VAL and STOR are data management operations that would be performed by appropriate routines.

An entry in a bead may contain a pointer to another bead as shown in Figure 4. The entry UNIT4 in the bead pointed to by PROC1 contains a pointer to a bead containing the list of inlet streams. The entry FEED contains a pointer to a bead containing the stream variables. Nested calls on the data management routines can be used to retrieve the temperature of the feed stream to unit 4 in process 1.

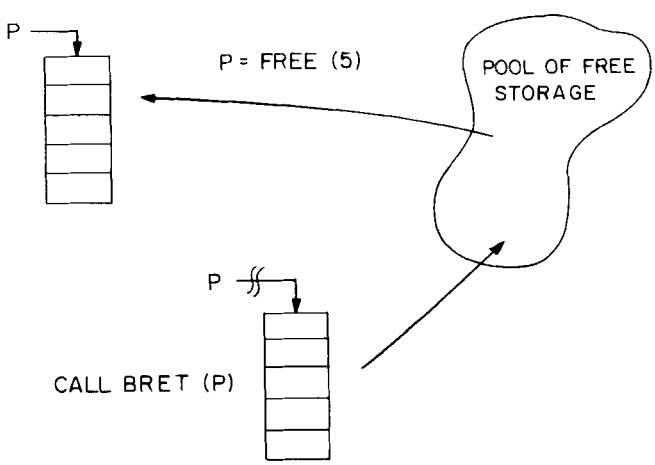
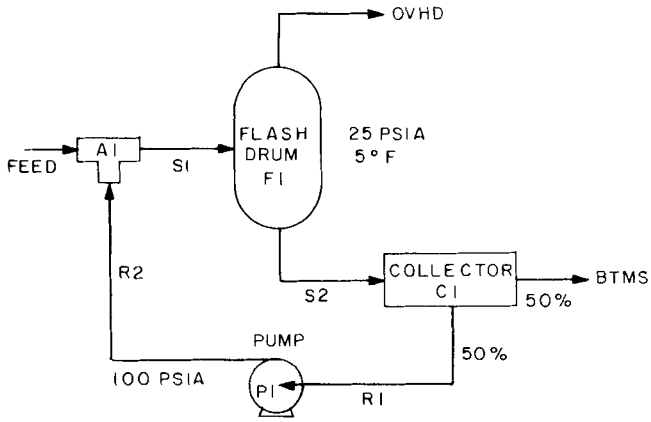


Fig. 5. Pool of free storage.



THE FEED STREAM IS AT 85 °F AND 100 PSIA
IT CONSISTS OF: 100 LB/HR METHANE
500 LB/HR ETHANE
50 LB/HR PROPANE

Fig. 6. Flash with recycle process.

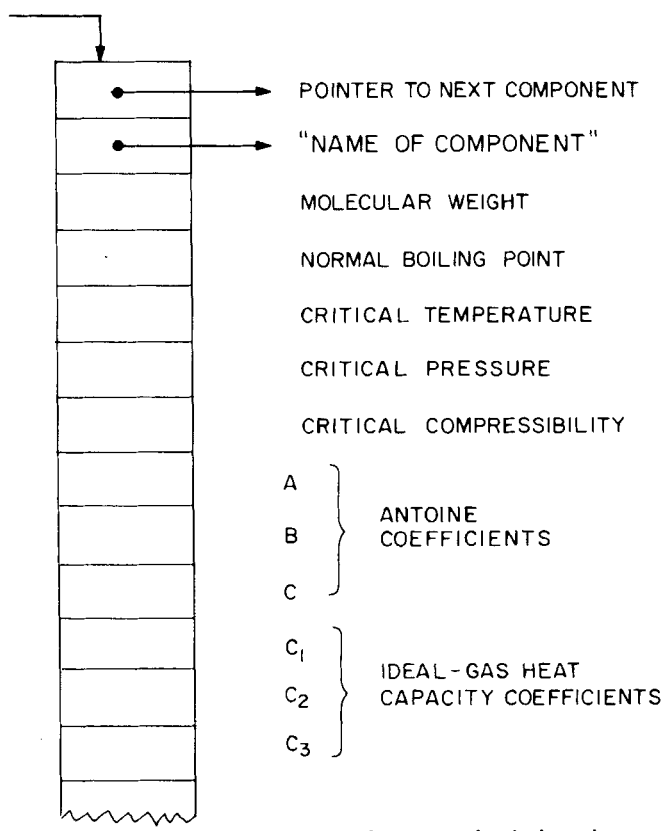


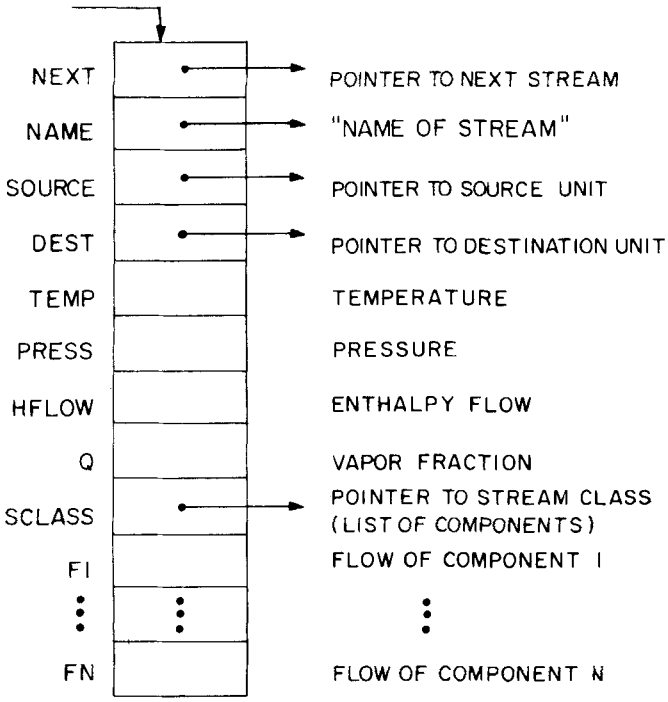
Fig. 7. Bead to contain constants for a pure chemical species.

Beads are created dynamically during execution of a program from a pool of free storage as illustrated in Figure 5. The statement

$$P = FREE\ (5)$$

would assign to P the value of a pointer to a bead with five locations. When the bead was no longer needed, it could be returned to the free storage pool by the statement

CALL BRET (P)



THERE WILL BE DIFFERENT STREAM BEADS FOR DIFFERENT TYPES OF STREAMS (SOLIDS, MULTI-PHASE, ETC.)

Fig. 8. Bead to contain stream variables.

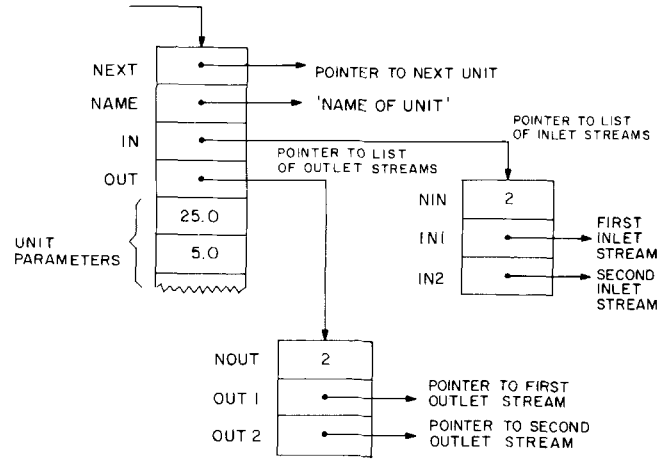


Fig. 9. Bead to contain unit parameters.

PROBLEM ORIENTED LANGUAGE TO CREATE PLEX STRUCTURE

The use of a plex data structure will be illustrated to represent information for steady state simulation of the simple recycle process shown in Figure 6. Beads will be created to contain physical constants for pure chemical species, stream variables, and unit parameters, etc. Typical beads are shown in Figures 7 to 9.

Figure 10 shows the statements in a hypothetical problem oriented language to simulate the recycle process for the conditions given in Figure 6. The first section creates the data structures and will be described next. The second section defines the building blocks and calculation order to be used in simulation. Finally, the third section prints the results of simulation.

```
CREATE DATA STRUCTURE
   BEGIN DEFINITION PROCESS 'FLASH WITH RECYCLE';
   RETRIEVE COMPONENTS METHANE, ETHANE, PROPANE;
   DEFINE STREAM CLASS ONE TYPE = VAPLIQ COMPONENTS (METHANE, ETHANE, PROPANE);
   DECLARE STREAM CLASS ONE FEED R2 S1 OVHD S2 BTMS R1;
   CREATE A1 TYPE = ADD/FEED R2/S1;
   CREATE F1 TYPE = ISOTHERMAL FLASH (P=25PSIA, T=5DEGF)/S1/OVHD S2;
   CREATE C1 TYPE = SPLIT (F1=0.5, F2=0.5)/S2/BTMS R1;
   CREATE P1 TYPE = PUMP (POUT=100PSIA)/R1/R2;
   SPECIFY FEED (T=85 DEGF, P=100 PSIA, FLOWS = 100, 500, 50 LB/HR);
   END DEFINITION;
DEFINE BUILDING BLOCKS AND CALCULATION ORDER
   BEGIN DEFINITION STEADY-STATE SIMULATION SEQUENCE 'FLASH WITH RECYCLE';
   L1 ASSUME R2 (T=70, P=50, FLOWS=50, 100, 50); CALCULATE A1/MIX;
      CALCULATE F1/IFLSH (TERROR=0.001); CALCULATE C1/SPLIT;
      CALCULATE P1/PUMP; CONVERGE R2/WEGSTEIN (TOL=0.01) L1;
   END DEFINITION;
   SIMULATE 'FLASH WITH RECYCLE';
PRINT INFORMATION FROM DATA STRUCTURE
   PRINT STREAMS; PRINT UNITS; PRINT COMPONENTS;
   SAVE 'FLASH WITH RECYCLE';
   END PROBLEM;
```

Fig. 10. Hypothetical problem oriented language to simulate the flash with recycle process.

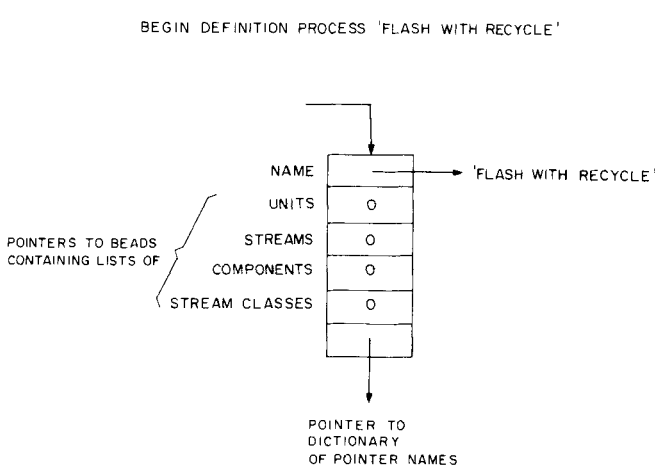


Fig. 11. Statement to create a process bead.

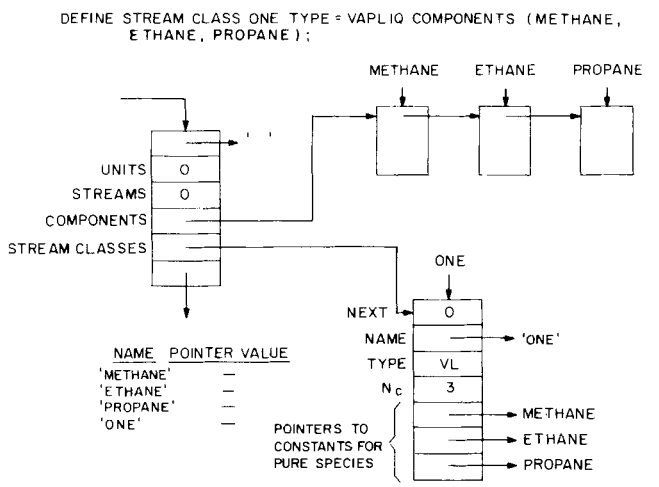


Fig. 13. Statement to create a stream class bead.

The first statement creates a process bead with a title for the process. Entries in the process bead are provided which will eventually contain pointers to the list of units, streams, components (chemical species), and stream classes; these pointers are initially set at zero. A pointer is established to a bead containing a dictionary of pointer names. Figure 11 illustrates the resulting bead.

The next statement retrieves data records for each chemical component from the data bank. The physical constants for each component are stored in beads which are linked together by pointers originating at the process bead. The name of the pointer to each new bead is entered into the dictionary. The new beads and pointers are shown in Figure 12.

Next, a stream class bead is created which defines stream class ONE to be for vapor-liquid streams containing methane, ethane, and propane. Its pointer value is placed in the process bead and pointer dictionary. It contains pointers to the beads containing constants for methane, ethane, and propane. This new bead is shown in Figure 13. Typical processes will contain several stream classes, especially when solids are present.

The next statement creates a bead for each stream in the process: FEED, R2, S1, OVHD, S2, BTMS, and R1. The stream class for all streams is ONE. A pointer is created from the process bead to the bead for the FEED stream and, in turn, from each stream bead to the next. Pointers to the unit from which the stream originates and to which it is destined are not set. Figure 14 illustrates the new beads and pointers.

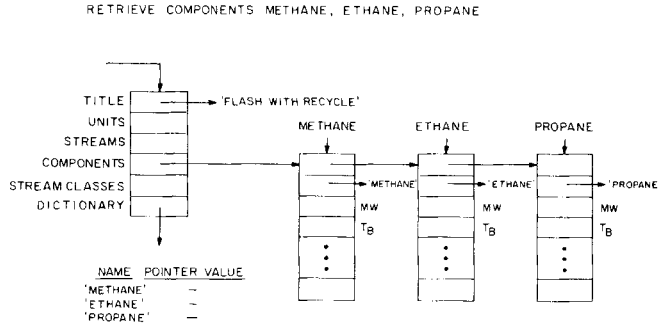


Fig. 12. Statement to create beads for physical constants of chemical species.

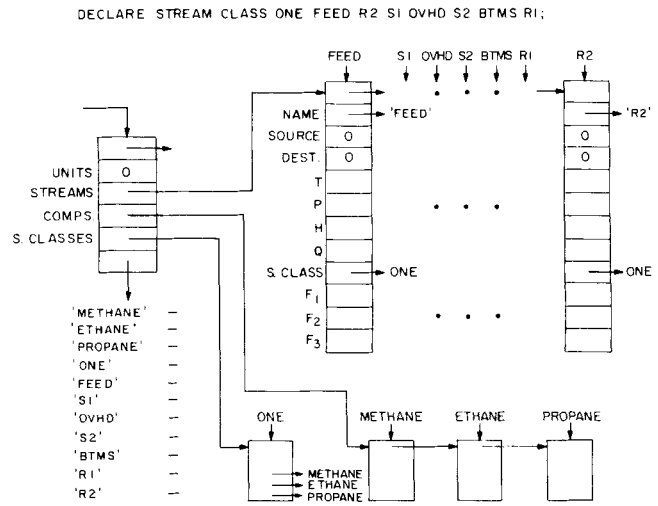


Fig. 14. Statement to create stream beads.

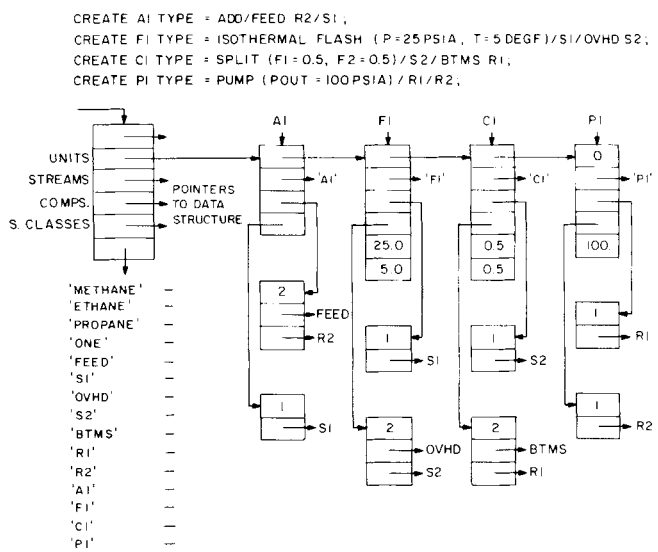


Fig. 15. Statement to create beads for process units.

The next four statements create three beads for each process unit: a bead to define the process unit with pointers to beads containing pointers to its feed and product streams, as illustrated in Figure 15. Pointers are also entered in the stream beads to the unit from which the stream originates and to which it is destined. Values for the unit parameters are also entered.

The last statement in the section defining the process specifies values of the feed stream variables. The entire data structure is shown in Figure 16. It is ready for use by simulation programs.

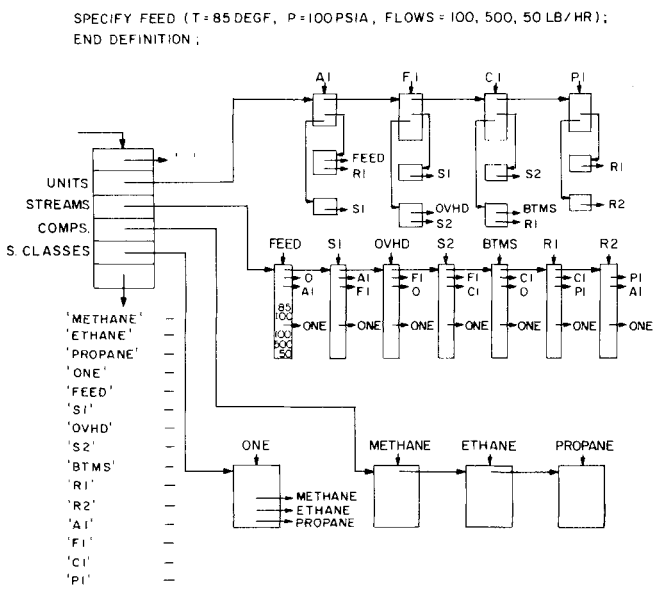


Fig. 16. Statement to specify the feed stream variables.

```
CALL INPUT
AI = POINT ('AI')
FI = POINT ('FI')          OBTAIN POINTERS TO
CI = POINT ('CI')          UNIT PARAMETERS BEADS
PI = POINT ('PI')          ESTABLISHED DURING READING
R2 = POINT ('R2')          OF INPUT DATA
CALL ASSUME (R2, TEMP, T, P, FLOW)     ASSUME VALUES OF
                                       RECYCLE STREAM
                                       VARIABLES
CALL MIX (AI)
CALL IFLSH (FI)            SIMULATE
CALL SPLIT (CI)            PROCESS
CALL PUMP (PI)             UNITS
CALL CONVG (R2, TEMP, &1)      IF NOT CONVERGED, GO TO 1
CALL REPORT                    ROUTINE TO PRINT RESULTS
CALL EXIT
END
```

Fig. 17. Problem oriented FORTRAN calling program.

The building blocks and their calculation order during simulation are given in the next section. First, initial assumed values for variables in recycle stream R2 are entered. Then, for each unit, the name of the simulation program to be used is given, as well as iteration parameters (for example, error tolerance). Finally, Wegstein's method is specified to converge the recycle calculations for R2.

The SIMULATE statement initiates simulation, and the PRINT statements cause information from the plex structure to be printed after simulation is completed.

```
PROCESS   FLASH WITH RECYCLE
COMPONENT  METHANE  16.04  -161.4...
COMPONENT  ETHANE   30.07  -88.6...
COMPONENT  PROPANE  44.09  -42.2...
STREAMCLASS ONE VAPLIQ METHANE ETHANE PROPANE
DECLARE  ONE FEED R2 SI OVHD S2 BTMS RI
UNIT  AI  ADD/FEED R2/SI/
UNIT  FI  ISOTHERMAL FLASH/SI/OVHD S2/25.0 5.0
UNIT  CI  SPLIT/S2/BTMS RI/0.5 0.5
UNIT  PI  PUMP/RI/R2/100.0
STREAM FEED 85.0 100.0 100.0 500.0 50.0
ENDDATA
```

Fig. 18. Data cards for problem-oriented calling program.

The principal advantages of the plex data structure are:

1. It promotes modularity—the building block routines require only a single argument, the pointer to the portion of the plex upon which they are to operate.
2. It provides flexibility, for example, classes for new types of streams are easily represented.
3. It permits more natural description of the attributes of models by intermixing real numbers, integer numbers, Boolean values, and strings of alphanumeric information in a single bead.
4. Modifications in the process configuration are easily made by deleting beads and creating new ones. As the concept of a problem being solved is altered, it is easy to change the data structure.
5. Storage is not wasted for large numbers of unused entries in predimensioned arrays.
6. There is no limitation on problem size imposed by predimensioned arrays. A data structure is created for each specific problem, and the sizes of each bead are tailored to the problem at hand.

The principal disadvantage is that execution time may be increased by the time required to access information from the plex. Preliminary experiments (Joseph et al. 1977), however, indicate that this penalty is not excessive.

PROBLEM ORIENTED CALLING PROGRAM

The use of the plex data structure does not require a problem oriented language. Figure 17 illustrates a FORTRAN program that calls upon subroutines to build the plex and carry out the simulation. A subroutine INPUT reads data cards of the type shown in Figure 18 and builds the plex. The data cards provide as much information to describe the process as the problem oriented language but in a more restrictive format and order.

The values of the pointers are retrieved from the pointer dictionary. Assumptions are entered for stream R2 followed by calls upon the building blocks. Finally, the REPORT routine prints the contents of the data structure.

A prime reason for including the problem oriented calling program in the design specifications is to assure that at least one version of each building block is available as a subroutine that can be called independently. It should be possible to utilize the building blocks on computers (for example, a real-time process control computer) that do not have facilities for translation or interpretation of the problem oriented language.

ROUTING

Developers of process simulation systems must consider the routing problem that exists when a building block calls upon building blocks, which in turn call upon other building blocks. The problem arises when a choice exists among building blocks, as occurs in the computation of physical properties. It has been referred to as the problem of route selection (Norris, 1965).

Consider the routing problem that arises in the computation of vapor-liquid equilibrium constants, K values. Figure 19 illustrates alternate calculation paths and the bold path corresponds to one possible route. To specify the route one must designate not only the options to be used by the routine KVALUE, but also the options for each routine that KVALUE calls upon, etc. The desired route could be indicated by a statement such as

DEFINE KROUTE 2(1,3), 3, 1(2)

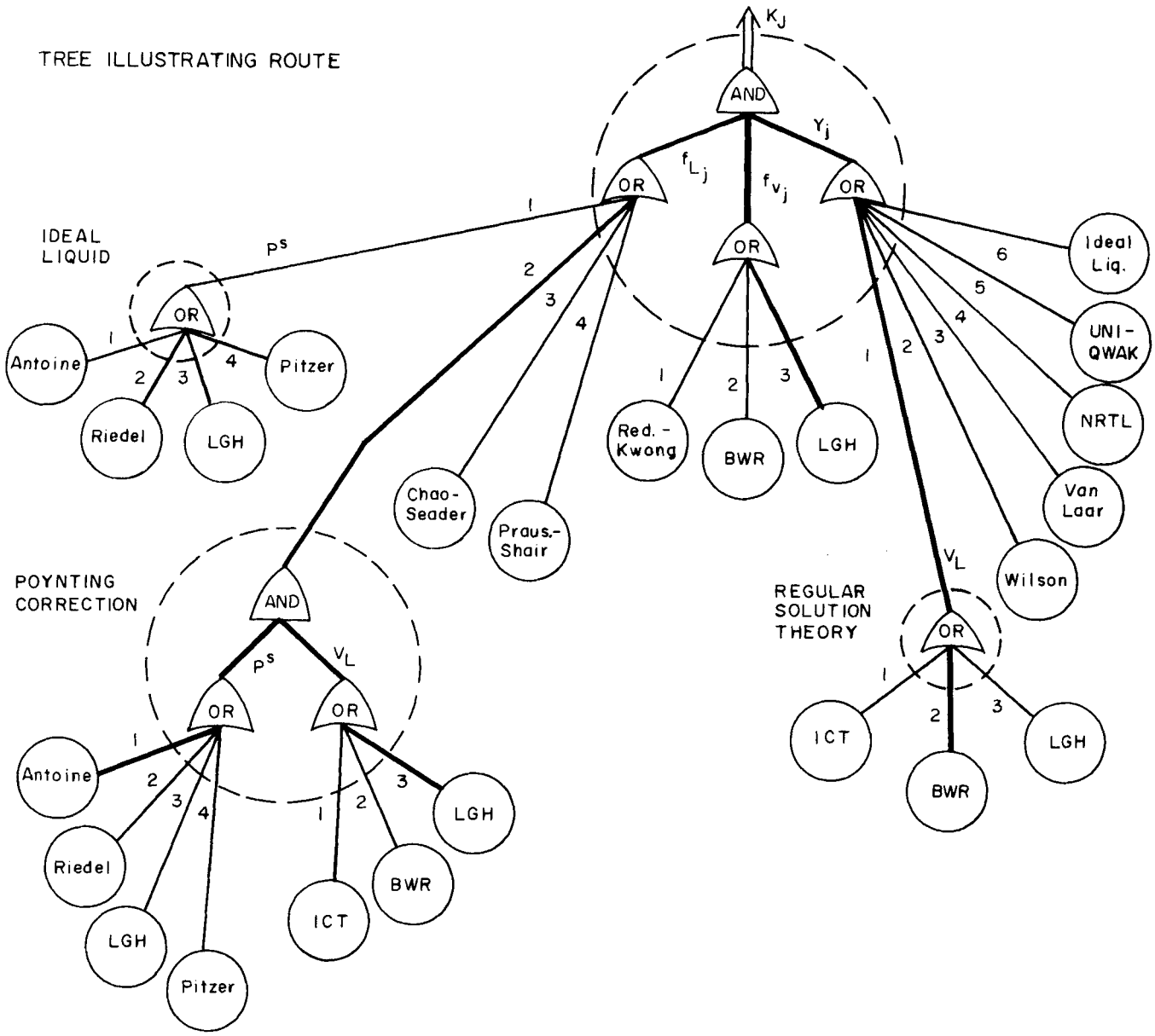


Fig. 19. Tree illustrating routes for K value computation.

This statement would build a plex structure for the route as shown in Figure 20. The pointer KROUTE would be included as an argument to KVALUE

KVAL(J) = KVALUE (other arguments . . ., KROUTE)

where J is the component index and KVAL(J) the result. The other arguments would specify the temperature, pressure, and compositions required. Each routine would refer to the plex to determine which subordinate routine to call and would transfer a portion of the routing plex to its subordinate routines.

The plex data structure is ideally suited for representing tree structures as arguments to the building blocks. It could be used to provide flexibility in the specification of routes for each building block. A separate route could be designated for each building block that calls upon other building blocks, where a choice exists. The subject of routing is treated in greater detail in a related paper (Seider et al. 1977).

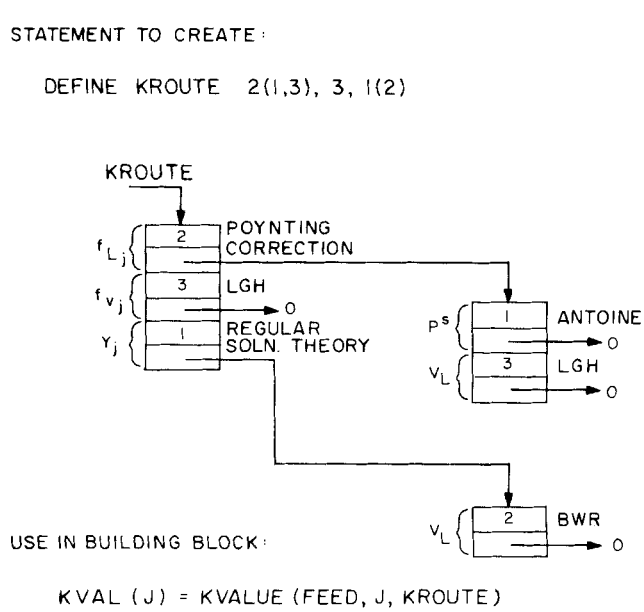


Fig. 20. Routing plex.

ACKNOWLEDGMENT

This work was supported in part by the Massachusetts Institute of Technology Energy Laboratory and the U.S. Energy Research and Development Administration.

LITERATURE CITED

Evans, L. B., and W. D. Seider, "The Requirements of An Advanced Computing System," *Chem. Eng. Progr.*, **72**, 6 (1976).

Fay, J. E., "A Prototype System for On-Line Computer-Aided Process Design of Heat Exchange Networks," Sc.D. thesis, Massachusetts Institute of Technology, Cambridge (1971).

Joseph, B., W. D. Seider, and L. B. Evans, "The Use of a Plex Data Structure for Process Simulation," submitted for publication to *Computers and Chemical Engineering* (1977).

Norris, R. C., "Estimating Physical Properties-Route Selection," *Chem. Eng. Progr.*, 61, No. 5, 96-101 (May, 1965).

Porter, J. H., "Computer-aided Process Design: An Exercise in Dynamic Man-Machine Communication," paper presented at the Annual Meeting of AIChE, Los Angeles (Dec., 1969).

Ross, D. T., "A Generalized Technique for Symbol Manipulation and Numerical Calculation," *Comm. ACM*, 4, No. 3, 147-150 (Mar., 1961).

Seider, W. D., B. Joseph, E. Wong, and L. B. Evans, "Routing of Calculation Methods in Process Simulation," Paper 30b 69th Annual Meeting of AIChE, Chicago, Ill. (Nov., 1976).

Manuscript received November 2, 1976; revision received June 2, and accepted June 17, 1977.

Vacuum-Spray Stripping of Sparingly Soluble Gases from Aqueous Solutions:

STUART G. SIMPSON

and

SCOTT LYNN

Department of Chemical Engineering
University of California
Berkeley, California 94720

Part I. Mass Transfer from Streams Issuing from Hydraulic Nozzles

Carbon dioxide, oxygen, Freon 114, and butane were stripped from deionized water, sodium chloride solutions, and freshly acidified seawater in a vacuum-spray chamber using three types of hydraulic nozzle. Purely physical desorption was found for each gas/liquid system.

For both conical-spray and fan-jet nozzles, the bulk of the gas was shown to desorb from the thin liquid sheet issuing from the nozzle, that is, before droplet formation occurs. Behavior was explained semiquantitatively by a mathematical model of the diffusion in the sheet. This model was also shown to predict well the desorption from the laminar liquid sheets formed by jet impingement nozzles. For turbulent flow in jet impingement nozzles, the fit of the model was again semiquantitative.

SCOPE

Oxygen and carbon dioxide must frequently be removed from the feed to desalination plants to minimize corrosion and scaling, respectively. Refrigerant gases must be removed from both the product water and effluent brine streams of some freeze-desalination processes. Vacuum stripping may be used in these cases to avoid the excessive heat load associated with steam stripping, since only one or two equilibrium stages are required for nearly complete removal of a sparingly soluble gas. The purpose of this study was to analyze the mass transfer behavior of sprays during vacuum desorption of sparingly soluble gases. Only aqueous solutions were studied in this work. However, the general principles would be expected to apply equally well to nonaqueous systems.

Of the six major categories of spraying devices (Lapple et al., 1967), hydraulic nozzles are the only type well suited for vacuum-spray stripping, since they use energy relatively efficiently and contain no moving parts. The major types of hydraulic nozzle are fan jet, centrifugal (swirl), and jet impingement. In all three types, the liquid leaves the nozzle as a thin, coherent sheet which then breaks up to form a spray.

Hydraulic nozzles are frequently characterized by the drop size which they produce. The implication is that the important mass transfer takes place after drop formation is completed. However, a number of workers have recognized that rapid mass transfer can occur close to the nozzle before and during drop formation.

It is important to the design of compact, efficient vacuum-spray equipment to determine the region of the spray in which mass transfer is most rapid. One of the goals of this paper was to make that determination by comparing experimental data with contrasting mathematical models. In a second paper a method of designing the equipment for a particular stripping duty will be presented.

Correspondence concerning this paper should be addressed to Scott Lynn. Stuart G. Simpson is with Union Carbide Corporation, Tarrytown, New York.