A THESIS
entitled

MATHEMATICAL MODELLING OF GAS FLOW NETWORKS IN PELLET INDURATION SYSTEMS

by

MUHAMMAD AFZAL
MSc (App Maths), MSc (Software Engng)

Submitted in partial fulfilment of the
requirements for the award of the

DEGREE OF DOCTOR OF PHILOSOPHY

of the

UNIVERSITY OF GREENWICH

This research program is sponsored by
Government of Pakistan and University of Greenwich

Centre for Numerical Modelling and Process Analysis
School of Mathematics, Statistics and Computing
Faculty of Technology
University of Greenwich
LONDON SE18 6PF
United Kingdom

JUNE 1994

Abstract

The objective of this research is to develop a simulation software tool, GASFLO, which should evaluate pressure, flow and temperature distributions of process gas in pellet induration system networks. Pellet induration systems are complex industrial systems composed of heterogenous components. The magnitude of gas through leaks i.e. the air entering or leaving the system from the points other than the known exits, is substantial and it adversely effects the performance of induration process. These leaks are very difficult to measure because of the hostile environment in the plant. The modelling of such industrial systems requires a notable amount of experimentation so the tool has been designed to enable the user modeller to change the component models and solution algorithms easily.

The conventional methods for flow network simulation are based on process centred approach, mostly composed of homogeneous components. For ease of computation, the non-pipe elements are modelled with an approximate linear or non-linear generic equation, whose coefficients can simulate different states of the element. The resulting set of non-linear equations is linearised and solved simultaneously using some iterative method. By contrast, GASFLO is based on device centred or unit based approach, and uses a two level hierarchical solution algorithm. The pellet induration system network is first idealised into a connected graph of streams (sets of serially connected components) and nodes. At the top or coordination level the flow and pressure distributions satisfying the Kirchhoff's laws are evaluated for the connected graph. At the lower or component level the exact mathematical models of components are computed, in order of their occurrence in respective streams, using coordination variables as parameters. The converged flows are used for the temperature computation. The solution algorithm requires partitioning of the connected graph into forest and coforest structures, for which secondary algorithms have been developed using specific heuristics relevant to the pellet induration systems. The rigorous application of software engineering techniques for the design and implementation of software, enabled the resolution of the complexity of the modelled system, embedded the characteristics of 'quality software' into the resulting code and benefits from object orientation, even though it is implemented in standard FORTRAN 77.

GASFLO predicted results are in a good agreement with the measured results, it has been validated for a real life pellet induration system. It has been applied to simulate several practical scenarios, like addition of extra wind boxes to the zones and to determine how the plant production can be increased by certain ratio, such simulations were not feasible otherwise. GASFLO takes less than a minute to simulate a real-life pellet induration system on a 486 PC. The combined simulation with an other software tool, INDSYS, which evaluates the heat distribution in the solids, is also feasible.

Acknowledgements

I am deeply indebted to my first supervisor, Prof Mark Cross, for his invaluable assistance, expert guidance and constant encouragement; during the whole course of my study, he has been a great source of support and motivation.

I am thankful to my second supervisor, Prof Martin Everett, and all my teachers especially Prof B Knight, Drs D Cowell, C H Lai, M K Patel, K Pericleous, for their contribution to the different aspects of the present work. I acknowledge the useful discussions with Jem Pearce (during his stay at Greenwich) which lead to some algorithmic improvements and generality.

My thanks are due to my colleagues; Jixin, Miltas, M Hughes, J Ewer, Drs P Chow, A Chan, M Agha and C Bailey who made my stay at university a memorable time; and to my friends; N M Malik, Riaz Malik, Yaqoub, Yameen, Yasser, Zaheer and Shakeel whose brotherly and affectionate attitude provided me home-like comfort during this period.

I am grateful to my brothers Ajmal and Ajaib, who shouldered all the family responsibilities and my family and children for their understanding and patience for my long absence from home. My thanks are also due to my friends whose letters and calls provided a constant encouragement.

Finally, I would profoundly acknowledge my sponsors' support: (i) Government of Pakistan for providing me subsistence at U K and maintenance to my family at Pakistan, (ii) University of Greenwich for providing extra financial support during this course of study. I am thankful to my employer at Pakistan for constant encouragement and chase which enabled me to conclude this research within their allowed time schedule.

To the memory of my father

MUHAMMAD ASHRAF
(1928-16/5/1982)

Who didn't live to realize his greatest ambition -
that of witnessing me achieving this degree

Table of Contents

Chapter 1
Introduction

Simulation of fluid flow networks using mathematical or computer models is widespread. The ever improving performance/cost ratios and decreasing hardware prices has motivated their increased usage and it also promises even wider use in future. The flow network models can be classified as: static (i.e. steady state) models which are independent of time; and dynamic (i.e. transient or unsteady state) models where the system variables vary with time. Both of these categories have their own specific domains of use, different formulations and development difficulty levels. Steady state models are easier to develop and used for network analysis. Further, these can be extended for design, optimization and model based process control systems. The static models are comprised of algebraic equations. The dynamic models are based on time dependent ordinary and partial differential equations and algebraic equations. These are used to analyze the unsteady behaviour of networks i.e. how the introduced disturbances can propagate with time in the network, and for optimization, operation and planning purposes. The required initial conditions which are sometimes provided by the respective static models of the network. In fact, for all these models, the steady state models play a key role, and chronologically, are the first to be developed, later these are progressively extended for design and other purposes. In this research we will focus on the development of a steady state model for the simulation of pellet induration system networks (which will be discussed in detail in Chapter 2).

All flow networks, whether electric, water, natural gas, mine ventilation or pellet induration system networks are analogous to each other. They must obey the Kirchhoff's laws, namely: Kirchhoff's Current Law (KCL) i.e. the net flow entering to a node is zero; and Kirchhoff's Voltage Law (KVL) i.e. algebraic sum of voltage or pressure drop across any

closed loop is zero. Because of this common basis, the formulation methods and algorithms for solution are also shared among them. Hamam and Brameller 1971 applied the electrical network solution algorithm (called a hybrid method) for the computation of natural gas pipe networks. Coulbeck and Orr 1990 have discussed the similarities between the water and power distribution networks. Nielsen 1989 has argued that, by changing medium specific parameters, his program can solve water and natural gas networks. Similarly, the method proposed by Yevdokimov 1969 is equally applicable to electricity, water, natural gas and mine ventilation networks. So, the terms 'pipe network' or simply 'network' in subsequent pages, refers to any fluid flow network, unless it is specifically mentioned.

In this chapter, the existing methods for steady state simulation of networks will be discussed briefly in Section 1.1. In Section 1.2 the objectives of the present research will be described, in Section 1.3 the other applicable literature will be mentioned which somehow influenced the proposed development. In Section 1.4, the differences between the existing methods and the proposed method will be described and the means to achieve the stated objectives will be discussed; and chapter will be concluded in Section 1.5.

1.1 Brief Review of (Static) Network Simulation

The steady state network simulation is a mature and well established field. The space limitation doesn't allow the complete coverage of any of the existing methods or even the mention of all of these methods. These methods have been well covered in texts like Deo 1974 (electrical networks), Jeppson 1976 (water networks), Osiadacz 1987 (natural gas networks) and Bhave 1991 (water networks) and thoroughly reviewed by Fincham 1971, Wood and Rayes 1981, Nielsen 1989, Goldwater and Fincham 1981, Moll and Lowndes 1992.

The network simulation can be regarded, as formed of two parts; first how the network is formulated into a set of mathematical equations, and then how these equations are solved.

This separation, though quite significant is not that well observed in the literature, sometimes the same names are used for solution methods as for their formulation. In following subsections we review the main formulation and solution methods.

1.1.1 Network Formulations :

Using the water networks analogy, any n node (including one source), p pipe general network will have $l = p - n + 1$ fundamental loops. There will be $n - 1$ independent continuity or flow balance equations and l loop or energy balance equations. The network solution provides; the pressure or head at these (load) nodes with respect to the given pressure at source (or reference) node, and flow in each of these p pipes for known node loads (or demands). The flow through the pipe is related to pressure drop across the pipe by a well defined equation which is specific to the nature of the network, flowing medium, and flow regime. This equation, in fact, relates the pressure at both ends of the pipe to its flow, so knowing any two of these, the third can be computed.

The network equations are written using Kirchhoff's two laws, as every fluid flow network should satisfy these two conservation laws. Secondly, for solvability reasons, there should be the same number of (linearly independent) equations as there are variables to be computed. In general, the networks are solved for the given values of the node loads and reference node pressure as boundary conditions, but some authors can deal with mixed boundary conditions.

There are three most commonly used formulations. These are dependent upon the availability of initial values and the variables intended to be computed.

a) Nodal Formulation. When initial values for load node pressures are known, then using Kirchhoff's first or current law (i.e. the net flow entering a node is zero) the nodal equations are written. These will be $n-1$ in number because the nth equation will be linear combination of the others. Initially, these equations are in flows, but using the

pressure drop - flow relationship, these can be transformed into load node pressures or heads (Osiadacz 1987, Jeppson 1976 referred them as H equations). The solution to these n - 1 equations provides the node pressures which can further provide the pipe flows. The solution methods based on this formulation are also called as nodal methods.

b) Loop Formulation. When initial values for pipe flows, satisfying nodal equations are available then using Kirchhoff's second or voltage law, loop equations are written. According to this law the algebraic sum of pressure drop across any closed loop in the network will be zero. The purpose is to find pipe flows which satisfy the loop equations, so these are usually written in terms of Q + ΔQ form, where Q is pipe flow and ΔQ is the corrective flow for respective loop. These are sometimes called as ΔQ equations (Jeppson 1976). Obviously, the number of equations are lesser than the nodal formulation, but these are more complex. This formulation also requires the information about loops in the network.

c) Nodal-Loop (Full Equation) Formulation. This formulation frequently appears in recent literature (e.g. Ormsbee and Wood 1986, Mucharam and Adewumi 1990, Boulos and Wood 1991 and Boulos et al 1992). According to it, all n - 1 nodal (or flow balance) equations and l $(=p-n+1)$ loop equations, hence total of p equations are written and solved simultaneously. Jeppson 1976 writes these equations in terms of pipe flows and calls them as the Q equations. This formulation increases the overall number of equations and also requires the information about the loops, but has been widely used for water networks for other advantages like its extension to compute the exact values of (design, operation and calibration) parameters explicitly.

In classical literature, the solution methods are categorised as Nodal and Loop methods. As described by Boyne 1970, for Nodal methods the Kirchhoff's (second law) loop equations are always satisfied whereas nodal equations are not, so starting from the assumed nodal pressures (such that they satisfy the loop equations), these pressures are systematically

amended until nodal equations are satisfied. Correspondingly, in Loop methods the Kirchhoff's (first law) nodal equations are always satisfied and loop equations are not, so the pipe flows are systematically amended, such that the loop equations are satisfied. This systematic adjustment is carried out using some well defined solution method, either one of those mentioned in next section or some other.

1.1.2 Network Solution Methods :

Real life networks give rise to very large sets of equations which cannot be solved analytically or manually and are solved using computers. The following three solution methods have been widely used for network solution. Since these have been thoroughly covered in literature and in the mentioned texts so for completion sake these are briefly described here.

i. Hardy Cross Method: It is the oldest, simplest and empirical method. It was proposed by Hardy Cross 1936. It suits well to manual calculation and easier to program. Because of its simplicity Bhave 1986, recommends the use of Hardy Cross method than its counterparts. Bhave mentions that the overall efficiency of Hardy Cross method is comparable to other methods, though it takes more iterations to converge, because Hardy Cross method is simple, it takes much less time to perform one iteration than the time taken by an iteration of the linear theory or Newton's methods. It can solve all the three formulations but is more often used for loop formulation. Fincham 1971 has described an efficient variant of original Hardy Cross method, which is like Newton Raphson applied to single equation, it has better convergence and needs lesser storage. Boulos 1989 has denounced its use for being a non-matrix method and poor convergence properties for large networks.

ii. Newton-Raphson or Newton's Method: According to this method the non-linearity of equations is resolved using Taylors series expansion. The mathematical derivation of Newton's method is well covered in numerical analysis texts (e.g. Burden and Faires 1989); Osiadacz 1987 has also discussed its development in scalar (single equation)

and vector (i.e. matrix) forms. It is generally applied in matrix form. It requires the evaluation of Jacobian matrix and is sensitive to initial conditions. For a good initial guess it has a fast convergence, however, the accomplishment of this guess for a large equation set is difficult. It has been widely applied especially to the full equation formulation. Ormsbee and Wood 1986, Boulos and Wood 1991 and Boulos et al 1992 have applied it to solve water networks. Boulos 1989 has also discussed different variations of Newton's method for improved computational speed, e.g. modified Newton method, where the same Jacobian matrix is used for certain successive iterations and then re-evaluated; this saves on the Jacobian evaluation computational load which is significant.

iii. Linear theory method: This has been proposed by Wood and Charles 1972. According to this method, the pipe equation which is quadratic in flow and source for all non-linearity in the system, is linearised explicitly. It is re-written in terms of another constant which contains previous iteration flow values. This linearised pipe equation should be used for respective formulation. More often the full set is solved for flows. Using these flows, the constant is re-evaluated and the iteration repeated. The iterative procedure is continued until the converged flows are achieved. Wood and Charles 1972 have compared it to Hardy Cross and Newton Raphson methods and found that it took minimum iterations to converge. Another advantage is that it does not need initial values, like other methods, instead these can be computed by the program itself. They have noticed that for successive iterations, after achieving the converged values sufficiently close to the true value, may oscillate and suggested to reuse the mean of two previous iteration values.

Hardy Cross is a non-matrix, easy to apply method but takes more iterations to converge and has poor convergence characteristics for large networks. Newton's and linear theory methods are more suited to matrix notation, take less iterations to converge, though time taken per iteration is much larger than for an iteration of Hardy Cross method. The computation using Hardy Cross and Newton methods needs initial flow values, convergence

of Newton's method is sensitive to the initial guess. Whereas the linear theory method does not need initial values, but its converged solution oscillates about the real solution. Nielsen 1989 has suggested to use linear theory method for first iteration and Newton's method for subsequent iterations thus combining the efficiencies of both methods. This approach has been used by Hansen 1988 and Hansen et al 1991.

1.1.3 Some other Formulations and Solution Methods :

The development of these methods requires thorough research, their realization into computer programs, validation of these programs and then comparison to already existing methods, to see how efficient they are. This requires a significant amount of human and financial investment, which only few groups or companies can afford. This has also encouraged university-industry collaboration. Because of the challenging nature and potential gains, these proven efficient methods frequently remain inaccessible to the public domain. The delay in publication in the public domain could be due to the immaturity of the method concerned but it is mostly intentional for commercial and competition reasons.

In the following some methods (from the recent literature) are stated, which are either the extensions to previously stated ones, or have a completely new approach.

Hansen 1988, Nielsen 1989, and Hansen et al 1991, have suggested to formulate the network, using $n-1$ nodal continuity equations and p pipe equations, thus forming an extended set of $p+n-1$ equations to solve $n-1$ load node pressures and p pipe flows directly. They used the linear theory method in the first iteration and Newton's method in subsequent iterations. Their programs solve water as well as natural gas distribution networks, They have also exploited the network topology, using graph theoretic techniques to reduce the computational load by lumping branched subnetwork demands as node loads to the looped network, which is solved iteratively and the final node pressures being transmitted to branched subnetworks for their pressure distribution computation.

British Gas, being a pioneer in the simulation of natural gas networks, have significantly refined these methods for natural gas transmission and distribution networks. Fincham 1971 and Goldwater and Fincham 1981 give a good review of programs developed for natural gas and/or at British Gas and the mathematical basis for the development of simulation programs. The British Gas network is structured hierarchically; the higher level or transmission network transports gas from source points to the distribution points, from where the lower level or distribution network distributes it to the end users. To save energy losses the transmission networks are maintained at high pressure (Batey et al 1961) and to minimise leaks and maintenance the distribution networks are operated at low pressures (Ellis et al 1987). The transmission networks are comparatively small having 00's of pipes, simulated using non-pipe elements (like compressors and regulators etc) and modelled for dynamic simulation; accordingly the nodal formulation and Newton type solver is used. On the other hand, the distribution networks contains 000's of pipes, and are modelled for steady state simulation. Here the loop formulation is used (since it has less storage overheads) and solved using Hardy Cross method. For loop generation improved algorithms are used which produce loops with minimal overlap (Fincham and Goodwin 1988). In this span of 30+ years of simulation research, British Gas has produced a number of programs like PAN (Program for Analysis of Networks), COSP (COmputer Scheduling Program), OSCAR, FALCON, OTTO and MINOS etc, for all practical purposes from analysis, design, control, operation, strategic planning, to scheduling. Details are covered in Fincham 1971, Goldwater et al 1976, Fincham and Goldwater 1979, Goldwater and Fincham 1981, Francis 1982 and Wilson et al 1986.

The use of direct methods rather than iterative methods was mentioned by Boyne 1970, but he stated the limitations of computer memory with problem size. Now, technological advances in hardware, has provided much larger memory size even on PCs, so this case is taken up by Gomasta and Devi 1989. They developed a graph theoretic approach, introduced a fictitious node and as many fictitious pipes as were the source and load nodes, partitioned this new connected network into tree and cotree structures, and evaluated the cutset and circuit matrices. By definition, *cutset*, is set of those edges of the connected graph

which, when torn, break the graph into two separate disjoint graphs. They wrote down the full equation formulation for this augmented network and used topological properties of the network and instead of using any of the above stated relaxation (Hardy Cross, Newton's or Linear theory) methods they directly evaluated the pipe flows and load node pressures. They have mentioned that their approach always give converged results, is efficient and can simulate large water distribution networks on a PC.

Deo 1974, has introduced the concept of node admittance matrix (i.e. the nodal matrix transformed using pipe equations such that its elements are the pipe conductances for respective incident pipes) for electrical networks. He demonstrated that clusters of serially connected components could be abstracted by one component offering the same equivalent resistance (or conductance). This node admittance concept has been extended by Kiuchi 1991 for natural gas networks. Kiuchi wrote down the node admittance matrix and, using assumed load node pressures, he evaluated node pressures and pipe flows with the help of a SOR (Successive Over Relaxation) type scheme and compared the results with the Hardy Cross method for 4 networks. His approach is less sensitive to the initial guess and gives better convergence, but it is dependent on the used relaxation parameter for which he has specified a recommended range.

The Critical Path Method (CPM) is an approach well used in operational research and activity scheduling, where a whole system is reduced to a connected graph of nodes and edges, where each edge represents an activity and assigned a weight, and nodes represent the time events. Wang 1982 has used CPM for the steady state analysis of mine ventilation networks. In his formulation node and edge had the same physical meanings of junction and airway respectively. However, he assigned the pressure drop across an airway as the weight of the respective edge and total pressure drop from source to the respective node as the weight of that node. He applied the graph theoretic tools like, spanning trees and cutsets, and successfully simulated a multi source, multi sink mine ventilation network having multiple number of fans, for a controlled flow (where flows in some airways are required to have some

pre-set values) environment. He discussed how this strategy could be extended for optimization purposes.

The graph theoretic concepts and properties of the resulting cutset and circuit matrices have been exploited by many authors (e.g. Osiadacz 1987, Gomasta and Devi 1989). Yevdokimov 1969 has used the orthogonality of these matrices to generate the final equation set. He wrote equations in such a form that out of the *3p* variables (i.e. flow, resistance and pressure drop across each of the *p* pipes) any *2p* variables can be computed using the remaining *p* values as initial conditions. Yevdokimov has provided the algorithms to generate these matrices and discussed advantages and shortcomings of Hardy Cross and Newton's methods and proposed another *coordinated gradient* method for computation of mine ventilation networks.

Wood and Rayes 1981 have reviewed the existing five algorithms for water networks. Three of these called PATH (single path adjustment), S-PATH (Simultaneous path adjustment) and LINEAR (flow adjustment) are based on loop equations, whereas the remaining two named NODE (single node adjustment) and S-NODE (Simultaneous node adjustment) are based on nodal equations. They programmed and tested these algorithms on a big database of available hydraulic networks. They tested for 60 networks of under 100 pipes and 31 network of over 100 pipes. These networks included pumps but not other non-pipe components like check valves and pressure regulating valves etc as these required special procedures for some methods. They found that LINEAR and S-PATH gave the best performance. LINEAR is the application of the Linear theory method using a full equation formulation (Wood and Charles 1972); and S-PATH is due to Epp and Fowler 1970 which is the application of an improved version of Newton's method to loop formulation.

Osiadacz and Pienkosz 1988 have described and compared the four most commonly used methods for steady state simulation of natural gas networks. Two of them are based on loop formulation, named as 'loop method' and 'loop-node method'; whereas the remaining two use nodal formulation and called as 'node method' and 'node-loop method'. The resulting

equations are solved using Newton's (multi-dimensional) method for all of them. The authors found that on the basis of computational time performance, 'loop-node method' is the most efficient. The main reasons for this efficiency are: the generated matrices for this method are sparser than those generated by the others, and secondly, the order of different steps of computation within the solution algorithm is optimal. British Gas uses the same loop-node method for computation of its high-pressure transmission networks with non-pipe components (Fincham and Goodwin 1986).

Lowndes and Weimin 1988 have given a good review of methods used for optimization of mine ventilation networks. Whereas, Moll and Lowndes 1992 have discussed the formulation of mine ventilation networks, and application of either of Hardy Cross, Newton's or Linear Theory methods to solve the formulated full equation model.

1.2 Objectives of Present Research

The main objective of the present research was to develop a steady state simulation software tool for pellet induration system networks. Pellet induration systems are a key component of the iron and steel making industry, and these will be described in detail in Chapter 2.

This proposed software tool should:

- determine the flow, pressure and temperature distributions in the pellet induration systems networks,
- communicate with other already existing software tools, used for the simulation of different aspects of the induration process,
- act as a workbench for the user modeller; enable him/her to refine or change the mathematical models of the system components, since the field is relatively un-explored, the exact nature of flow is not known and also there does not exist any such

models, thus the components' (mathematical) models would require significant experimentation,

- assist the training of plant operators, analyze networks for plant engineers and provide guidelines for the managers of the induration systems,
- be extensible to accommodate the needs of all concerned, from the developer to the end user (including operators, plant engineers and system managers). These requirements would also change with time and by the use of this tool. For example it should be user friendly, easier to use and display results in the format the operators and plant engineers are accustomed to.
- have flexible architecture to facilitate: the developer to improve the computation algorithms; the user modeller to program and link his own software modules for new component entities of his interest to be appended to the tool for simulation; the end user to add or delete any instance of the existing modelled entities at run time through input files; the addition of further optimization and other required modules to enhance its functionality for planning and other purposes,
- work on high-end PC compatible, especially 486 machines - since these are powerful enough, widely and readily available in the induration industry,
- be fast, robust and have qualities proposed for a 'quality software' by the software engineering community especially the low maintenance costs i.e. ease to accommodate the required unforseen changes at later stages, of the developed software.

With all the above stated qualities, the architecture of the software tool, will be flexible enough to extend it to a generic code which might simulate any fluid flow network as claimed by Yevdokimov 1969 and others.

1.3 Other Applicable Literature

To achieve the objectives stated in the last section, and literature survey revealed that it is possible to attain these goals. In this section, a few of the approaches are mentioned which directly or indirectly influenced the present research.

In abstract, the mathematical model for steady state simulation of a network, is a set of non-linear algebraic equations whose size is proportional to the size of the network. These are coupled equations, the variables computed by one equation are used as parameters for the computation of others, and a simultaneous solution of this equation set is sought. For our case, the variables are flows, pressures and temperatures of the process gas, distributed spatially in the network.

Sargent 1978 has proposed a method, according to which, the original set of equations is partitioned into smaller subsets, and each of these subsets are solved separately. He represented these systems of equations by a directed graph, whose nodes correspond to the respective subsets and edges represent the communication between the nodes. The edges coming into the node are the information (or parameters) required for the computation and outgoing edges are the output produced by the node i.e. values of the variables computed inside the node. The resulting graph may contain closed (directed) loops, showing that the inputs of a node, say 'A', are coming from another node 'B' which used A's output (directly or indirectly through some other nodes). Sargent has given algorithms to resolve these loops (by tearing edges to reduce the original graph into an acyclic graph) and specified the criteria for optimal solution. He proposed that the solution algorithm should have two iteration cycles, in the inner iteration cycle the respective nodes be solved treating all incoming variables (or parameters) as constants, whereas in the outer iteration cycle the variables corresponding to the torn edges be fed-back to the respective nodes which require them as input. The iterations are carried out until the variables corresponding to the torn edges converge. This is an efficient method, it reduces the problem size, requires less storage and is computationally efficient. He hoped that packages could be developed where these nodes will correspond to

the subprograms having respective sets of the equations and the edges would then relate to the argument lists being used by these subprograms.

Sargent's method has emerged as the well known *sequential modular approach*, where the original set of equations is partitioned into smaller subsets or modules and these modules are computed sequentially. Motard and Westerberg 1981, have mentioned simulation/modelling packages like FLOWTRAN, CONCEPT and PACER, for chemical engineering which are developed using a sequential modular approach. These packages resolve the to be modelled chemical plant as a flowsheet, where each block or node actually represents a physical unit, and solves its relevant mathematical model. Montagna and Iribarren 1988a, have given algorithms to evaluate the optimal sequence for the computation of these nodes/modules. Further application of these algorithms to the chemical plants' simulation and other flowsheeting programs has been discussed by Montagna and Iribarren 1988b.

The general purpose simulation package for process analysis and control of chemical plants, SPEEDUP, has been based on an extended version of sequential modular approach named as *equation oriented approach*, which is even more flexible, as each module/node can be solved for any of its variables by specifying others as inputs. SPEEDUP can simulate steady state as well as dynamic behaviour, and can be used for control, operation and optimization of chemical plants (see Perkins et al 1987 and Bogle and Pantelides 1988).

Livny and Melman 1982 have described their WEizmann Network SIMulation (WENSIM) package, which is initially intended for the solution of queuing and scheduling problems on computer networks, however it is claimed to be flexible enough to be extended for the solution of industrial processes. In WENSIM, the computer network is represented by directed graph of nodes and edges, where nodes are the processing units and the edges represent data/signal information. The processing units are independent, highly modular, have a uniform interface, and can only be activated by the data passed on through edges. The user modeller can define the network connectivity, and can write his own processing units or modules conforming to the pre-set interface (necessary for communication with other already

existing modules) and embedding all controls required for the computation inside the respective module (to make it an independent processing unit).

Babrow 1984 has categorised the approaches used for process analysis and quality physics into two classes: *process centred* and *device centred* (which will be discussed in detail in section 2.3.2). According to the *process centred approach*, the attention is focused on the whole system and its behaviour is analyzed. For example all existing simulation packages for fluid flow networks use this approach, and concentrate on the whole network and study the flowing medium properties with respect to the whole system. Whereas in the *device centred approach*, the attention is centred on the behaviour of individual components of the system, since the whole system is composed of these primitives. The device centred approach has been explained by DeKleer 1984. In fact, he has implemented it in the form of an electronic circuit simulation package, EQUAL, where all individual circuit elements are modelled as units/modules. The user input circuit schematic is validated and resolved by the program in the form of a 'mechanism' or directed graph, where each node corresponds to the component model and the edge conveys the control information or variables required to compute the respective node. The order of computation of component models is sorted out automatically by the program and it is unidirectional i.e. on the completion of a component computation the computed variables are fed to the next connected component and its computation is invoked. The finally achieved solution should satisfy Kirchhoff's current and voltage laws as well as individual component models. DeKleer has claimed that the described algorithms are generic and can be used for any other network by replacing the respective equivalents of current, resistance and voltage etc.

The device centred approach has further been implemented by Boghosian 1990 and Chandra et al 1992 in their modern state-of-the-art computer packages. Boghosian has stressed that harnessing of the available terra-flop raw computational power, to solve the previously unthinkable and scientifically challenging problems like simulating 'appropriate physics', is only possible if the shift in basic programming and modelling methodology is made. He has proposed a data parallel programming methodology for massively parallel Connection

Machine CM-2. According to this methodology, finite difference or finite element grid of the actual modelled domain is mapped onto the configurable (2-D or 3-D) array of processors, and the connectivity of original grid is also accordingly mapped. Each element of a CFD problem domain is simulated by the respective processor. The grid related local data resides in the processor's memory, whereas the data common, and required by other processors for computation is declared in the form of 'parallel variables'. The implementation of numerical algorithms and data structures is straightforward, for example, the finite difference application of CFD problem does not reduce to seven diagonal banded matrix, instead it is a single linear equation with six coefficients (the parallel variables from neighbouring grids). A 'context flag' is assigned to each processor, which can deactivate its computation explicitly, if required. Boghosian has shown results of some really challenging problems from CFD, computational physics and biological sciences domains. The inability of process centred approach to resolve the complexity of problems like 'turbulence modelling' is also evident from the projected computational times, quoted by Jones 1993a, for example, an implementation of k-ε turbulence model for channel flow with refined mesh takes 250 hours on CRAY X-MP super computer, which confirms the need for strategic shift in modelling methodology, as emphasised by Boghosian.

An 'Interacting Object Process Model' (IOPM), to simulate real life physical processes and structural systems, is described by Chandra et al 1992. For the implementation of device centred approach into IOPM, they benefited from the widely propagated object oriented technology. They defined the system as composed of hierarchical objects, called *holons*, i.e. each holon simultaneously behaves as a whole (when considered individually along with its children), and as a part (from the viewpoint of its parent or whole system). Like artificial intelligence, each object can be interrogated, can store its related temporal information, and can respond with its shallow (or rule based) or deep (model based) knowledge, as desired by the user. The extensibility of the knowledge base for different views of objects, enables the modeller to refine the process incrementally. The authors have illustrated the functionality of IOPM, by applying it to solid and continuum mechanics systems to study their transient behaviour.

For fluid flow networks; Turner et al 1982 and Turner and Rainbow 1983 have reported their package NAIAD for the simulation of natural gas transmission networks. This is based on similar lines; all components have been modelled as separate modules, and each one solves for pressure, temperature and flows by solving mass, momentum and energy conservation equations. Unfortunately these papers do not provide any information about the solution algorithm or implementation. Another package SIROGAS, by the same group of developers, for steady state and transient simulation of natural gas networks, is also developed using device centred approach. Turner and Simonson 1985 describe a whole network as composed of two (pipe and node-like) hierarchical components types, which include all sorts of hydraulic and natural gas network components. They have illustrated the functionality of the package by simulating a compressor station.

Several packages have been mentioned by IFAC'87 (International Federation for Automatic Control) proceedings, in context of computer aided process design, operation, control and automation of chemical plants, which are based on device centred approach or its variants. For example, Marquardt et al 1987 has described the structure and working of their dynamic flowsheet simulator, DIVA.

The international conferences arranged by CACHE (Computer Aids for CHemical Engineering education) and FOCAPD (Foundations Of Computer-Aided Process Design) has also made similar recommendations for the next generation of computer programs for model based process control systems. McRae 1990 has asserted that the solution of large scale flowsheet problem on advanced computer architecture will require new algorithms for optimal performance and provided guidelines for the design of such new algorithms. He has pointed out that the presently used approach based on parallelization of serial algorithms will severely limit the size of the problem and will not be an optimal alternative.

1.4 Pellet Induration System Modelling and Solution Scheme

The literature cited in the last section, suggests that, to benefit from the advanced computability, in terms of hardware, software and numerical methods, the device centred approach should be chosen rather than process centred approach. Apart from these recommendations, there are some specific requirements from the perspective of pellet induration system network which lead to this choice. For this we first, briefly review the classical network solution methods and then discuss the proposed method based on device centred approach.

1.4.1 Classical Methods for Network Simulation :

The classical methods for the solution of fluid flow networks (discussed in Section 1.1) have following salient features:

- The formulation and solution methods are based on process centred approach,
- The network components' equations are usually transformed into a consistent (linear or quadratic) generic equation for computational ease - i.e. the components are treated as if they are mathematically homogenous (e.g. Osiadacz 1988 treats all non-pipe components as three termed linear equation whose coefficients can produce the desired effect of constant upstream or downstream pressure or constant flow; Boulos and Altman 1991 have assumed a quadratic equation to simulate non-pipe components),
- The relaxation methods, Hardy Cross, Newton's, and Linear Theory, linearize the original non-linear equations and use iteration to resolve this non-linearity. Whereas the graph theoretic methods take benefit of the network topology, cutset and circuit matrices and reduce the equations to a linear set. However, computation in both of these categories is centralised, carried out in matrix form (except in case of Hardy Cross method). The user or modeller has neither any control on the computation nor any choice for the solution method, for an individual component,

- The matrix methods, have many advantages, but are complex and have a penalty for extra storage requirements. For complexity reasons Bhave 1986 has preferred to use Hardy Cross method. Also the extra storage requirements are more pronounced for large size networks and sometimes even dictate the choice between loop and nodal methods (Fincham and Goodwin 1988),
- In general, the node loads or demands are known, and while solving the network, these are used as parameters.

1.4.2 Proposed Method for Pellet Induration System Network Simulation :

Evaluation of leaks is one of the main objectives of pellet induration system air flow distribution package. In the literature, detection of leaks has been discussed for natural gas pipe networks, but these are backed by transient models, which either use SCADA (supervisory control and data acquisition), telemetry data (e.g. SIROLEAK code by Turner and Mudford 1988) or depend on the leak detection hardware (Butler 1982). For pellet induration system, the (to be developed) model should be steady state, and as conveyed by the practitioners, the instrumentation to determine leaks is not available, hence the described methods are not of much use.

The proposed model for the evaluation of steady state airflow distributions in pellet induration system networks is to be based on device centred or unit based approach (as called by Afzal and Cross 1992) for formulation and for solution, as well. The unit based approach will be discussed in detail in section 2.3.4 and it will enable us to :

- Model heterogenous components of pellet induration systems as such,
- Carry out decentralised or distributed computation and benefit from the nature of equations and use appropriate numerical schemes,
- Model leaks like any other network component by treating atmospheric nodes as fixed grade nodes, having known pressures,

- Refine or change the existing (mathematical) models of any of the components,
- Add or delete the instances of any of the modelled component,
- Include more components into the database of modelled components, by writing their mathematical models, programming them to conform to a standard pre-defined interface, and
- it would not require any extra storage space for computation as needed by matrix methods.

The benefits and qualities of the resulting code will be discussed later on in sections 2.4 and 4.5.

1.4.3 Route to Implement the Proposed Method :

The proposed model or computer package has to be developed from scratch, so there are no constraints about the component models, solution algorithms, or code architecture and we can benefit from the existing state of the art methods and techniques, as long as these are useful to the applied context. For example, graph theory provides a very elegant way to handle network connectivity and leads to efficient computation, and for the very reason, has been widely used for the simulation of flow networks (section 1.1). Similarly, from a software perspective, object oriented technology has been proposed as solution or '*silver bullet*' for the development of complex engineering software systems (e.g. Cox 1990, Wilkinson and Byers 1993). In this work we will use these proven techniques.

The evaluation of the airflow distribution in a pellet induration system network, using proposed method will mainly require the following steps:

- The actual pellet induration system network will be abstracted into a connected graph of nodes and streams, where streams may consist of single or multiple serially connected components, and nodes are meeting points of more than one stream, or endpoints of a stream,

- The equation set will be decoupled, and first pressure and flow distributions will be computed, which will later be used for temperature distribution computation (see Wilson et al 1986, Fincham and Goodwin 1988),
- The computation of pressure and flow distributions will be carried out by a two level hierarchical algorithm. At higher or coordination level the abstracted network of nodes and streams will be solved to satisfy Kirchhoff's current and voltage laws. The so computed stream flows and node pressures, the coordination (or 'parallel' as called by Boghosian 1990) variables, will be passed on to the streams as parameters for lower or component level computation. The finally computed values should satisfy Kirchhoff's laws as well as component models. Osiadacz and Salimi 1988a, 1988b proposed a similar two level algorithm for transient simulation of gas flow in single pipe and pipenetworks respectively, and discussed its benefits for parallelisation. For coordination level computation we will use an algorithm similar to the one proposed by Boyne 1970.
- The coordination level solution algorithm (section 3.3) requires the partitioning of the network into forest (collection of trees i.e. acyclic graphs) and coforest structures, for which algorithms based on heuristics related to pellet induration system networks and the constraints of the intended code, will be developed (section 3.5). This development will be founded on graph theory.
- The code development will be solely carried out on software engineering concepts, as it promises the solution to the *software crisis* and guarantees the benefits like re-usability, portability, low maintenance costs etc. In addition, it will provide the benefits claimed by object oriented paradigm.

How and to what extent these objectives are achieved will be explained in the following chapters. The mathematical models of network components and definitions relating to the graph theory (to the level it is needed), will be discussed in section 2.2.3. The primary solution algorithm for network computation and secondary algorithms for network partitioning and temperature computation etc will be illustrated in Chapter 3. In Chapter 4 the software engineering concepts and techniques used to realize the proposed solution method into a

computer code, GASFLO, will be explained, also the resulting code will be graded against the object orientation criteria. In Chapter 5 the capabilities of GASFLO and its application to simulate real life pellet induration plant will be discussed. In Chapter 6 the research will be concluded pointing out the directions in which further work could be carried out.

1.5 Summary

In this chapter, after a brief discussion on the commonalities of all fluid flow networks, an overview for network formulation and solution methods is given. In the classical literature three main formulations; loop, nodal and full equation; and three main solution methods; Hardy Cross, Newton's and Linear theory; are encountered. These all along with some more recent methods are briefly described in Section 1.1. In Section 1.2 objectives of present research are stated, namely, to develop a software tool which should evaluate steady state pressure, flow and temperature distributions in pellet induration system networks and this tool should have all properties of quality software.

In Section 1.3, pointers to other applicable literature are provided, which somehow influenced this research. The principal difference between the classical network solution methods and the proposed method is the fundamental approach to modelling, the former used process centred approach whereas the proposed one uses a device centred or unit based approach - responding to the fact that systems are composed of primitive components, so the behaviour and functionality of their primitives should be reflected in the performance of the system.

In section 1.4 the means to achieve the stated objects are briefly described and for details the pointers are provided.

Chapter 2
Mathematical Modelling of Pellet Induration Systems

2.1 Introduction

In iron making and steel making industry blast furnace is the main production process. The raw material fed into the blast furnace, is commonly called '*burden of blast furnace*', and it needs to be of good quality, to provide a uniform gas-solid contact across the stack. It should be chemically reducible and keep thermal demand on blast furnace as low as possible. High grade crushed ore, with size between 10 - 25 mm, could be fed in directly as burden, but common iron ore, or particulate cannot be used directly, as burden, instead it is first agglomerated, in the form of *sinters* or *pellets*. According to one estimate, the annual production from blast furnace process was over 200 million tonnes in late 1970s (Cross et al 1982), which is much more than at present day. The large production rate, desired high quality of the product and economics of the process, emphasize the importance of the detailed study of the agglomeration process.

There are two main processes used for agglomeration:

a) Sintering Process
b) Pelletizing Process

2.1.1 Sintering Process :

In sintering process, the mix of crushed iron ore, coke, water and other binding material, is loaded onto grate or bed, in a layer of constant thickness and covered by a hearth layer of pre-sintered material. It is passed through the sintering machine. Sintering machine

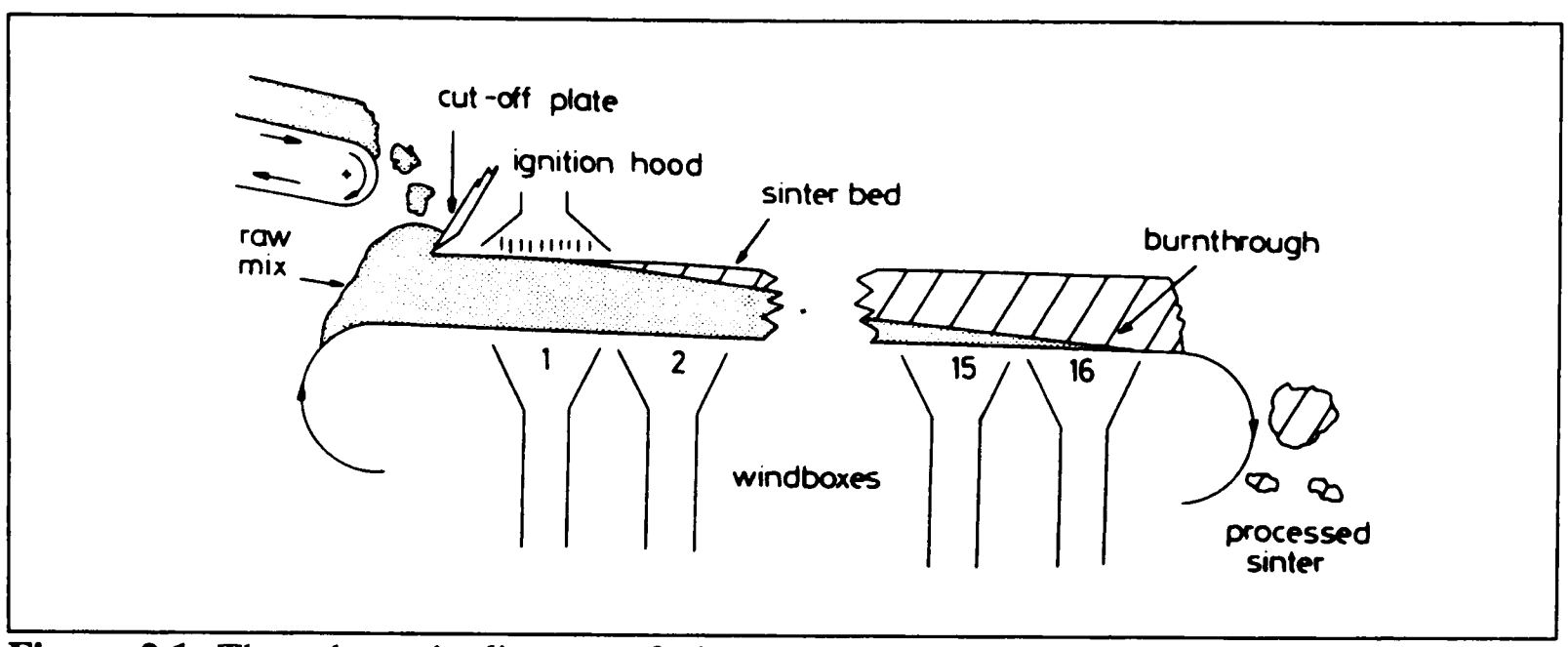

Figure 2.1: The schematic diagram of Sintering Method

is in fact a combination of an ignition hood followed by a series of wind boxes (as shown in figure 2.1). The downward moving hot air ignites the coke particles in the layer near top surface. As the bed moves further, the downward moving air, drives this ignition layer towards the bottom of the bed. Thus igniting and fusing the whole mix, which on cooling reduces to small chunks called sinters. These are directly fed into blast furnace.

Powerful fans, regulate the flow of hot and cold air through the bed. This also includes the recirculation of the process gas among different wind boxes, so as to minimize the heat loss from the system.

It is a continuous process and sintering machine is usually used as an integral unit of blast furnace plant. Rose 1981 has described the process and given references for detailed reading on the subject.

2.1.2 Pelletizing Process :

In the pelletizing method, the crushed iron ore or the particulate (< 5. mm diameter) is mixed with water and some binding material like, Bentonite and rolled into small pellets either of spherical or cylindrical shapes of typical diameter of 12 mm (Rose 1981). The wet pellets commonly known as '*green pellets*' are passed through three stages of, drying and pre-heating; firing; and cooling. The first two stages are similar to that of the sintering process.

All these three stages are crucial for produced pellets' quality. Sudden and uncontrolled temperature changes may damage the texture and strength of the pellets.

Although the pelletizing process is a continuous process, it does not have to be located at the iron or steel works. Many of the plants are located at mineral sites and the produced pellets are transported and marketed internationally. Pelletizing is also known as concentration process, as the moisture which is about 10% by weight evaporates during drying and pre-heating, and on firing the volume is reduced (Cross and Wade 1989), so the produced pellets are richer in iron content.

There are three mainly used pelletizing processes:

a. Shaft furnace process
b. Straight grate process
c. Grate kiln process

a. Shaft Furnace Process : The green pellets are fed in at the top of the vertical furnace and hot air is forced into it from the middle (see Figure 2.2a). The pellets move down by action of gravity in controlled environment, and exit at the bottom. During this travel they are dried up, pre-heated, fired and cooled down.

This is the oldest method and has an upper limit of 500,000 tonnes on its annual production. Due to this constraint and other efficiencey reasons, no new system is built after mid 1960s.

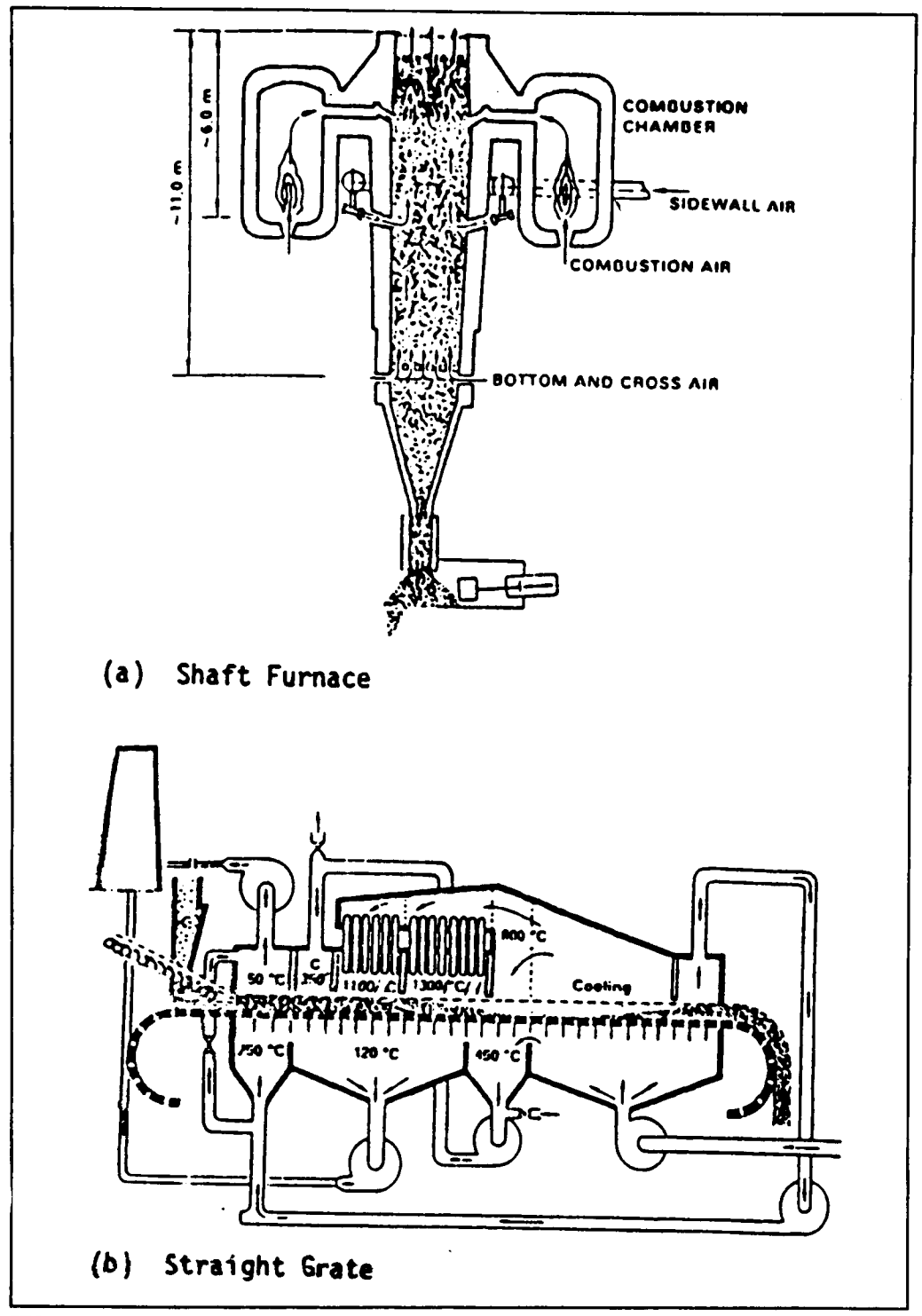

Figure 2.2: Schematic diagrams showing Shaft Furnace and Straight Grate Methods

b. Straight Grate Process : Structure wise it is like sintering machine, but instead of crushed iron ore, green pellets are loaded on moving grate, as input (shown in Figure 2.2b). As they move deep into the system, they are dried up, pre-heated, burned and finally cooled. For heat conservation, the entire chamber is divided into zones, which enable unidirectional gas flow, either in upward (up draft) or downward (down draft) direction. These zones are interconnected by large diameter pipes or ducts so that the heat transfer between gas and pellets is maximum, and heat loss to the atmosphere is minimum. High power fans are used to drive the process gas through the system. External burners supply the heat energy to the system. Air at atmospheric temperature is sucked in at cooling end, which on passing through the packed bed extracts heat from fired pellets. This hot gas is recouped to the system at convenient zones, such that heat could be transferred back to cold or wet pellets and finally it is pushed out of the system.

The zone connections are so designed that the temperatures in all the three stages of the process (drying-heating, firing and cooling) should remain within desired limits as well as there should be no sudden temperature changes across the packed bed of pellets. All the three stages of the process occur in the same chamber. This is an efficient and widely used method.

c. Grate Kiln Process : It is an improved version of straight grate, where all three stages of induration process are physically separated. The firing takes place in an exclusive unit, kiln, heat is supplied by external sources and by chemical reactions. The temperature and complementing chemical reactions, could be controlled precisely, as compared to other methods. The working principle and zone connections for gas flow are similar to the former method. Figure 2.3 shows the schematic of a grate kiln system.

Further the bed depths could be varied for drying and cooling stages. Also at some plants the circular grate are installed for cooling stages. All these refinements make the system more efficient and hence, the most used one.

The straight grate and grate kiln systems are similar in structure and functionality, these could be simulated by the same software. This approach has been implemented for heat distribution codes, like INDSYS (Cross and Englund 1987, Cross 1988) and CASCADE (Patel et al 1993). The developed mathematical model and the resulting software code, GASFLO, for grate kiln system, would simulate both systems. The simulation of straight grate system, using GASFLO, will require the substitution of kiln by the respective firing zone.

In this thesis, hereafter the term *'pellet induration system'* would refer to either of the two systems in general, but for the sake of clarity and consistency, only grate kiln system would be mentioned as the example pellet induration system in all text and figures. Its working, with reference to airflow, is discussed in next section.

2.2 Pellet Induration System

The objective of an efficient system is to produce high grade pellets at minimum costs. The process is illustrated in Figure 2.3.

The green pellets are charged to a moving grate and fed into the down draft DD1 (see Figure 2.3) zone of drying stage, and moved slowly in temperature controlled environment towards pre-heat, PH zone. At the exit of PH, their temperature reaches to about 1000 ^{0}C, then they are fed into the kiln, where they are fired at the temperature of about 1300 ^{0}C. The

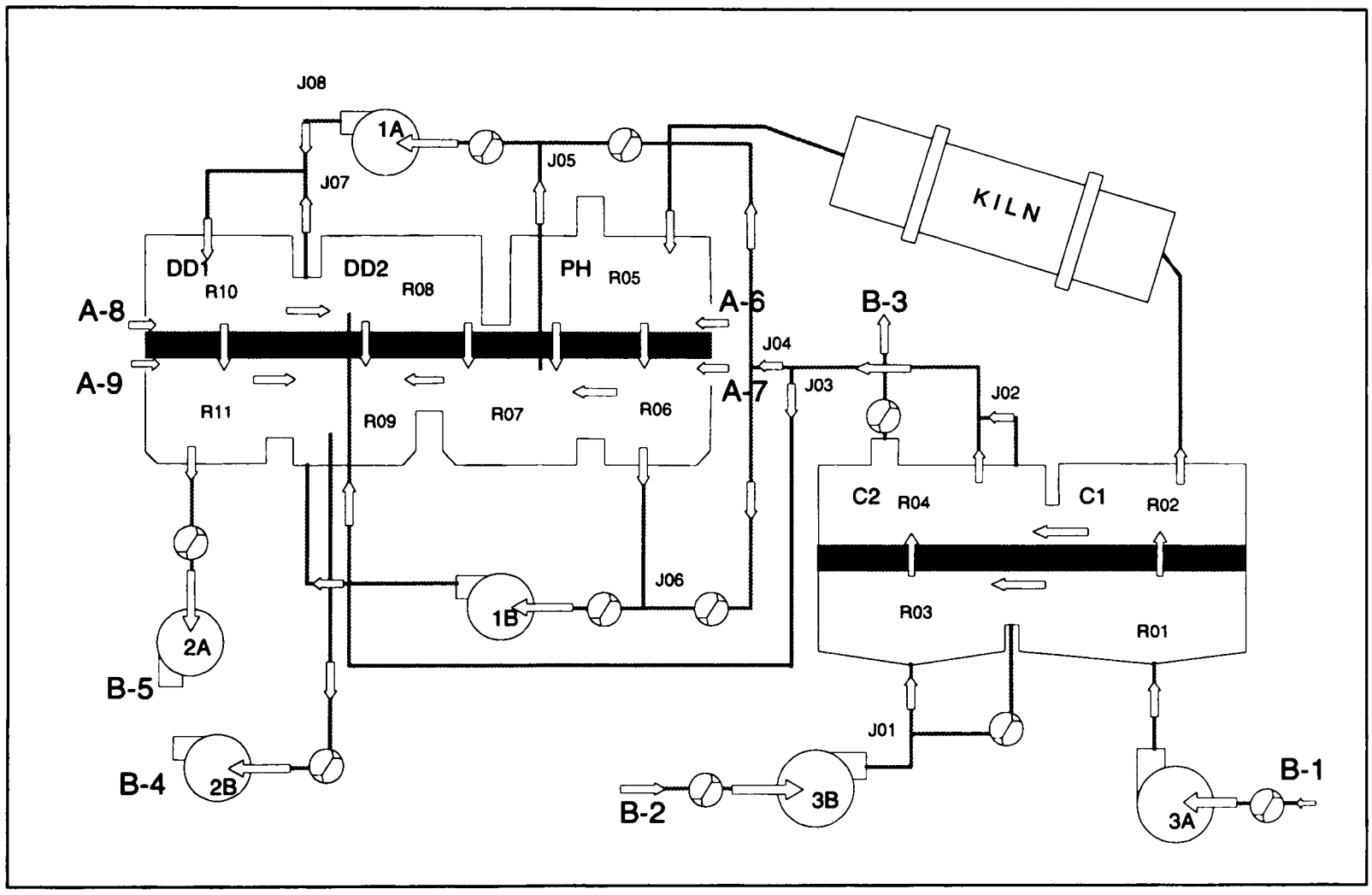

Figure 2.3 : A typical pellet induration System

fired pellets are then cooled in cooling stage. In this whole process, air is used as the heat transport medium. The cool air, at ambient temperature is sucked in by fans 3A and 3B in cooling zones, C1 and C2 respectively. It is forced to pass through the packed bed of fired

(hot) pellets and on contact, the air extracts heat from the pellets. This hot air is recouped into the system. Some of it is passed onto the pre-heating stage via the kiln, thus raising its temperature further, whereas the remainder goes to the drying stage directly by connected pipes or ducts.

The booster fans 1A and 1B, help in maintaining flow within the network, and generate sufficient head to overcome the head-loss suffered by airflow due to duct wall friction, the resistance offered by packed bed and other in-line instruments. The valves are used to regulate the flow through different paths of the network. The zones are so connected, that the temperature variation is smooth in the whole process and specially in drying stage. The hot air blown through the packed bed in drying zones, extracts moisture from the green pellets and dries them up, losing its own temperature. Finally the used air or off-gas is pumped out of the system by fans 2A and 2B.

To ensure the free, unobstructed movement of the loaded bed, sufficient clearance between the bed and the zone or system partitions is provided. Though these gaps are very small as compared to other dimensions, only few centimetres in hight, still these are potential source for leaks into the system, and contribute towards the process inefficiency. Further, these leaks can not be measured due to very high temperatures in the system and lack of instrumentation. Depending upon the neighbouring regions' pressures and the cross-sectional area of the clearance, these leak flows are quite large and effect the temperature and pressure distributions significantly. It is practically observed that the 30-45% more air passes through the fans 2A and 2B than is recuperated into the drying and preheating stages from cooling stage of the system.

2.2.1 Pellet Induration System as Pipe Network :

Our aim is to find the pressure, flow and temperature distributions of process gas (air) flow in the system, from that perspective the shown system is like a pipe network in abstraction. In this respect, it is comparable to natural gas and water distribution municipality

pipe networks. Although it is smaller in size than the distribution networks, it is comparatively more complex due to the variety of the units involved. In section 2.4 this aspect will be discussed in detail. The pipes, in fact very large diameter ducts, are for the conveyance of the airflow among different parts of the system. Leaks and packed beds have nearly the same functionality, as they connect different regions and provide access for process gas to flow through them. Whereas at the junctions and zone regions the gas coming from different paths gets mixed. These components are discussed in detail in the following sections.

Like other pipe networks, the induration system is also solved as a network, and two Kirchhoff's laws must be satisfied.

2.2.2 Pellet Induration System Components :

The pellet induration systems are big industrial plants, covering large areas of land, having components of all sorts and complexities. From the gas flow distribution perspective, the following components play an active role:

Fans: Fans drive the process gas in the system, as does pumps in water networks or compressors in the natural gas distribution networks. These are usually at system extremities, ie upstream to cooling stage to suck-in the on-gas (cold air) into the system, and at the downstream to drying stage to push-out the off-gas (the used hot air) from the system to the atmosphere. Fans are also installed inside the system to efficiently regulate the air flows among different zones. The economics of plant and ultimate cost of the produced pellets are significantly dependent on fans, due to their initial installation and subsequent running costs.

Pipes: Pipes or ducts provide the interconnection between different components of the system. These are of different shapes and different cross sections, connected in series or in parallel. These are very large in size, like mine ventilation system airways,

having few squares of meters of cross sectional areas. Pipes are insulated to avoid heat loss.

Zones: Zones are the physical partitions in the heating and cooling stages of the system. They provide a controlled temperature environment, and help to regulate gas flow in specific directions. Zones have input and output regions, separated by packed bed of pellets. All heat transfer from pellets to air and back to pellets, take place in zones. In other words a zone consists of an input region, a packed bed and at least one output region.

Regions: Regions are enclosures above and below the packed beds where different airflow streams meet. These streams have specific values of system variables i.e. flow rates, pressures and temperatures of the gas.

Packed beds: These are beds of pellets, through which process gas is forced to flow. At the entry to the system the pellets are wet, their moisture is evaporated by the hot air as they move deep into the system. A packed bed has a fixed width, height and other pellet properties for a stage. However, these properties vary between heating and cooling stages.

Junctions: Junctions are analogous to regions, but these are meeting point for two or more pipes. They provide alternative paths to gas flow in the system, which can be used for optimal functionality. For example, the pressure of off-gas (i.e. the gas leaving the bed) is always less than that of on-gas (the gas entering the bed), and its temperature is dependent upon the temperature of the bed. The pressure of the off-gas stream leaving the bed, could be improved by providing a path, bypassing the respective zone. This down stream pressure could be controlled (to some extent) by controlling the flow through the by-pass. In Figure 2.3, the pipe connecting junctions J01 and J02 has similar function.

Valves: Valves are used to control and regulate flows in different paths of the network, by restricting the cross sectional areas of respective pipes of the system.

Leaks: Leaks are associated with regions. These are flows between two neighbouring regions or between a region and the atmosphere. The former are called 'internal' and the later as 'external' leaks. In other words these are the flows, which escape through the small clearances above and below the bed. As stated earlier, complete sealing of these leak flows is practically impossible.

2.2.3 Graph Theoretic Representation of Pellet Induration System :

All disciplines have their own terminology and representation styles for easier communication. For example the schematic of pellet induration system shown in Figure 2.3 is very straight forward for an operator or a plant engineer in the induration industry, all the geometrical data of the components and measured data from the systems' instrumentation could be plotted on the schematic. Similarly the opening and closing of valves and variation of wattage of any of the labelled fans would be equally un-ambiguous to all concerned. Unfortunately that representation is not that useful from analysis and computational point of view, where we would be more interested say, in the detection of any cycles or loops for the satisfaction of Kirchhoff's second law.

In sections 1.1 and 1.2 the use of graph theory for the solution of flow networks was mentioned. In fact graph theory provides an alternative presentation, which is graphical as well as set theoretic in nature, and very helpful for the development of algorithms and their implementation. It is widely used in Computer Science, Deo 1974, for sparse matrices Duff et al 1990, and for network computation Boulos et al 1991, Osiadacz A J 1987. Graph theory is a full fledged subject in its own right and reader can consult any standard book (e.g. Deo 1974, Wilson and Watkins 1990, or Syslo et al 1983). Graph theory provides a powerful tool to visualise networks and their computation and analysis.

A Graph G can be defined as combination of a finite set of streams $S = \{ s_1, s_2, .. s_S\}$ and another finite set of nodes $N = \{ n_1, n_2, ... n_N\}$ and can be represented as $G = (S, N)$. Also each s_i can be presented by an ordered or un-ordered pair of nodes. If the streams have some specified direction then the G is called as *directed graph or digraph.* Different authors use different names for streams and nodes; as edges, links or lines and as vertices or points etc; respectively. No pre-requisite knowledge of graph theory is assumed, the related definitions would be given when required in the text.

To convert the pellet induration system schematic into a directed graph, we examine the system and

- mark all the *nodes*, where more then one flow paths meet
- mark all the paths linking distinct nodes as *streams*. Some of the streams will consist of single components like leaks, beds or pipes, whereas the others will have multiple components like pipe, fan and pipe; all connected serially and having constant flow through them.

This categorisation points out that the node will have a constant pressure and temperature whereas the stream will have fixed flow. The system is linked to atmosphere, through streams containing fans 3A, 3B, 2A, 2B and external leaks, which will cause different local pressures, so it will be reasonable to treat these locations as separate nodes and further, as these will have just one incident stream so we can characterise them as *external nodes,* whereas, all other nodes which lie with in the system, have more than one incident streams, are called as *internal nodes*. These internal nodes will consist of either a junction or region and external nodes can represent a boundary. The atmospheric boundary linked to the system by a leak has different characteristics than the boundaries which act as source or sink, so these are named differently. The abstraction hierarchy from actual components to the graph theoretic node stream objects is shown in Figure 2.4.

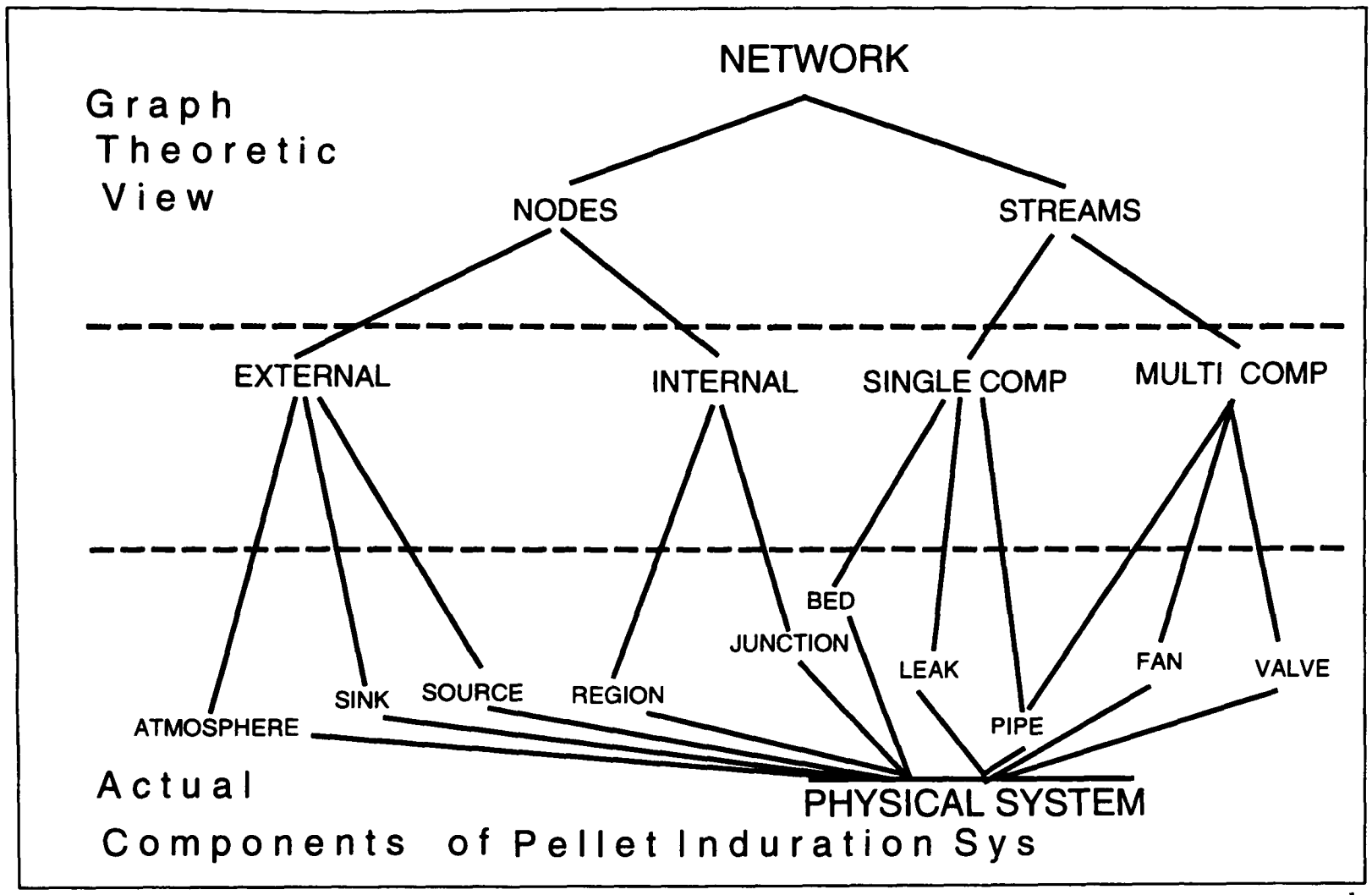

Figure 2.4 : The hierarchical relationship of pellet induration system components to graph theoretical network object.

Applying above stated transformation to the original system and representing streams by straight lines and nodes by circles, we get the graph theoretic representation (see Figure 2.5). Different shading patterns are used for internal and external nodes to show their distinct nature. For cross referencing (to Figure 2.3) the node names have been marked. This representation is a good tool for visualization of the original system and for development of algorithms e.g. the loops in the corresponding graph can be very easily specified.

Referring to Figure 2.5, the node R05 has 4 *incident* streams, two of them are *incoming* from nodes R02 and A-6, so it has *in-degree* of 2, and the other two are out going streams, so *out-degree* is also 2; and *total-degree* of node R05 is 4. Also R02 and A-6 and the corresponding streams are called *predecessors* to R05, whereas R06 and R07 are its *successor* nodes. *Up-end* and *down-end* nodes of a stream are with respect to the (assigned) flow direction of the stream. *External* nodes have 'total-degree' of one, i.e. these have only one incident stream. If the respective stream is 'incoming' then the node is *sink*, and if it is

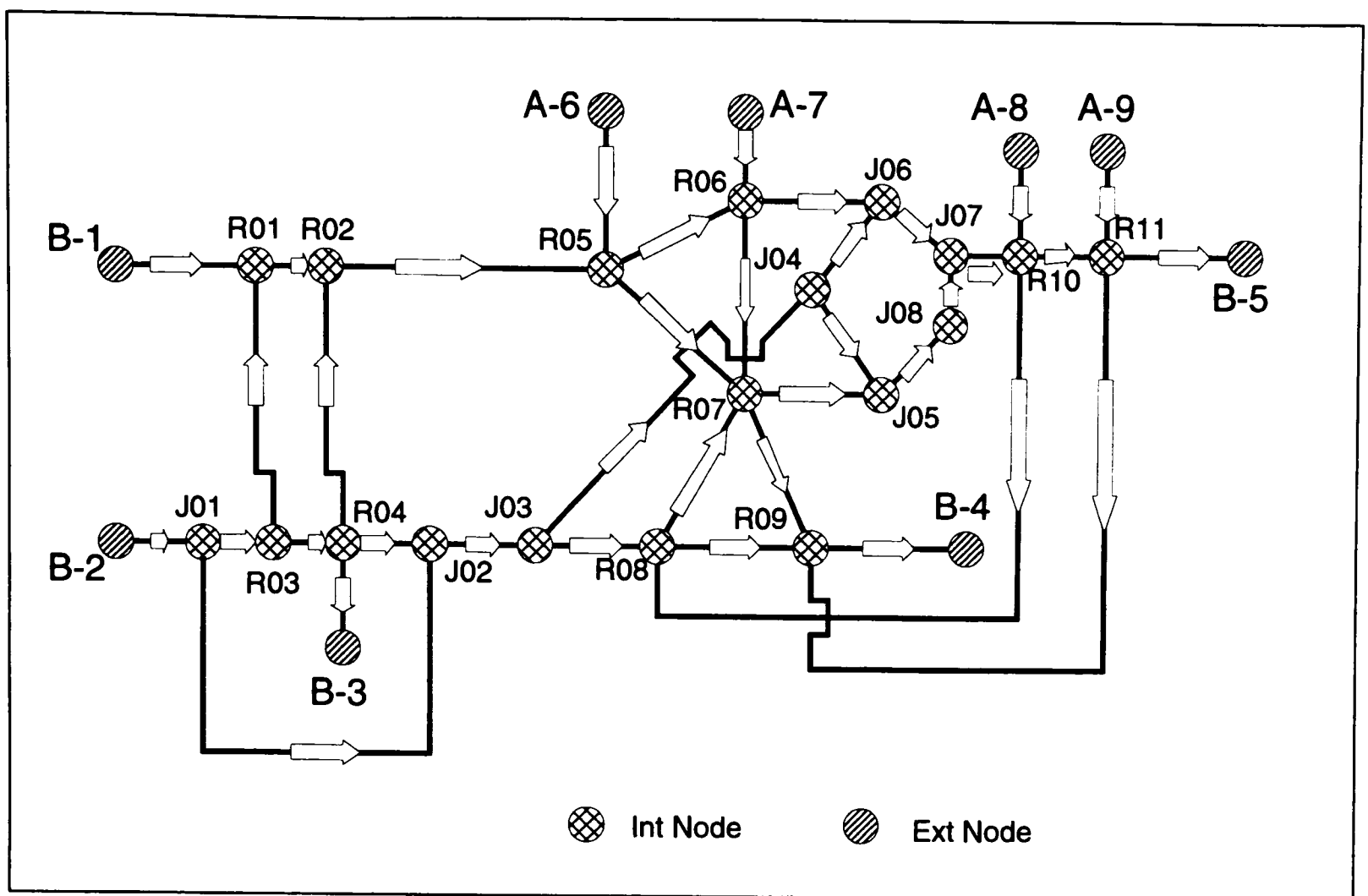

Figure 2.5 : The graph theoretic presentation of pellet induration system

'out-going' then the node is *source.* All *internal* nodes have 'total-degree' of more than one.

The connectivity shown in the Figure 2.5 could be presented mathematically in the form of *adjacency* or *node-stream incidence* matrices of order N × S, for a N node and S stream network. All the nodes and streams are assigned integral numbers and are referred by the same numbers. The incoming stream are be presented by +1, and all out going by -1; this convention has been adopted because intuitively the incoming flow adds to the holdup of the node whereas the outgoing depletes it (Yevdokimov 1968, Hamam and Brameller 1971 and Boulos et al 1992). Most of the other authors (e.g. Osiadacz 1987, Boulos and Altman 1993) have used reverse signs for entering and leaving stream, whatever convention is adopted it is important that it should be followed consistently through out the formulation. For the sake of clarity the node and stream numbers were not shown in the figure. In the 'node-stream incidence' matrix, the ith row will have non-zero elements (±1) in columns corresponding to the streams incident on this ith node, similarly the jth column corresponds to the jth stream,

and will have only two non-zero elements against its up-end and down-end nodes. In short the node-stream incidence matrix can be presented as

a_{ij}	=	**+1**	if jth stream is entering ith node
	=	**-1**	if jth stream is leaving the ith node
	=	**0**	otherwise

Similarly the *loop* or *mesh* matrix of order L × S, could also be written as

bij	=	**+1**	if jth stream is in ith loop and has same direction as the loop
	=	**-1**	if jth stream is in ith loop but has direction opposite to the loop
	=	**0**	otherwise i.e. jth stream is not in ith loop

The above is the simplest form of representing graph mathematically, more compact forms like, *node-node adjacency* (N x N) can contain the stream numbers with +ve or -ve sign showing whether it is in-coming or out-going stream. Further, the 'in-', 'out-' and 'total-degree' of a node could be worked out by examining the number of non-zero elements on the corresponding row. The *predecessor, successor linked lists* are even more compact and efficient representations, which will be briefly discussed in section 3.5.1 and shown in Block A.4. Deo 1974, Syslo et al 1983, Osiadacz 1987 and Ahuja et al 1993 have described the graph theory application and the related data structures, in detail.

2.3 Mathematical Modelling and Process Analysis

Mathematical modelling is as old as any of the science subjects like Physics and Mathematics. Any formula say, Hooke's law of stress analysis, Ohms law of electricity, Maxwell equations of electrodynamics or Navier Stokes equations of fluid mechanics, each one of these, models a physical phenomena or relationship among some quantities. Even till the late 1960s it was restricted to only small problems constrained by available human computational power. Some relatively larger problems were only catered by research

organizations or universities, but that also had more theoretical content. The advent of computers, their increasing compute power, the mass production of hardware and ever decreasing prices have changed the whole scenario of application of mathematical modelling. Now it is hard to find a single field where computer models are not used. From social sciences to microbiology, all branches of engineering; aeronautics, automobile, avionics, chemical, civil, electrical, electronics, hydraulics, mining, petroleum, to name a few; nearly every one has put computer modelling to the best possible use. Now the mathematical modelling has become a vital tool for every field of engineering and industry.

Some other contributing factors responsible for this rapid growth are:

- the widespread of knowledge of numerical analysis and other related disciplines
- maturity of science subjects to an extent that their results could be integrated to develop full fledge models
- experience and benefits gained by the use of existing models and the possibility to build models on the top of the existing ones, by improving or re-using them
- presence of infra structure for their development, availability of proper computer languages and other related hardware and software tools
- willingness and sincerity of the experts from related fields to share their skill and cooperate in the development of integrated models
- openness of user, to use them as a valuable tool and to exploit their power even beyond the foresight of their designers
- the most important of all is, the availability of immense 'raw' compute power and inexpensive hardware.

The wide range use of these mathematical or more precisely the computer models in all walks of life is a practical proof of their utility. In the following we mention few of the benefits these models provide.

- models provide reliable, robust and fast solutions to industrial problems

- these are inexpensive in terms of time, as well as, in finances, as compared to the other alternatives
- these are flexible, easier to use and doesn't penalise the user for not being expert
- models provide greater control and insight of the problem, to the user at his own pace and wish for refinement
- these can be used for the initial design of plants, analysis and optimization of the existing plants, working out operation strategies, training of operators, fault diagnosis, expert advice and real-time control of plants
- these can simulate what-if scenarios, and can provide answers to the situations, which otherwise involve risk of human life and property
- models eliminate the need of building prototype, and so the risky situations, when the large scale actual plants built on well tested prototypes, do not work; leading to multi-million losses.

Of all these models, a wide spectrum is encompassed by process analysis and Computational Fluid Dynamics (CFD). Any process or system which embodies, a 'continuum' somehow, could be dealt as a CFD problem, whether it is airflow around an aerofoil, fire propagation within a building or a complex multi-phase flow involving chemical reactions. Petridis et al 1991 and Knight and Petridis 1992 have covered the internal and external requirements for the development of CFD software, such aspects will be discussed in section 4.1.3 and properties of the intended code will be covered in section 4.5. Here we restrict ourselves to core of these models, i.e. mathematical model, and concentrate on process analysis.

2.3.1 General Aspects of Mathematical Modelling

Every computer model has an underlying core, the mathematical model, a set of equations which are solved numerically and in the end, these should respond to the valid, realistic inputs, in a manner consistent to physics. The models are evolving by nature.

Although huge compute power is available nowadays, but still the whole physics of a process can not be embedded into the model in one go, instead it is added to the model incrementally. The main activities involved in the development of a computer model are:

a. Selection of the processes of the system to be modelled
b. Mathematical representation of these selected processes, and specification of the made simplifying assumptions if any
c. Computation of solution of the mathematical model (numerically)
d. Validation of the computed results; these should
    i. be convergent and stable numerically
    ii. not violate, the involved (or modelled) physics
    iii. justify the available experimental data
e. Enhancement of the modelled domain by the
    i. inclusion of the deletions made during simplification of equations i.e. up dating of the modelled process to full extent
    ii. addition of further processes

The mile-stones of the modelling process are shown in Figure 2.6, the above activities are shown by arrows. In principle, the 'computed (numerical) solution' should satisfy the discretised domain, which is subset of 'modelled domain', but by satisfaction of (d) above we can safely assume that it is valid for whole of the modelled domain. The ultimate aim is that the modelled domain should be as close to the real world as possible. It is evident from the figure that the modelling process is cyclic by nature.

2.3.2 Approaches for Process Analysis Modelling

To develop a mathematical model for a complex system, first we analyze its working, which could be done either by looking at its overall behaviour and analyzing the involved processes; or by seeing its physical constituents and finding out, what is the functionality of each of its components. Accordingly there are two main approaches for modelling of process or any of the complex system (see section 1.3 and Babrow 1984). These are :

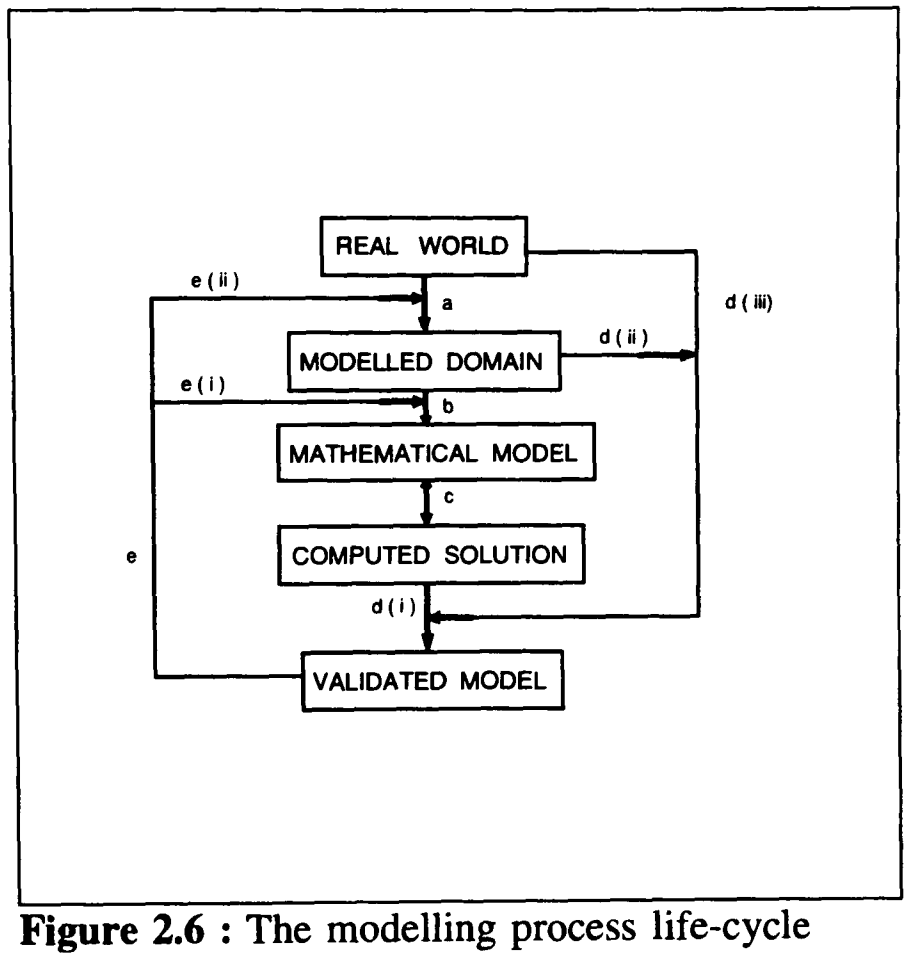

Figure 2.6 : The modelling process life-cycle

Process Centred Approach: Where the main stress is on the detection of the involved '*Processes*', and modelling of these processes to form the whole system. This is a distributed approach, since each process spans over a number of physical components, usually the complete modelled domain, so the complete set of equations is computed simultaneously. This approach is most widely used in the present day CFD modelling. For example, if we have to find the steady state flow, pressure, temperature and species concentration distributions in the chemical plant network, then by 'process centred approach', the equations for these distributions are written for the whole plant, (as these are coupled i.e. mutually dependent on each other), and solved simultaneously. In solution strategy, the inter dependencies of the state variables could be exploited, say if, pressure and flow are mutually dependent, and temperature and species concentrations are dependent on flows, but not vice versa; then at the first stage the coupled set of pressure and flow equations could be solved, and later using these values temperature and concentration distributions are solved.

This is an established approach, the usage of variables inter-dependencies and execution of independent processes before the execution of dependent processes, can lead to significant gains in computational time. Similarly in the transient case the slowly varying processes could be skipped for couple of intermediate time steps.

Most of the present day CFD and other process analysis software is based on this approach and hence it can be equally blamed for their shortcomings and in-efficiencies.

Device Centred Approach: In this approach, attention is fixed on the physical components of the system, these components, called *'Devices'* are modelled separately, and finally linked together to form the whole system. This approach has prompted time and again in the literature with different names; as *device centred* by Babrow 1984 and DeKleer 1984; as *sequential modular* by Montagna and Iribarren 1988a and 1988b; as *object centred* by Lipworth et al 1991; and as *unit based* by Afzal and Cross 1992. This conforms to the well known *object oriented* paradigm of computer science and it does have nearly all the benefits mentioned in object orientation context. The components are physically connected to each other, this connectivity and the information transferred along the interface is very important and somehow, it has to be imbedded into the model.

The components are modelled and executed separately and use the interface information as parameters or independent variables for their internal computation. Boghosian 1990, while explaining his 'data-parallel programming methodology' calls these variables as parallel variables. The equations related to all the processes being carried out in the component are executed simultaneously inside the component module.

Unfortunately this approach was not much used, so its benefits could not emerge until recently, when the shortcomings of other approach came to evidence as mentioned in section 1.3.

Detailed models for CFD, computational physics or any other natural science, if modelled using former approach, soon become infeasible, even on super computers with teraFLOP performance, due to their huge computational load and hence exhaustive CPU times and storage requirements. For example Jones 1993a quotes that the computation of turbulence (k-ε) model, with refined scale, for channel flow problem, would require about 250 hours on CRAY X-MP. Which suggests that some fundamental strategic change in the basic modelling approach is required. The device centred approach seems to be the promising solution.

2.3.3 Changing Environments and Mathematical Modelling

In this section we briefly discuss the main features of computer models in the present day changing environments. The presence of these models in the scientific environment shows that these are irreplaceable and evolving by their nature, so they should be designed with broader prospective. The reasons for their longevity being their functionality and the investment in terms of human effort to improve them to their existing states. As mentioned earlier, every computer model has a core, i.e. mathematical model, which we will be referring to in this section. Other implementation matters of these models will be covered in Chapter 4.

At the design level we should take into account the following factors, to which our models should complement:-

Technological Advances: Most of the time these are related to the advancement of computers, which is effecting the environment from many angles. Computers are getting cheaper, their usage is increasing and so is the usage of models, as Chansler and Rowe 1990, state that now the microcomputers are being used in all the fields of

water industry, from design to control of distribution networks due to their inexpensiveness and speedy response. Previously the users of the models were qualified engineers, but now they could be operators with lesser academic qualifications. Also as their compute power is increasing, so the more complex engineering problems are now solvable, hence more complexity in the models can be added. This varied spectrum of addressed problems and users, suggest that the models should have an 'ease of use' factor.

Secondly this advancement has introduced the wide spread use of computers in all other related fields. Now computers are widely used in the field of measurement and instrumentation. Petley 1991 has given the refined (exact to the date) values for physical constant used in SI system of units. The results produced by a model having equations with imprecise coefficients (or physical constants) would be harder to match to the presently available precise measured data within micron accuracy. Our models should be adaptive to such technological changes.

Experience has shown that by advances in measurement techniques and use of computers in Industry, the hardware components are continuously improving and so their related equations, especially if these are empirical relations. For example Young et al 1979, quoted the following relation for temperature dependent specific heat capacity of air,

$$Cp(T) = 1026.3486 - 7.14326\times10^{-2} \times T + 4.54916\times10^{-4} \times T^2 - 2.1334\times10^{-7} \times T^3 \qquad (2.1)$$

which gives the value of 1040.06 J/Kg-°K as compared to the measured one of 993 J/kg-°K at 300°K as given by Tennent 1971. In contrast Zografos et al 1987 gave an improved relation

$$Cp(T) = 1061.3 - 0.43282 \times T + 1.0234\times10^{-3} \times T^2 - 6.4747\times10^{-7} \times T^3 + 1.3864\times10^{-10} \times T^4 \quad (2.2)$$

which gives the value as 1007.21 J/kg-°K, which is closer to the experimental. Thus the architecture of mathematical model should be such that it could readily accommodate these changing in component equations.

Thirdly, there would be more target sites for the produced models. Just like a Japanese car designer, the model developers should also think globally for the requirements of these future users. The facilities for the use of local system of units, the presentation style and other cultural issues, should be taken into account.

Human knowledge advancement: This is effected by the constant use of the existing models. The state of knowledge of a user, constantly improves; the first time user, would have different expectations and would interpret the produced results differently, then an experienced user. The availability of alternative models for the same problem, may lead to a different usage of the same model at a later stage. For example previously it was used as prediction tool for experimental measurements, but later it may be used to validate the new model. The models should be flexible enough to co-exist and communicate with other models of the field.

The scientific world is constantly changing, people are coming up with better mathematical models of different complex phenomena every day. For example now there exist about a dozen turbulence models, each one having its own specificities and claiming to be better then its predecessors (see Boyson 1993 for a new one). Likewise Manning proposed two formulas for hydraulic computation in 1889, and in 1895 he recommended one of them, which was dimensionally consistent but interestingly due to its complicated nature, was not well received and the other was extensively used by hydraulic engineers until present day. Yen 1992, has now obtained an improved dimensionally homogeneous Manning's formula, for water networks' computation.

The architecture of the models should be such that it could readily adopt these sub-models as disposable and replaceable items. Similarly the algorithms and computational techniques are improving, so these should also be swappable. The device centred or unit based approach seems very promising and able to accommodate these specified future needs of our models.

2.3.4 Device Centred or Unit Based Approach

According to this approach the physical boundaries of the system components serve as the modelling boundaries of 'devices' or 'units' of the system. Their connectivity with the neighbouring units point out the source and target of the information, and the nature of information being passed through the interface governs the order of computation. The models of all the processes occurring inside the units, are encapsulated inside the computational units, so the variables involved in the mathematical equations would require some considerations. The distinction between local and global variables is very important, global variables (or parallel variables as called by Boghosian 1990) are the state variables, for which the system is being solved. The process could be any physical process taking place in the unit. Depending upon the problem the respective conservation laws should be satisfied locally, as well as, globally for a steady state system.

To elaborate the above, consider an example of a pellet induration system, where the airflow distribution is to be determined, and it is literally the airflow being passed on from one unit to the other unit, so the information about the airflow i.e. its flow and pressure will be the state variables. Their values will be exchanged among the neighbouring components. These variables have a dual role in computation; the unit in consideration will treat the values of the state variables (passed on by other units) as parameters or independent variables and will compute them as dependent variable within the unit to pass them on to its neighbours. This is more or less same as in Finite Difference or Finite Element computation, the values of state variables at a node are computed in terms of their values at the neighbouring nodes.

2.4 Mathematical Model for Pellet Induration Systems

Our aim is to develop a model to determine the airflow and temperature distributions in pellet induration systems. As mentioned in section 2.2.1 it can be simulated as a pipe network, hence the model should satisfy the Kirchhoff's laws. These were proposed initially by Kirchhoff, for the analysis of electrical networks, but are equally applicable to any sort of network, which is transporting some conservable continuum medium. These laws are widely used in water, electrical power, natural gas distribution networks, as well as, mine ventilation and other such networks. Pellet induration systems use natural air as the flow medium. Here 'nodes' refer to the internal nodes of the network, consisting of either junctions or regions and 'loops' correspond to the fundamental cycles of the equivalent graph (the graph resulted by ignoring the flow directions in the directed graph). These laws are :

Kirchhoff's First Law or Kirchhoff's Current (KCL) Law:

Total flow entering to a node is same as total flow leaving the node. Mathematically, for *j*th node, with DEG_j incident streams

$$\sum_{i=1}^{DEG_j} a_{ji} \cdot F_i = 0 \qquad (2.3)$$

where a_{ji} is the element of node-stream incidence matrix, with value +1 if ith stream is incoming to jth node; it is -1 if ith stream is an outgoing from node j; and F_i (Kg s^{-1}) is mass flow rate in ith stream.

Generalising it for all N_{int} internal nodes of the network:

$$\sum_{i=1}^{DEG_j} a_{ji} \cdot F_i = 0 \qquad for\ all\ j = 1,2,\ldots N_{int} \tag{2.4}$$

Kirchhoff's Second Law or Kirchhoff's Voltage (KVL) Law: Each network node should have a unique pressure, no matter through which route it is approached. Alternatively, or in the most commonly stated form, the pressure drop across any loop, l, in the network should be zero, i.e.

$$\sum_{i=1}^{TSTR_l} b_{li} . \Delta P_i^l = 0 \tag{2.5}$$

Where b_{li} is an element of loop-stream incidence matrix, having value +1 if the flow direction of ith stream has the same direction as of the lth loop, and value -1 if the directions are opposite; $TSTR_l$, the total number of streams in the lth loop, and ΔP_i is the pressure drop in ith stream of lth loop. b_{li} has a value zero for streams which do not participate in respective loop. The pressure drop for each stream is unique and independent of the loop to which it is contributing. In generalized form, the above equation can be written for all NT_{loop} fundamental loops of the network, i.e.

$$\sum_{i=1}^{TSTR_l} b_{li} . \Delta P_i = 0 \qquad for\ all\ l = 1,2,\ldots,NT_{loop} \tag{2.6}$$

The flow and pressure distributions provided by the network solution must satisfy these two laws. The first law may be directly imbedded into the system as node equation. The second law needs the computation of pressure drops for each of the stream and knowledge of the fundamental loops of the network. Further these streams are composed of serially

connected components, so the pressure drop across each one of these would be required to determine the overall pressure drop in the stream.

The mathematical models for the system components provide the specific relations for this pressure flow inter-dependence. Although the exactness of these component models is desired, but in reality; the identification and comprehension of all involved physical processes, complexity of system, precision of measurements and available experimental data are usual constraints which restrict modellers to live with the approximate component models.

Among different fields, where models are being used, water distribution systems, is one of the oldest and the most mature branches. Ideally, the component model for flow through pipe, should contain all the effects, like, nature of flow, pipe data, pipe fittings, properties of its material, horizontal position and slope, ageing factor and corrosion content etc, how its friction factor will be affected by time and sufficient experimental data to validate all these aspects. Availability of these all is clearly impossible so even to the day, all models start with an approximate but consistent component model (Chansler and Rowe 1990), compute the network and then refine the basic model.

The main obstacle in comprehension of a physical process or a complex system is the complexity of the system. Different factors are attributed to complexity by different authors (see e.g. Chandra et al 1992, Wilkinson and Byers 1993) depending on the modelled environment. In our case there are three main factors which contribute towards the complexity:

- Process complexity: The respective process itself is not well explored yet, say for example, the exact nature of flow through packed bed or through pipe is not known; so to start from simplest possible model and use an incremental approach is the best solution.
- Complexity due to connectivity of components: The components forming the system, are in large number, so are their inter connections. Hence the knowledge of which

component is effecting which, is required, to resolve the interference or 'ripple' effect due to interactions of components.

- Complexity due to incompleteness of information: The information about the processes, as well as the exact nature of components and their geometrical and other data is not completely known.

In modelling the pellet induration system, using the device-centred approach, the components themselves are not that complex, but they are in large numbers, at least not digestible by wetware (human brain) without the aid of computer. Their interactions among each other, adds further complexity to the system. To find the solution of whole system, we will solve it as network, by satisfying Kirchhoff's laws globally. First we discuss the component mathematical models and general principles applied in their formulation.

2.4.1 Simplifying Assumptions

Considering the induration system at a macroscopic level, we make the following assumptions, to simplify the modelling process. Simplifications relating to individual components' equations will be discussed in related sections.

i. Pipes or ducts in the system have different shapes, i.e. having square, rectangular or circular cross sections and are connected in various configurations, in series, parallel or in combination of both. First each non circular pipe is replaced by a circular pipe of same length, but having a diameter which offers an equivalent wet area (see e.g. Francis 1975). Secondly, the combination of pipes between two nodes is substituted by an equivalent pipe which offers the same resistance as the combination. Later in the simulation the data for this virtual pipe would be used (Jeppson 1976).

ii. The pellet induration is a time dependent process, but with extremely varying time constants; the time constant for packed bed movement is in hours, whereas for the gas

flow it is in seconds. So, the bed appears as static to moving gas, or from other perspective, the gas flow will be reaching instantaneously from one end to the other end of the unit. Hence steady state model is assumed for gas flow.

iii. One dimensional flow is assumed in network. It is normal convention for pipe network solution and saves significant amount of computation. Pipes are the main constituent of the network, so as suggested by Ward Smith 1971 and Goldwater and Fincham 1981, one dimensional treatment of flow is valid. This assumption is also conducive to other components of the network.

iv. The variations of air density is small enough that the process gas could be treated as incompressible medium. In terms of natural gas networks, it is a medium pressure system; the measured region pressures range from -30" to +30" of water gauge, which is nearly atmospheric. In SI units, it turns out to be form 94 KPa to 108 Kpa, whereas the atmospheric pressure is 101.325 KPascals. The pressures at fans suction or discharge ends may be beyond the said range, but still the density does not vary appreciably.

v. Ideal gas law is used in derivation whenever required for conversion of density into pressure etc; as the average pressure in the system is about atmospheric. Azbel and Cheremisinoff 1983 states that it is valid up to 10 atmospheres i.e. 10^3 KPa.

Gas flow in the system is governed by the pressure gradient, i.e. gas flows from higher pressure to lower pressure, except in the active components like fans. Fans provide a pressure gain, which is utilised in overcoming the resistance offered to flow by other passive components like pipes, valves, packed beds and leaks.

There could be two ways for the development of equations of individual components of the Networks.

a) To start with the basic laws of physics, i.e. mass, momentum and energy conservation, and derive the network component's equations, with all possible complexities. Then simplify these equations to the practical situation by applying the feasible assumptions.

or alternatively,

b) To benefit from literature and implement the available knowledge by using the well tested, valid, analytical or empirical equations for the respective components.

The results of the model based on (a) would be qualitatively consistent, but from mathematical modelling aspect, these would still require fine tuning of parameters and validation with experiments for quantitative correctness. The validation phase would be impossible, if the experimental data is already not in-hand, because the related industry would not be interested in conducting the experiments for an in-progress model, as it costs time and effort. In cases, like pellet induration systems, the experiments are very costly and difficult to perform due to the hostile environment. The results produced by models based on (b) does not require detailed verification, as the equations or models have already been through this verification & validation cycle. For example Hazen-Williams formula, Darcy-Weisbach equation, Manning equation (Chansler and Rowe 1990 and Yen 1992) for water networks; and Atkinson equation (Wang 1990) for mine ventilation networks; are all different variations of Bernoulli (mechanical energy balance) equation, with coefficients being validated for the respective networks, so each one has its own specific range of application, where it provides correct results. Similarly Osiadacz 1987, gave equations for flow of natural gas through pipes for low-pressure, medium and high pressure regions. Osiadacz also mentions that the predicted flows, even by these widely used and well tested equations, are usually higher than the actual flows, as all friction losses are not catered for in these equations, so he introduced a notion of 'efficiency factor' which modifies the theoretical friction factor, to produce comparable results.

In the present research the combination of both of the above two methods would be used with more emphasis on method (b).

In principle, before incorporating into the model, the respective component equations are thoroughly checked by verifying that these:

- hold under physical laws
- are dimensionally (and also units wise) consistent
- provide practical values for output variables, when fed with practically possible values for the input variables.

In a network simulation, main attention should be given to the computation of network as a whole system. The component equations are the foundation stone of the computation, as these simulate the behaviour of system variables locally in that component. For network simulation we require an algorithm, which should compute the component/basic equations in such a way, that finally not only each of these basic equations is satisfied locally, but also they provide a consistent global pressure, flow and temperature distributions, for the whole system.

The resulting software/code is intended to be used internationally, by iron making and steel making industry, which has not yet adopted a unique system of units, instead each plant uses the units of its own convenience, depending upon its geographical location and installed instrumentation. Hence all user related information will be communicated by the code in plant's units, whereas the computation (and component equation's coefficients) will be carried out in System International (SI) units. The choice of SI units, will keep the model's computation independent of user's units, to which the variables will be translated at the beginning and at the end of computation. It will also enable to compare and incorporate the improved equations from published literature. This (SI) is the main system of units, which has been adopted by most of the academic journals and publishers. To conform to any of the new

set of units in future, an other block of code, or a new subroutine will be needed, which will translate the new set to SI units and back.

2.4.2 Component Equations

Initially when the project was started the zones were treated as a separate entity, but later it was realized that in fact, it is combination of regions and packed bed, so it was replaced by the respective components.

The valves could be simulated as a separate entity, but as these occurred physically with pipes, so these were lumped with the respective pipes, taking into account their specific nature.

Bernoulli's equation is used to simulate the flow through pipes and flow through orifices, for leaks. It is recommended by many authors, like Bird et al 1960, especially for systems with components having single entry and single exit. Nearly every book on fluid mechanics or transport phenomenon covers its development in detail; Massey 1972, Theodor 1972, Azbel and Cheremisinoff 1983 and Douglas et al 1985 could be seen as few examples. Osciadacz 1987, derived Bernoulli's equation starting from Newton's second law and developed it to general flow equation for the transport of natural gas through pipes. He used different friction factors and obtained; Lacey's equation for low-pressure (or distribution) networks, Polyflo equation for medium-pressure networks, Panhandle 'A' equation and Weymouth equation for high-pressure (or transmission) networks. The details for its derivation are skipped as it is widely covered elsewhere.

For temperature distribution computation, most of heat transfer takes place between pellets and air, the process gas, in the packed bed. Another heat distribution code, INDSYS (INDuration SYstem Simulator) Cross 1988 and Cross and Englund 1987, computes the heat transfer at microscopic level, by solving partial differential equations, and taking into account

all complexities of the system and involved chemical reactions. INDSYS requires airflow distribution as input and computes temperature distributions of process gas and of pellets, in the packed bed, in two dimensional space, assuming symmetry in the third dimension. These temperature distributions are averaged out and fed to GASFLO model as input. The global heat conservation cannot be applied to the system, unless INDSYS is taken into account. The overall conservation of thermal energy is achieved by combined iteration of GASFLO and INDSYS. In GASFLO heat conservation is applied locally at junctions, regions and pipes; and isothermal flow is assumed for fans and leaks, i.e. there is no heat loss and so temperature of gas entering to the unit remains unchanged at the exit of the unit.

All equations presented have SI consistent units for their variables and physical constants. The units of the variables are given in nomenclature section at the end of chapter.

Temperature dependent specific heat capacity of gas, given by equation 2.2 will be used in the model. The ideal gas law relation is also used in derivation of the presented equations. Where possible the equations are written in readily computable (FORTRAN) format, with left hand side variable as to be computed in terms of all others known variables on right hand side. The computational procedure and solution algorithms will be discussed in detail in chapter 3.

Junctions and Regions: These are nodes, where two or more streams are meeting. The mass conservation and thermal energy conservation equations are applied here and the mass conservation equation (2.3) is re-written for a node, having DEG_j incident streams, Flow F (kg s^{-1}) and Temperature T (° K) values for all streams are known except for one, say the xth stream, which are to be computed in terms of the other known ones. The right hand side of these equations exclude the xth stream, for which flow and temperature are to be computed so the limits for summation are from 1 to DEG_j-1. The mass balance equation for jth node is

$$F_x = -\frac{1}{a_{jx}} \sum_{i=1}^{DEG_j-1} a_{ji}.F_i \qquad (2.7)$$

where a_{ji} is element of node-stream incidence matrix, having values +1 if ith stream is entering jth node, and -1 if it is leaving, 0 otherwise.

To compute temperature we consider thermal energy balance, i.e. the algebraic sum of thermal energy (or the rate of thermal energy) entering and leaving the node is zero. In the pellet induration process the temperature of process gas vary from atmospheric temperature, 20 °C (≈300 ° K) to 1700 ° C (≈2000 ° K). Young et al 1979, have pointed out that the specific heat for process gas cannot be treated as constant in this range of temperature variation. An experimentally validated relation for its temperature dependence is given by Zografos et al 1987. It covers the temperature ranges from 100 ° K to 3000 ° K, which spans our range of application. Hence this temperature dependent relation, described by equation 2.2 will be used for the computation of specific heat of process gas, Cp (J Kg^{-1} $°K^{-1}$). This will make it non-linear in temperature so its solution will require iteration. The rate of thermal energy q_i for ith stream is $Cp(T_i) * T_i * F_i$ (J s^{-1} or Watts). Hence for jth node the heat balance equation would be

$$Cp(T_x).T_x = -\frac{1}{a_{jx}F_x} \cdot \sum_{i=1}^{DEG_j-1} a_{ji}.Cp(T_i).T_i.F_i \qquad (2.8)$$

Pipes: Pipes are the main unit, responsible for the transport of process gas from one unit to the other. These are very large diameter ducts, closely resembling the airways of mine ventilation systems. Applying the simplification (i) of section 2.4.1 all pipes occurring physically between two nodes of the network; of any shape and connected in any serial or parallel configuration; can be replaced by a circular pipe, for which the equations are

developed. These pipes are well insulated, to conserve thermal energy but still a loss of few degrees of temperature is noticed at the ends.

Assuming the flow is friction-less, isothermal and incompressible, its steady state behaviour is represented by Bernoulli's equation. It is a high driving system, and flow is always turbulent, so the resistance offered to flow due to turbulence, as suggested by Azbel and Cheremisinoff 1983, is used. The friction factor λ, uses Reynolds number, ***Re***, which is further computed from mass flow rate per unit area G (Kg s^{-1} m^{-2}) for the respective pipe. The given form of Bernoulli's equation resembles to the equations developed by Azbel and Cheremisinoff 1983 and Lugt 1983; and to the well known Atkinson's equation used for mine ventilation networks, as quoted by Bruce and Koenning 1987, Wang 1990 and Moll and Lowndes 1992, for incompressible, unidirectional and steady state flow of air in airways. In the following equations the subscripts 'pipe','in' and 'out' refer to respective pipe, its input (up-stream) and output (down-stream) ends of the pipe.

$$G = \frac{F}{A_{pipe}} \tag{2.9}$$

$$Re = \frac{D_{pipe} \times G}{\mu} \tag{2.10}$$

$$\lambda = 0.0123 + \frac{0.7544}{Re^{0.38}} \tag{2.11}$$

$$P_{out} = P_{in} - \frac{2 \times \lambda \times L_{pipe}}{D_{pipe} \times \rho_{air}} \times \alpha_{pipe} \times G^2 \tag{2.12}$$

where D_{pipe}, L_{pipe} and A_{pipe} are diameter (m), length (m) and cross sectional area (m^2) of pipe; P is pressure (Pascals or N m^{-2}); ρ is density (Kg m^{-3}) of process gas; μ is dynamic viscosity (Kg m^{-1} s^{-1}) and α is pipe dependent (dimensionless) calibration or efficiency factor (see Bhave 1991 and Osiadacz 1987).

For temperature computation, the over all heat loss by the pipe would be combination of heat loss; due to convection, by process gas to the pipe internal surface, and due to conduction within the pipe from internal surface to external surface and in the layers of insulation. For steady state case both of these losses will be same i.e. all the heat convected to the pipe would be conducted to the atmosphere. Bird et al 1960, pp 283-288, has developed the case for composite cylindrical pipe, which can be directly applied to our scenario. Pipes are insulated but the material properties of the insulation, its thickness etc are not known; so its inclusion in implementation is temporarily postponed, hence only single pipe with process gas flowing inside it, is modelled. The length and cross section wise, approximate temperature profiles for pipe, are shown in Figure 2.7.

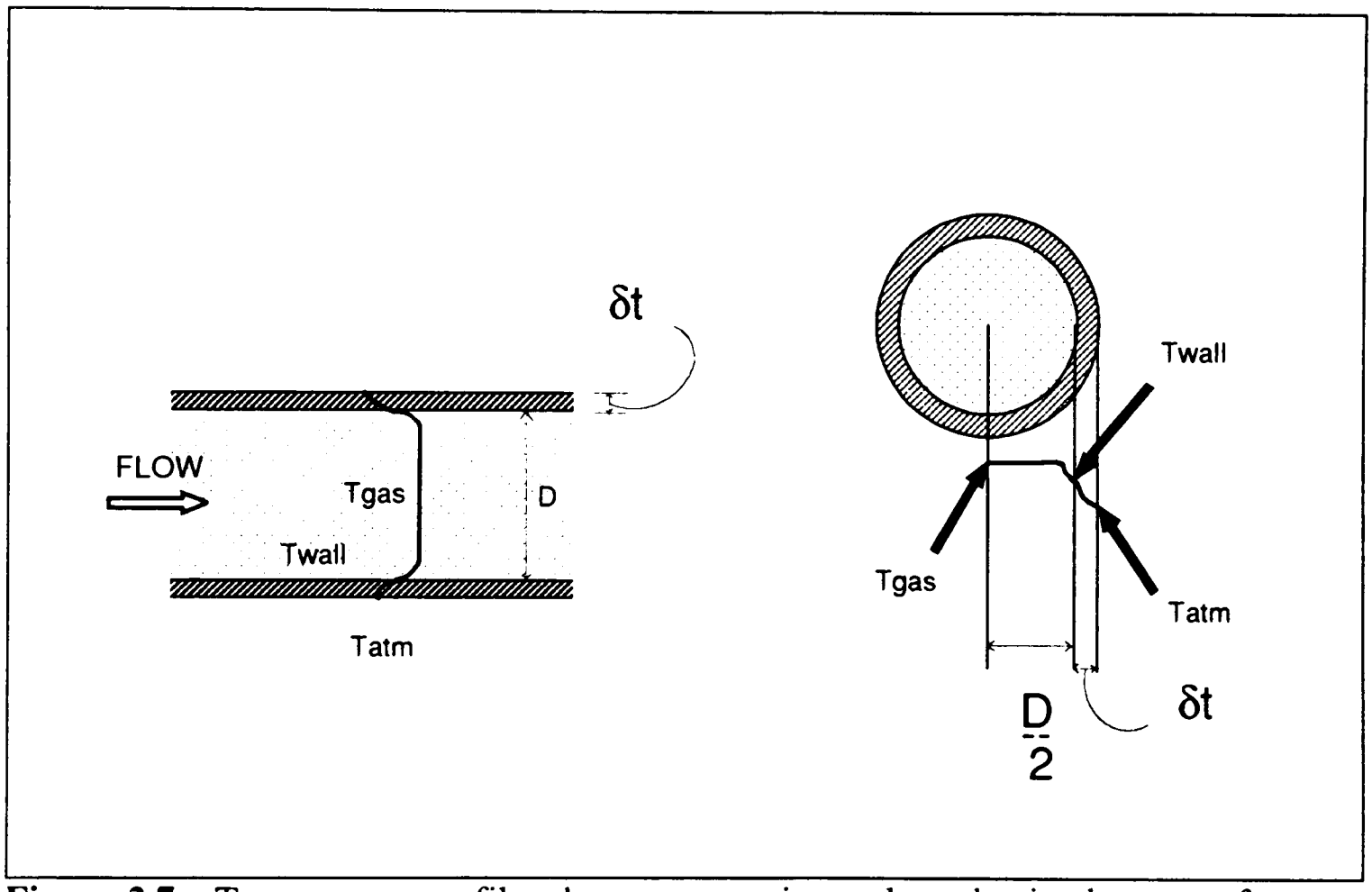

Figure 2.7 : Temperature profiles due to convective and conductive heat transfer

Assuming that the process gas has fixed temperature, T_{gas} (°K), which can be taken as average of the temperatures of the gas entering and leaving the pipe, i.e. of T_{in} , T_{out} as the temperature variation through the pipe is not more than few degrees. Secondly constant temperature T_{wall} could be assumed for internal wall, this cancels out during derivation and

does not appear in the following equations and ambient temperature T_{atm} is assumed on the pipe (external) surface. Simplifying the equation given by Bird et al, for single layer pipe, we have a rate of heat loss from pipe q (J s^{-1} or Watts) as

$$q = \frac{2 \times \pi \times L_{pipe} \times (T_{gas} - T_{atm})}{\frac{2}{D_{pipe} \times h_{gas}} + \frac{\ln\left(1 + \frac{2 \times \delta t}{D_{pipe}}\right)}{k_{pipe}}} \qquad (2.13)$$

where Temperatures are in °K, D_{pipe} (m) is internal diameter of pipe and δt (m) is pipe thickness, k_{pipe} (Watt m^{-1} $°K^{-1}$) is thermal conductivity of pipe and h_{gas} (Watt m^{-2} $°K^{-1}$) is heat transfer coefficient of process gas. For the inclusion of insulation into model we would have an other term in the denominator of the above equation.

The overall heat transfer is dependent on both processes, in abstract it is like conductance through the series of resistances, so the highest resistance governs the overall transfer, in this case it is the convective part which is more effective. In practice the value for heat transfer coefficient, h_{gas}, should be provided by the experimentalists and fed into the model. In the absence of that we used the approach adopted by Fogiel 1984, and Kay and Nedderman 1985, and computed its approximate value from the definition of dimensionless numbers. Physically, the ***Nu*** Nusselt no, is the ratio between actual heat loss and conductive heat loss; the ***Re*** Reynolds no. equation 2.10, is the ratio between mechanical and viscous forces; and the ***Pr*** Prandtl no, is the ratio between momentum and heat transfer by molecular action. In terms of our variables, ***Nu*** and ***Pr*** numbers are

$$Nu = \frac{h_{gas} \times D_{pipe}}{k_{gas}} \qquad Pr = \frac{\mu \times Cp_{gas}}{k_{gas}} \qquad (2.14)$$

Where Cp_{gas} (J Kg^{-1} $°K^{-1}$) is specific heat capacity of process gas at constant pressure, k_{gas} (Watts m^{-1} $°K^{-1}$) is thermal conductivity of gas, other variables have same meanings as

defined previously. Fogiel 1984 has given the following empirical result, relating these numbers for turbulent flow.

$$Nu = 0.023 \times Re^{0.8} \times Pr^{0.4} \tag{2.15}$$

Zogrofos et al 1987, has given a temperature dependent relation for thermal conductivity of air. They obtained it by fitting curves to experimentally available data. Like Cp its range also conforms to the temperature range of pellet induration system, so it was used to obtain more realistic results.

$$\begin{aligned} k_{gas}(T) = \; & -7.488\times10^{-3} + 1.7082\times10^{-4}\times T - 2.3758\times10^{-7}\times T^2 \\ & + 2.2012\times10^{-10}\times T^3 + 9.46\times10^{-14}\times T^4 \\ & + 1.5797\times10^{-17}\times T^5 \end{aligned} \tag{2.16}$$

Using this relation for input temperature $T = T_{gas}$; the variables k_{gas}, Cp_{gas}, are determined; which are used to evaluate Pr, Re and Nu from the above equations 2.14, 2.10 and 2.15 respectively. Using the definition of Nu number equation 2.14, h_{gas} and finally q from equation 2.13, are determined.

Now considering overall heat balance for pipe, assuming the work done by the gas as zero, the total heat entering into the pipe would be heat leaving the pipe with gas and due to heat loss from its surface.

$$F_{in}\times Cp_{in}\times T_{in} = q + F_{out}\times Cp_{out}\times T_{out} \tag{2.17}$$

As by mass conservation $F_{in} = F_{out} = F$, similarly we can safely assume $Cp_{in} = Cp_{out} = Cp$ and substituting the values for q we can find T_{out} in terms of T_{in} or vice versa.

$$T_{out} = T_{in} - \frac{q}{Cp\times F} \tag{2.18}$$

Substitution of Cp by equation 2.2, will make it non-linear in temperature, which is resolved by iteration.

Packed Beds : The gas flow through packed bed is well represented by Ergun's equation. Bird et al 1960 has shown its comparison to other equations and concluded its best fit to experimental results and its applicability for wider range. It was originally presented by Ergun 1952. Now it is widely used in industry. Fenech et al 1987, Ingham et al 1988, Osinki et al 1989, Patel and Cross 1989 and Cross et al 1990 has mentioned its various forms in use. Most of the authors have used superficial velocity, i.e. the velocity of gas when packed bed is not present, instead of actual velocity. Resolving the form given by Bird et al, substituting actual gas velocity, replacing density ρ of gas in terms of atmospheric pressure P_{atm} and using gas law with gas constant for air, R (286.68 J Kg^{-1} $°K^{-1}$); and using the mentioned units; the equation reduces to

$$P_{out} = P_{in} - \frac{H_{bed}}{d_{par}} \times \frac{R\,T_{bed}}{P_{atm}} \times \frac{(1-\epsilon)}{\epsilon^3} \times G \left[\frac{150\,(1-\epsilon)\,\mu}{d_{par}} + 1.75\,G \right] \qquad (2.19)$$

where H_{bed} (m) is bed height, T_{bed} (°K) is temperature of gas in the packed bed, d_{par} (m) is particle or pellet's diameter and ε (dimensionless) is voidage, i.e. ratio between volume of voids and volume of the bed.

The first term in the brackets represents the contribution to the pressure drop due to viscous effects whereas the second term is the contribution due to inertial effects. Ours is the 'high driving' system as mentioned by Fenech et al 1987, so the viscous contribution is negligible as compared to its inertial counterpart, so first term could be ignored, and hence

$$P_{out} = P_{in} - 1.75 \frac{H_{bed}}{d_{par}} \times \frac{R\,T_{bed}}{P_{atm}} \times \frac{(1-\epsilon)}{\epsilon^3} \times G^2 \qquad (2.20)$$

This equation is used in the model for simulation of packed bed. It is quadratic in terms of air flow.

The packed bed is the main unit responsible for all heat transfer between the process gas and the pellets. It embeds all sorts of complexities of air flow in porous medium to chemical reactions, so the thermal computation of bed is simulated separately by the other

model INDSYS. The process gas temperatures required for GASFLO computation, are computed by INDSYS and are fed-in as parameters, air flow distribution required by INDSYS is computed in GASFLO and supplied to INDSYS.

Leaks : These are un-wanted but un-avoidable flows of process gas between the adjacent regions and between the system and atmosphere. These adversely effect the heat balance of the system. Ironically due to hostile conditions and instrumentation constraints these cannot be measured. For un-obstructed movement of packed bed few centimetre clearances above and below the bed are required, which ultimately results in leaks flow.

It is experimentally observed that a significant amount of process gas is sucked into the system, from atmosphere, through these leaks, which degrades the system efficiency. Without determining the sources, and their contribution to this additional flow it is impossible to minimise it or analyze its effects on the overall system efficiency.

Physically a leak is flow through a small cross sectional area linking the two regions. It is governed by pressure gradient between the two, from higher pressure to lower pressure. It is similar to flow through orifice and simulated by the orifice equation.

Applying Bernoulli's equation to the pipe of diameter D_1 (m), shown in Figure 2.8, with a sharp edge orifice of diameter D_2 (m), assuming no energy loss and pipe being horizontal, we have

$$\frac{P_1}{\rho g} + \frac{v_1^2}{2g} = \frac{P_2}{\rho g} + \frac{v_2^2}{2g}$$

where P_1, P_2 (Pa) are pressures and v_1, v_2 (m s^{-1}) are velocities of gas at section '1' and '2' respectively; and g (m s^{-2}) is acceleration due to gravity. Applying continuity equation i.e. mass entering at section 1 is same as the mass leaving the system at section 2 in steady state.

$$A_1 v_1 = A_2 v_2 \quad \Rightarrow \quad v_1 = \frac{A_2}{A_1} v_2$$

where A_1, A_2 (m^2) are cross sectional areas at the respective sections. So substituting v_1 in Bernoulli's equation, replacing areas in terms of diameters and rearranging we get

$$v_2 = \sqrt{\frac{2}{\rho} \frac{(P_1 - P_2)}{1 - \left(\frac{D_2}{D_1}\right)^4}}$$

Assuming that diameter of orifice, D_2 is very small as compared to the pipe diameter, D_1, so $(D_2 / D_1)^4 << 1$, hence the term in the denominator can be ignored. To find out the Flow through the section 2, F_2 (Kg s^{-1}), we multiply both sides by ρ and cross sectional area A_2, we get

$$F_2 = A_2 \sqrt{2\rho (P_1 - P_2)}$$

The practically measured flow is smaller than the theoretically calculated flow, about 60% for sharp edged orifice (Douglas et al 1985 pp 162-167). The two contributing factors for this reduced measured flow are; there is some loss of energy due to contraction in contrast to our initial assumption that there is no loss of energy, so the velocity at orifice is in fact less than v_2; and the flow profile further converges to say point 'o' commonly known as *vena contracta*, so the cross sectional area of flow stream is A_o, which is smaller than the area A_2. Hence the net flow through the orifice is C_{dis} times the theoretical flow F_2. Thus

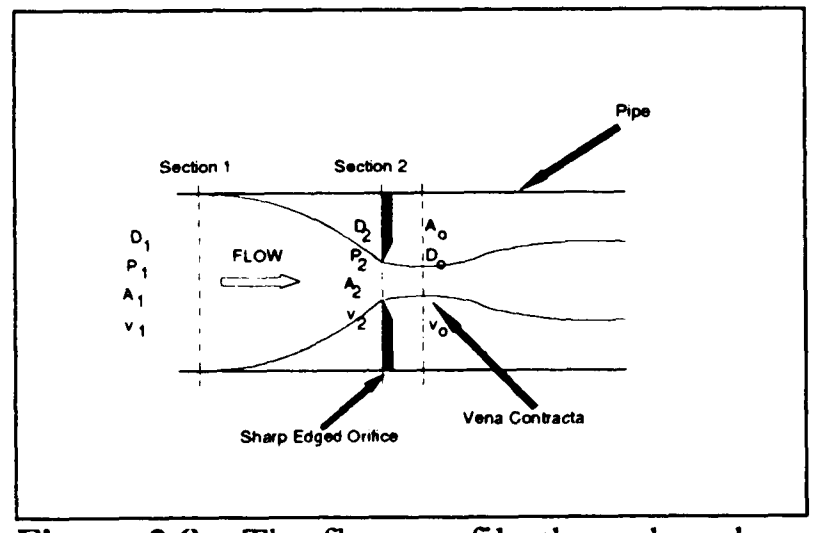

Figure 2.8 : The flow profile through a sharp edged orifice.

$$F_{orf} = C_{dis}\, A_2 \sqrt{2\,\rho\,(P_1 - P_2)}$$

where C_{dis} is discharge coefficient for the orifice, and its values are dependent on geometric configuration of the orifice and on the nature of flow, i.e. $C_{dis} = f\,(Re,\, D_2 / D_1)$. It is experimentally determined and for standard configurations, its values are given in the literature.

The above equation can be generalised for leaks, since the aperture and region cross sectional areas are of comparative sizes as used in the development of above equations, so re-writing the above equation for leaks;

$$P_{out} = P_{in} - \frac{F_{leak}^2}{2\,\rho\, C_{dis}^2 A_{leak}^2} \qquad (2.21)$$

where P_{in} and P_{out} are the up-stream and down-stream region pressures.

The thermal computation for leak flows are simple assignment, as there is no heat loss, so the temperature of the gas entering to the region through a leak would be same as its temperature at the source region, from where it is coming, hence

$$T_{leak} = T_{in} \qquad (2.22)$$

Valves : Valves are used by operators to route process gas in desired paths of the system and to control its magnitude. Like other pipe fittings, valves also cause increased resistance to flow, by restricting the cross sectional area of respective pipe. The opening and closing of valve in a flow stream causes disturbance, which, in reality, is a transient process, but in our steady state model, it is assumed that all the changes in valve positioning have been carried

out before the start of simulation and these remain constant during the simulation. After viewing the results the user can change the valve positioning and restart the simulation.

The valves can be modelled either; as a separate entity, like leaks using a developed valve equation (see Osiadacz 1987), where valve cross sectional area could be adjusted by the user; or alternatively, these could be lumped with the pipe model. In principle, valves offer resistance to flow, so can be dealt in similar way, like other pipe fittings, as treated by most of the authors.

This lumping can be carried out in two ways. One is by finding the head loss or pressure drop, caused by the valve individually, using head loss coefficient K, and the notion that it is proportional to square of flow velocity, and then adding this pressure drop to the pressure drop of straight pipe of length l_{orig}. This will be the overall pressure drop offered by the pipe including valve.

The other method is conceptually simple and widely used to accommodate pipe fittings in flow computation. Here the fitting or valve in the pipe, is replaced by a fictitious pipe (of same type, characteristics and diameter) of specific length l_e, known as equivalent length. In principle this fictitious pipe offers the same resistance to flow as the corresponding valve or fitting. So the new length l_{new}, to compute pressure drop would be

$$l_{new} = l_{orig} + l_e \tag{2.23}$$

The equivalent length of standard pipe fittings are given in the literature (see Massey 1972, Azbel and Cheremisinoff 1983, Daugherty et al 1985 and Douglas et al 1985). These are usually given in terms of number of diameters, (L/D), so l_e the equivalent length (m) of a pipe of diameter D_{pipe}, for the given fitting could be evaluated by

$$l_e = \left(\frac{L}{D}\right) D_{pipe} \tag{2.24}$$

Experimentally determined values for head loss coefficients; in terms of K and in terms of (L/D); for the standard pipe fittings and valves are well covered in the literature. In Table 2.1, values for some valves are quoted, as referred by Azbel and Cheremisinoff 1983.

The exact nature and the characteristics of the valves, used in the induration system are not known, so we assume that these could be of any of the Gate, Globe or Check valve type. To provide a handle to the user to open and close the valve, we use the following transformation for globe valve:

$$\left(\frac{L}{D}\right)_{globe} = \begin{cases} \dfrac{670.92\,r + 271.57}{\pi\, r^2} & \text{for } 0.1 < r \le 1.0 \\ 1.08 \times 10^4 & \text{for } 0.0 \le r \le 0.1 \end{cases} \tag{2.25}$$

where r is (dimensionless) radius of circular cross section of valve opening. This equation gives the value of (L/D) as 300 for fully open i.e. $r = 1.0$ and as 475 for half open position i.e $r = 0.707$. This is derived using the concept that resistance offered is inversely proportional to valve opening area. The fixed value for the domain, $0.0 \le r \le 0.1$, is merely to avoid the zero divide exception error for fully closed valve. For other valve types the numerical coefficients would change.

Knowing the value of (L/D) and pipe diameter D_{pipe} of associated pipe, the equivalent length l_e is computed and added to the original length of pipe. This total length is used for pipe's computation, using Bernoulli's equation, whereas for temperature computation the original pipe length is used. In valve computation it is assumed that there is no heat loss, the temperature of gas leaving the valve is same as it entered the valve.

Table 2.1 : Friction loss coefficients for turbulent flow through valves

Valve Type	Head Loss Coefficient *K*	*(L/D)*
Gate Valve		
Wide Open	0.17	9
Half Open	4.5	225
Globe Valve		
Wide Open	6.0	300
Half Open	9.5	475
Angle Valve		
Wide Open	2.0	100
Check Valve		
Ball	70.0	3500
Swing	2.0	100

Fans : These are the only active components responsible for gas flow in the system. These are identical to the compressors in natural gas pipe networks, pumps in water distribution networks, or fans in mine ventilation networks. In contrast to their equivalents, where graphical characteristic curves are used to simulate, we use the exact equation to model the behaviour of fans more realistically. We assume isothermal compression, that is any heat generated due to compression of gas, is removed from the system, by the fan cooling assembly, thus keeping the fan and gas temperature unchanged.

Azbel and Cheremisinoff 1983, have developed expressions for fans, blowers, compressors and vacuum pumps, classifying them according to compression ratio P_{out} / P_{in}. The following fan equation is selected from those proposed, in accordance with the range constraints of our system, and it is modified according to the variables used. It shows that pressure gain is proportional to the fan electrical wattage N (Watts) and is inversely proportional to the throughput, mass flow rate of gas passing through the fan, F_{in} (Kg s^{-1}).

$$P_{out} = P_{in}\, e^{\left(\frac{\eta_{iso} N}{R T_{in} F_{in}}\right)} \qquad (2.26)$$

P_{in}, P_{out} are suction and discharge pressures (Pascals); η_{iso} (dimensionless) is isothermal efficiency of fan, with suggested range of 0.64 to 0.78; R (J Kg^{-1} $°K^{-1}$) is gas constant for air, in the used units it is 286.68; F_{in} (Kg s^{-1}) is flow, and T_{in} (°K) is temperature of gas entering the fan.

For a fixed input pressure, pressure gain verses throughput graph is similar to the characteristic curves for mine ventilation fans (Hall 1987 and Wang et al 1988), pumps and compressors (Osiadacz 1987 and Fincham 1971). The used equation responds automatically to the temperature and flow changes in the process gas passing through the fan. Another aspect worth noting is the exponential relationship between pressure gain and flow rate, as compared to their linear or quadratic dependence in other network components.

Like leaks equation 2.22, the temperature computation has simple assignment equation

$$T_{out} = T_{in} \qquad (2.27)$$

where T_{in} (°K) is temperature of process gas at fan's entrance.

2.4.3 Unit Computation

Using a device centred or unit based approach, each of the last section units (except valves) will be computed independently, in the order of their connectivity. The state variables, F, P and T computed by one unit will be passed on to the next unit, and so the process will be repeated for all units in the network, till the sink boundary is reached.

Stream like units; pipe, leak, bed and fan; have a single entry and single exit, and fixed flow. So generically these will have a combination of five global variables. These are

P_{in}, T_{in}, F_{in}, P_{out} and T_{out}. The variables with known values are treated as parameters for the respective unit. Most of the time, variables are known at one end, and the other end variables are computed. In some cases both end temperature and pressures are known and flow is computed for the unit.

Nodes (i.e. junctions and regions) are more complex, and have multiple entry and exit streams. These will have $2n + 2$ variables (and parameters) for n incident streams; comprising of F_i , T_i for $i=1,2 \ldots n$ incident streams, and P_j , T_j for the respective *jth* node. Referring to the two equations for node unit, the two variables for any of the *xth* stream will be computed using the (known) values of all others. The *xth* stream will be mostly the out going stream, and so the node temperature T_j will be computed, and all outgoing streams temperature T_x will be initialized with it. Whereas, the node pressure P_j will in fact be computed during the computation of one of its incoming streams, and will be initialised during the node's computation. Algorithms described in chapter 4 will further elaborate the computation strategy, and the specification of *xth* stream.

Considering each unit, the above described solution strategy satisfies the *"n equations for n variables"* constraint. The non-homogeneity of the units can be seen by the respective pressure-flow equations given in the last section; this might require different numerical schemes for different units.

The objectives of the present work are that the developed model, should:

- work as a frame-work, where the unit's mathematical model and numerical scheme could be easily swapped by better alternatives, whenever these are available.
- be robust and should provide reasonable answers to physically valid inputs.
- be fast, so that it could be used for operator training and control purposes. This indirectly implies that computational load should somehow be minimised, by using simple, reliable numerical schemes.
- address to wider application domain and useable on high end PCs.

The above goals could be achieved by embedding the appropriate numerical schemes, well suited to the nature of the specific equations of the unit, i.e. the mathematical model of the unit; and computing the associated global variables in terms of parameters locally. The other advantage is that problem size is much reduced, instead of solving single large set of equations now, smaller sets of fewer equations are solved simultaneously (Sargent 1978). Finally, passing these converged results on to other units, so as the minimize the overall instability in the system.

2.5 Numerical Scheme for Local Computation

In this steady state model, each unit is represented by set of algebraic linear and non-linear equations. These are to be solved simultaneously. There exist a wide variety of numerical methods to solve such sets, each having its own advantages and shortcomings. Since different units have different equations, and these are solved independently, so in principle each unit could use its own local computation scheme, as suited to the nature of its equations.

The linear equations could be solved by method of substitution, that is evaluating the desired variable in terms of the known variables (either by its parametric values or by values known from previous iteration).

The solution of non-linear equations is harder. For this we use the ***One Point Iteration Method***. This method is referred to by many names, like *Iteration Method, Fixed-Point Iteration* or $x = g(x)$ *Method*, by different authors. It is covered by nearly every book on Numerical analysis (e.g. see Smith 1979, Gerald and Wheatley 1989 and Burden and Faires 1989). It is simple, robust and gives converged results in few iterations.

According the this method, the solution or approximation of a non-linear equation

$$f(x) = 0 \quad for \quad x \in R$$

is obtained by re-writing it as

$$x_{i+1} = g(x_i)$$

where $g(x_i)$ is such that, $g(x_i)$ and $g'(x_i)$ are continuous functions for all $x_i \in R$. There could be many such functions, which can generate different sequences of $\{x_i\}$, but those satisfying

$$| g'(x_i) | < 1$$

for all $x_i \in R$ would give convergent results.

In our model as all equations and the variable domains are known, so all non-linear equations could easily be approximated by suitable $g(x)$ functions, and the set of equations for a unit is solved iteratively. Later if some better method emerges then that can replace this method, for all or some of the units.

2.6 Nomenclature

A	Cross sectional area	m^2
Cp	Specific heat capacity of gas at constant pressure	J Kg^{-1} $^oK^{-1}$
D	Diameter	m
d_{par}	Diameter of particle or pellet	m
DEG_j	Total degree of jth node i.e. no of incident streams on the node	#
F	Mass flow rate of process gas	Kg s^{-1}
g	Acceleration due to gravity	m s^{-2}
G	Mass flow rate of process gas per unit area	Kg s^{-1} m^{-2}

h	Heat transfer coefficient of process gas	W m^{-2} $°K^{-1}$
H	Height of packed bed	m
k	Thermal conductivity of (pipe) material	W m^{-1} $°K^{-1}$
$K_{fitting}$	Head loss coefficient for (pipe) fitting	m
L	Length of the unit	m
Nu	Nusselt number	#
P	Pressure of process gas	N m^{-2} or Pascals
Pr	Prandtl number	#
q	Rate of thermal energy or heat loss	J s^{-1} or Watts
Q	Volumetric flow of process gas	m^3 s^{-1}
R	Gas constant for process gas (air)	286.68 J Kg^{-1} $°K^{-1}$
Re	Reynolds number	#
T	Temperature of process gas	°K
v	Velocity of process gas	m s^{-1}

Greek letters

α	Calibration or efficiency factor for pipe	#
δ **t**	Pipe thickness	m
ε	Packed bed voidage	#
η_{iso}	Iso-thermal efficiency of fan	(0.6 - 0.9) #
	Dynamic viscosity of process gas	Kg m^{-1} s^{-1} or Poise
ρ	Density of process gas (air)	Kg m^{-3}

Subscripts:

in / out	Up-stream / Down-stream end of the unit
pipe / bed / fan / region / junction	Refers to the respective unit of the network

Chapter 3
Solution Algorithms

3.1 Air Flow and Temperature Distributions Coupling

Considering the component models presented in the last chapter, the resulting equations could be categorised in to two sets; the one relating to the computation of flow and pressure or simply the flow distribution and the other relating to the computation of temperature distribution. The first set is comprised of equations 2.7, 2.9 - 2.12, 2.21 and 2.26 which are used for the computation of flow in junctions, pipes, leaks and fans respectively; whereas the second set consists of equations 2.8, 2.13 - 2.18, 2.22 and 2.27 which provide the temperature distribution in the respective components.

The packed bed model uses the Ergun equation (2.19) to relate the pressure, temperature and flow. The heat transfer between pellet and process gas takes place in packed bed, taking into account all chemical reactions and material properties, the resulting mathematical model for heat transfer (and hence temperature computation) is very complex. It is solved for separately and the computed temperature of process gas are fed into the GASFLO model as a parameter. The temperature of gas in equation 2.19 can be treated as constant. The algorithm for temperature computation will be discussed in section 3.8 and the import of bed temperatures will be discussed in section 5.5. The fan equation 2.26 is also temperature dependent, but since in the pellet induration system the fans are located at either suction end, where they suck in natural air which is always at ambient temperature; or within

the network at downstream to some zones (i.e. packed bed), where the temperature of process gas passing through the fan would be governed by the average temperature of the respective packed bed. The temperature distribution is mainly effected at junctions or regions (equation 2.8), where mixing of different streams occurs and in pipes (equations 2.13 - 2.18) where some heat loss to the atmosphere is taking place. The other equations of the set are simple assignment equations, describing no change in temperature. These junction and pipe related equations are dependent on the values of flows in the associated streams.

Thus, there exists, a coupling between flow and temperature distributions, but the dependence of flows on temperature is quite weak, whereas the temperatures are strongly dependent on flow distribution. Fincham and Goldwater 1979, have also suggested computation of temperatures in the outer loop whereas the flows and pressures are computed in the inner loop.

This inter-dependence can be exploited and the two sets of equations can be decoupled, then each one can be computed in isolation. The flow distribution computation uses fixed parametric values of temperatures for packed bed and boundary conditions, whereas the temperature distribution computation uses the recent computed converged flow distribution. In case, if this strategy is not applied, and the system is solved simultaneously for both distributions, then the temperature equation 2.8 for junctions, may give fluctuating values for stream temperatures for some intermittent non-converged stream flows, and may result in overflow errors.

The present model (GASFLO) is intended to be used as:

- an aid for operator's training, so it should respond to; varying inputs to boundary conditions, changes in components' parameters e.g. different valve openings, fan characteristics and leak clearances; and predict the flow and pressure distributions for his guidance;

- a potential tool for a control system so that the flow and pressures at critical components could be displayed and the operator be appraised of any emergency situation before its occurrence and if possible could assist him for remedial measures; and
- a tool for the optimization of the whole process to produce high quality product at minimum cost. In this regard it will be incorporated into the heat distribution model as a part.

Considering the above goals, the computation of a reliable flow distribution in the network has primary significance and this is also needed for temperature distribution computation. Accordingly the main emphasis in describing algorithms and solution procedures would be, on the evaluation of flow distribution. Later it will be extended to interact with an already existing heat distribution model INDSYS (see Cross and Englund 1987, Cross 1988), which would require flow as well as temperature distribution of streams, so the requirements of this later stage are also taken into account and necessary provisions for temperature computation in algorithms are also made.

3.2 Air Flow Distribution Computation

In section 1.1 an overview of existing algorithms for the network computation is given. In section 2.4 the sub-models for the basic components of pellet induration networks are discussed and conforming to their heterogeneous nature, the unit-based or device-centred approach (section 2.3.4) is selected from the other available approaches.

The basic principles for the solution of the network remain the same, but the algorithms described elsewhere are for *nodal* or *loop* methods, which are based on a process-centred approach where the networks also had homogeneous components. Hence such algorithms can not be applied here without modification.

In practice, pellet induration system networks have multiple sources and multiple sinks. Natural air being the flowing medium is sucked into the system by large fans and after use it is pumped out to the atmosphere. Also some parts of the system are exposed to atmosphere, from where the air can *leak* into or out of the system depending upon its pressure at this location. The flow at each one of these leaks is not known and is to be determined. Pressures and temperatures at sources, where the flow is sucked in, are known and have ambient values. The installed instrumentation provides the values for the flows going out of the system to the sinks through main pipes or stacks. Fans are used to pump out the process gas, and this gas is being used for heat transfer from fired pellets to cool and wet pellets, so it will have varying temperature, and also the pressure of gas at exit from the system will be higher than atmospheric pressure. Hence the boundary conditions of our networks are :

- Fixed pressures and temperatures at source boundary nodes
- Fixed flow rates of the streams entering sink boundary nodes
- Fixed pressures and temperatures at atmospheric nodes adjacent to leaks, where the air is being sucked into the system
- Fixed temperatures of process gas at packed beds

All these values of state variables along with the other component related data will be fed in as parameters for a particular run of the simulation. The model then predicts the values of state variables; pressure, flow and temperature; for all components of the network.

3.2.1 Mathematical Formulation

Here we formulate the above stated airflow distribution problem in terms of mathematical equations. Any single-source network of N_{TNDS} total nodes, would have N_{TNDS} - 1 branches or streams in its tree, and if there are N_{TSTR} total streams in the network then there would be N_{TSTR} - N_{TNDS} + 1 fundamental closed loops and same number of cotree branches. In pellet induration system networks there are atmospheric nodes, which have fixed

pressure, thus for N_{ATM} atmospheric or fixed pressure nodes there would be N_{ATM} pseudo or open loops each one linking the respective atmospheric node to the reference or source node. Hence the total number of fundamental loops would be N_{TLOOPS}, where

$$N_{TLOOPS} = N_{TSTR} - N_{TNDS} + 1 + N_{ATM}$$

As the pellet induration system networks are multiple source and multiple sink networks, for N_{SRC} sources, the total fundamental loops would be

$$N_{TLOOPS} = N_{TSTR} - N_{TNDS} + N_{SRC} + N_{ATM}$$

The nodes can be categorised as internal and external nodes according to their connectivity (see section 2.2.3). The internal nodes correspond to the pipe junctions and zone regions whereas the external nodes are sinks, sources or atmospheric nodes. At internal nodes the Kirchhoff's Current Law (KCL) must be satisfied. The number of nodes in these categories are related as

$$N_{TNDS} = N_{INT} + N_{EXT} = N_{INT} + N_{SRC} + N_{SINK} + N_{ATM}$$

or

$$N_{TNDS} - N_{SRC} - N_{ATM} = N_{INT} + N_{SINK}$$

So total number of fundamental loops in a N_{SRC} network would be

$$N_{TLOOPS} = N_{TSTR} - N_{INT} - N_{SINK}$$

To compute flow distribution in the network, we would solve

- KCL equations for all N_{INT} internal nodes i.e.

$$\sum_{i=1}^{N_{TSTR}} a_{ji} F_i = 0 \qquad for\ j = 1,2,\ldots N_{INT} \tag{3.1}$$

where $a_{j\,i}$ is an element of node-stream incidence matrix and F_i is flow in the ith stream; and

- Kirchhoff's Voltage Law (KVL) equations for all N_{TLOOPS} fundamental loops

$$\sum_{i=1}^{N_{TSTR}} b_{li}\, \Delta P_i = 0 \qquad for\ \ l = 1,2,\ldots N_{TLOOPS} \tag{3.2}$$

where $b_{l\,i}$ is an element of loop-stream incidence matrix and ΔP_i is pressure drop in the *i*th stream.

The stream flow F_i and pressure drop ΔP_i in each stream are related by respective stream components' mathematical models i.e.

$$\Delta P_i = \psi_i(F_i) = \sum_{K=1}^{N_{COMP}} \varphi_{K,i}(F_i) \qquad for\ i = 1,2,\ldots N_{TSTR} \tag{3.3}$$

where ψ_i represents the overall function of the ith stream, $\phi_{K,i}$ corresponds to the Kth component (mathematical model) of the ith stream and N_{COMP} is number of components in the ith stream. As discussed in section 2.4, these component models are non-linear and heterogeneous by nature. In fact for some components $\phi_{K,i}$ is equivalent to a set of equations, whose complexity depends on the nature of the component and the physics embedded into its model. These models are completely described in section 2.4.

Now to determine the airflow distribution in the network, equations 3.3 is substituted in equation 3.2 and total of N_{TSTR} - N_{SINK} equations (formed by equations 3.1 and 3.2) are solved to find flow in N_{TSTR} - N_{SINK} streams i.e. all streams of the network less the streams connected to sinks, whose flows are known as boundary conditions. Knowing flows in all streams and pressure at the source nodes (which is another boundary condition), equation 3.3 plus the network connectivity (i.e. information about the path linking the respective node to the source), we can provide the pressure at each of the internal node.

3.2.2 Open and Closed Networks

The networks, in general, can be of two types:

Open Networks: Networks which do not have any loop or mesh, are named as open networks. These transform to tree type structure, with source at the route node and sinks at leaf nodes. Their solution is straight forward and can be described by the following steps:

Step 1. As the flows at the sinks are known, using Kirchhoff's current law (KCL), at each of the internal nodes the incoming flow can be computed as sum of out going flows;

Step 2. Knowing the flow in all streams of the network (by Step 1) and pressure at source node; the pressure at downstream end (or down-end) of each of the streams leaving the source node can be computed. These down-end pressures become the node pressure for the respective successor node and upstream end (or up-end) pressures for the streams leaving these nodes. Then the streams at next level are picked up and computed. This process is repeated until all the streams of the network have been exhausted;

Step 3. Knowing the flow in the network streams and temperatures at source node and of process gas at packed beds, the temperature of out going stream of each of the nodes can be computed.

The computation of the flow distribution (step 1) is carried out in sink to source direction that is the direction opposite to the natural flow; whereas the pressure and temperature distributions (the steps 2 and 3) are computed in source to sink direction.

If the network is composed of multiple sources, then it could be treated as multiple single sources networks, each having a tree structure and the above computational procedure can then be applied to each one of them.

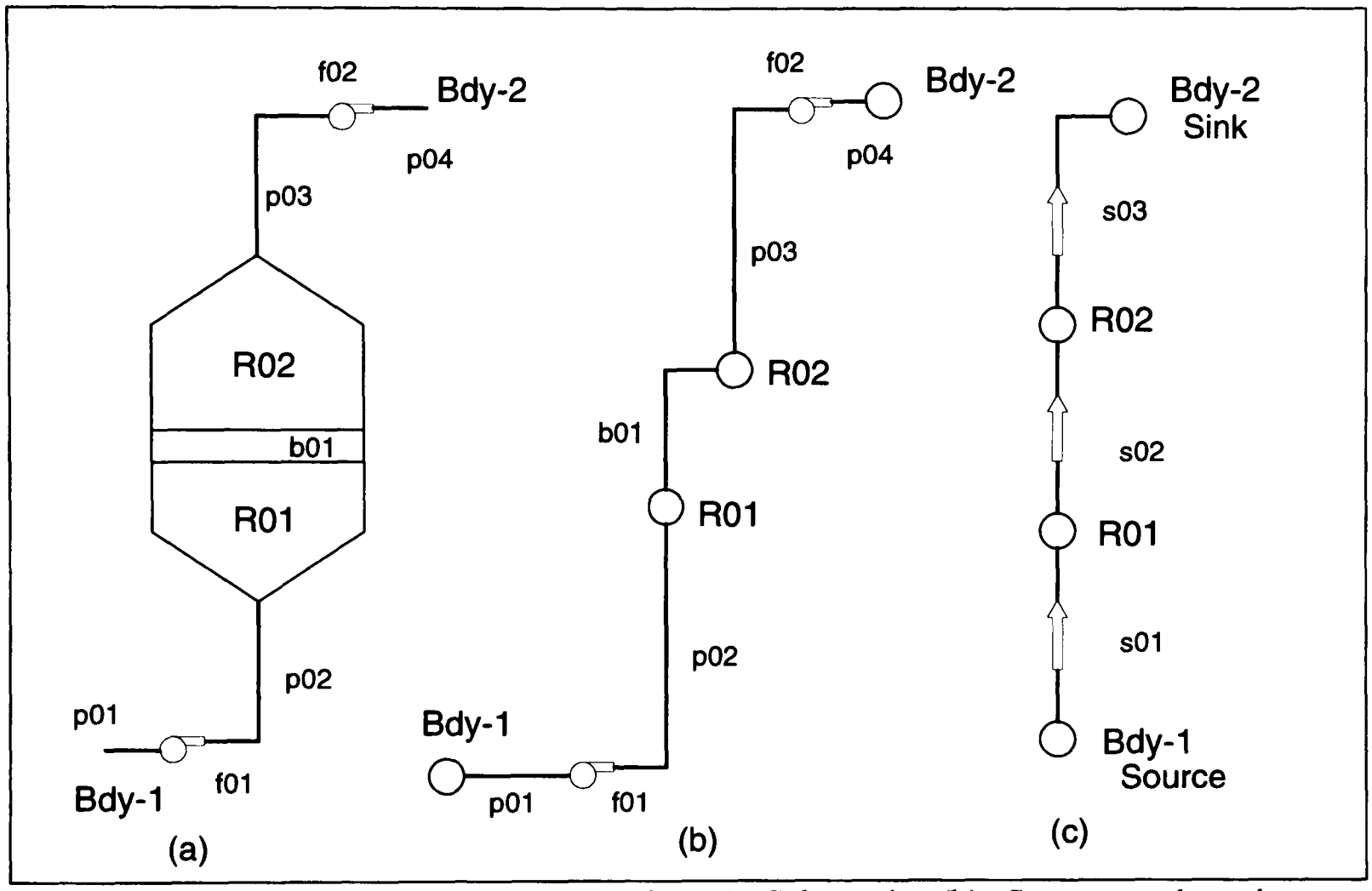

Figure 3.1 Single zone open network; (a) Schematic (b) Stream and node type components and (c) Directed graph of nodes and streams

To illustrate above procedure for computation, in Figure 3.1, the simplest possible network, comprised of single zone is presented. Fan *f01* sucks in air from the atmosphere (i.e. *Bdy-1* or source node), it passes through pipes *p01* and *p02*, enters to region *R01*, which is treated as node, because mixing of different streams can take place here if there are leaks from adjacent zones or atmosphere. Then it passes through packed bed *b01*, region *R02*, pipe *p03* and fan *f02*, which pumps it out of the system to atmosphere, treated as *Bdy-2* here. The

schematic is shown in Figure 3.1(a), whereas in Figure 3.1(b) all the stream type components are presented by lines and node type components are presented by circles. The components linked serially, which have the same flow and lying between the nodes (regions and junctions), are further combined into streams and represented by lines, *s01, s02* and *s03.* Also the direction of flow is allocated to every stream. The result is a *directed graph* of streams and nodes, and is shown in Figure 3.1(c).

As mentioned earlier, the pressure and temperature values are given at the up-end or source nodes whereas flow values are provided at down-end or sink nodes. In this simple network Figure 3.1(c), the flow in *s03* is given, so by Step 1, execution of KCL at nodes *R02* and *R01* can determine the flows in streams *s02* and *s01* respectively. As the pressure at source node, i.e. at *Bdy-1* is given and knowing flow in *s01*, we can use Step 2 to determine node pressure at its downstream end. Similarly pressures at other nodes can be found.

The evaluation of a stream involves sequential execution of mathematical models of its basic components, in order of their connectivity. For example the evaluation of *s01,* will in fact be composed of computation of pipe model for *p01*, fan model for *f01*, and pipe model for *p02.*

In practice open networks are not as simple as this, but they can always be transformed into a tree structure, where leaf nodes correspond to sinks and the root node corresponds to the source node. The open networks do not involve any loops, so only one iteration would be required to compute flow, pressure and temperature distributions.

Closed Networks: Networks having loops (or circuits or meshes as called by different authors) are called closed networks. The definition of loops vary from author to author and also it is application dependent. For example, Sargent 1978, defines the loop in directed graph only if the direction of constituting arcs forms a loop, whereas Osiadacz 1987 considers the original graph rather than the directed graph, because for distribution networks the direction

of flow may reverse in some streams during computation. We adopted the latter convention, as the feed back loops of former type are less likely in pellet induration systems. In our case, we define a loop as a combination of any two alternate paths (where a *path* is a combination of consecutive streams) connecting two distinct nodes i.e. having common start and terminal nodes. In other words, to determine loops, we consider the equivalent graph (the one without any directional signs) of the directed graph.

In general a connected network with multiple, say N_{SRC}, sources will in fact, have N_{SRC}-*1* pseudo loops, as dealt with by Boulos and Wood 1990 and Bhave 1990, so it will be treated as a closed network. Our source node notion is the same as the reference node adopted by Boulos and Wood 1991 and other researchers in hydraulic networks.

The solution of networks involving loops is problematic and requires an iterative approach. Since virtually all practical networks have loops, any general solver should be able to cope with them. To see this increased complexity we consider the simplest possible network with one loop.

In pellet induration systems, the provision for alternate paths to process gas supports the process in many ways. It enables operators to by-pass a zone in case of emergency, as well as to control the flow and pressure distribution in different parts of the network for optimal running of the system. We can introduce a loop in the previously described open network, by connecting a pipe by-passing the zone. This will introduce two junctions *J01* and *J02* and three more pipes to the system. The schematic for this new setup is shown in Figure 3.2(a).

This new network can be similarly transformed to a directed graph of streams and nodes. Now as the number of nodes has increased so the number of streams would also increase proportionately. However, at nodes *R01* and *R02* there are no leaks or other flows, so *p03, R01, b01, R02* and *p05* are embedded in a single stream *s02*, whereas *s03* consists of a single pipe *p04*. Now *J01, s02, J02* and *s03* form a single loop. As a convention we take

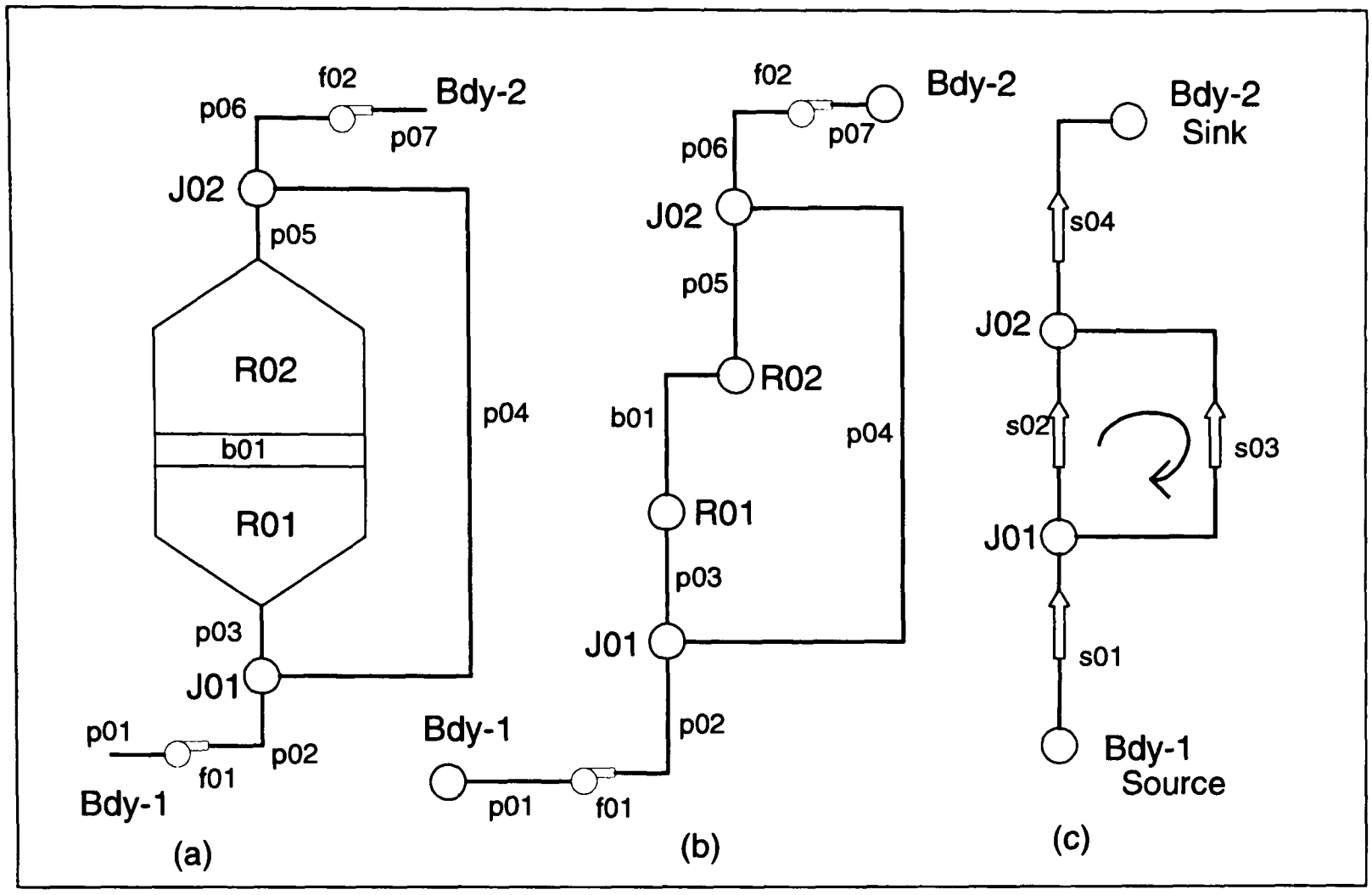

Figure 3.2 Simple closed network with one loop; (a) Schematic (b) Components in stream and node form (c) Reduced directed graph of nodes and stream with a loop

the clockwise direction as positive direction for the loop.

For solution of this closed network we have the same boundary conditions, i.e. flow at sink and pressure at source, are known. Likewise the geometrical data of pipes, pellet and bed data, fan characteristics etc are also specified.

For a network solution the flow and pressure distributions must satisfy the equation 3.1, 3.2 and 3.3 or namely

i. The Kirchhoff's current (KCL) law at all the internal nodes (i.e. the nodes excluding the boundary nodes) of the network;
ii. The Kirchhoff's voltage (KVL) law for all the loops of the network;
iii. The pressure drop in any stream should be governed by its respective constituent components' models.

In the literature a wide variety of methods of solution have been described (also see section 1.1); Hardy Cross 1936, Batey et al 1961, Jeppson 1976, Boyne 1970, Fincham and Goldwater 1979, Goldwater and Fincham 1981, Wood and Rayes 1981, Daugherty et al 1985, Osiadacz 1987, Nielsen 1989, Bhave 1990, Boulos and Altman 1991 and Turner et al 1991 represent some of the main approaches. Goldwater and Fincham 1981 has compared about 20 of the then existing computational models for the simulation of natural gas networks, whereas Fincham and Goodwin 1988, have concentrated their attention on the comparison of underlying methods used in these codes for simulation of natural gas transmission and distribution networks. Nearly all of these methods used a process-centred approach, and most of them solve the network using matrix notation, whereas we are interested in solving the network according to device centred or unit base approach, for the reasons mentioned in sections 1.2,1.3 and 2.3.

To solve the closed network shown in Figure 3.2(c), the flow in *s04*, say F_{s04}, and pressure at *Bdy-1*, say $P_{Bdy\text{-}1}$, are known, and we have to evaluate flows in streams *s01, s02* and *s03* and pressures at nodes *J01* and *J02* such that they satisfy the KCL and KVL constraints. The second of the laws can be stated as the pressure at nodes should be unique and independent of the path followed for computation; that is, whether pressure at *J02* is computed via *s02* or *s03* it should have same value.

The solution of this network can be obtained by carrying out the following steps:

i. Assume some non-zero, positive flow in stream *s03,* say F_{s03}, applying KCL at node *J02* we compute

$$F_{s02} = F_{s04} - F_{s03}$$

and similarly for node *J01* we have

$$F_{s01} = F_{s02} + F_{s03}$$

ii. As $P_{Bdy\text{-}1}$ is known, using the recently computed flow F_{s01}, we can compute *s01*. This will give us pressure P_{J01} at node *J01*.

iii. Repeating (ii) above for streams *s02* and *s03* we can determine the pressure at node *J02*, say it has values P'_{J02} and P''_{J02} respectively. If these two pressures are equal or within a specified tolerance then we are done and can compute stream *s04* to complete flow and pressure distributions of the network. But if these are NOT equal then

either

$P'_{J02} > P''_{J02} \Rightarrow \Delta P_{s02} < \Delta P_{s03}$ that is contribution of anti-clockwise stream, *s03*, is greater than the clockwise stream, so the assumed flow F_{s03} should be decreased by an amount ΔF to adjust this inequality, and by a similar amount the F_{s02} should be increased to satisfy KCL at nodes

or

$P'_{J02} < P''_{J02} \Rightarrow \Delta P_{s02} > \Delta P_{s03}$, which is reverse of the previous case, implying that, the clockwise stream *s02* is dominant, so to adjust the inequality the flow in this stream F_{s02} should be decreased.

iv. The determination of correction factor ΔF is crucial for ultimate convergence, it can be done by a trial and error method for a simple network like the above example but it would be not practical for any real multi-loop network. So we link ΔF to the overall pressure drop across the loop to make it self-correcting. In this case when $P'_{J02} > P''_{J02}$, the overall pressure drop $\Sigma(\Delta P)$ in loop, obtained by summing up pressure drops in the direction shown for loop, would be negative. Suggesting that the anti-clock wise contribution is dominant, so the flow in *s03* should be decreased.

The components comprising the streams *s02* and *s03* are pipes and packed bed, in the latter case, the pressure drop ΔP has quadratic dependence on flow, which may be approximated as

$$\Delta P = f(F^2) = KF^2$$

where K is stream or component dependent constant. Define ΔF as a corrective fraction of loop flow ($<< F$), such that when it is added to the previous flow value the new flow satisfies KVL i.e.

$$\sum_{i=1}^{N} \Delta P_i = \sum_{i=1}^{N} \{K_i (F_i + \Delta F)^2\} = 0$$

where N is total number of streams comprising the loop, i.e. 2 in this case. Now expanding the squared term and ignoring second order terms in ΔF (assumed very small when compared to F) and re-writing we get

$$\Delta F = -\frac{\sum_{i=1}^{N} K_i F_i^2}{2\sum_{i=1}^{N} K_i F_i} = -\frac{\sum_{i=1}^{N} \Delta P_i}{2\sum_{i=1}^{N} K_i F_i}$$

The flow direction is taken into account while evaluating overall pressure drop in the loop. If the flow direction in the stream is along the loop direction then it will have positive sign, and a negative sign for stream flow direction opposing loop direction. The negative sign on the right hand side of above equation takes care of the increase or decrease of flow in respective streams. The flow direction can be introduced into the above equation with the help of a *loop-stream incidence matrix,* (section 2.2.3) and the corrective flow for any *jth* loop of any general network can be written as

$$\Delta F_j = -\frac{\sum_{i=1}^{N} b_{ji} \Delta P_i}{2\sum_{i=1}^{N} |KF_i|}$$

where b_{ji} is element of loop-stream incidence matrix, with value +1 if *ith* stream has same direction as *jth* loop; if these directions are opposite then it is -1.

v. Knowing ΔF_j the flows F_i corresponding to streams of the *jth* loop can be updated as

$$F_i^{new} = F_i^{old} + b_{ji} \Delta F_j$$

Since we have single loop, say loop 1, comprised of 2 streams *s02* and *s03*; and by the shown flow directions, $b_{1,s02}$ is +1 and $b_{1,s03}$ is -1, then the flows will be

$$F_{s02}^{new} = F_{s02}^{old} + \Delta F$$

$$F_{s03}^{new} = F_{s03}^{old} - \Delta F$$

The steps (i) to (v) will be repeated with this improved flow distribution until the pressure distribution satisfying KVL is achieved for all loops.

Consider the convergence behaviour of the stream flows and node pressures by above procedure. Suppose $\Sigma\ (\Delta P)$ is negative, i.e. counter-clock wise effect is dominant; then computed ΔF by step iv would be positive, and so for next approximation of flow, step v above, F_{s02} will increase and F_{s03} will decrease by amount ΔF. If $\Sigma\ (\Delta P)$ is positive, then the effect will be reversed on *s02* and *s03* flows. This shows that the algorithm is self-correcting and will lead to a converged solution.

For practical networks, having multiple loops, this incremental flow will be computed for each of the loops, and the corresponding streams' flows updated. Streams participating in more than one loops will have an effect from each one of these loops. This flow updating would also effect the flows of the other streams in the network.

In the pellet induration system networks, the streams have components like fans, where this quadratic relationship of flow with pressure drop is not valid, so the method described above cannot be generalised. Also to cater the future needs of the model, it is intended to provide the facility to add more components to network, whose nature of pressure-flow relationship is unknown at this stage. So we look forward for some method which could correct flow distribution in the network more efficiently for any generic stream components. Instead of depending on pressure drop - flow relationship, it should depend upon the values of state variables at the ends of the network components.

3.2.3 Existing Solution Methods and Pellet Induration System Network

In sections 1.1.2 and 1.4.1 the existing methods for the solution of flow networks are discussed. All of these methods assume an explicit, well defined and uniform relation between pressure drop ΔP_i and flow F_i for all streams of the network. This relation is dependent on the nature of fluid (e.g. air, water or natural gas), geometry of the stream component, operating conditions (e.g. low, medium or high pressure networks for natural gas distribution and transmission); and manipulated somehow in the development of solution method. One such relation for hydraulic networks is exponential or power formula i.e. $h_f = K Q^n$ where h_f is head loss or pressure drop ΔP, Q is flow same as F in our notation, and K and n are constants having specific values for Hazen-Williams, Manning or Darcy-Weisbach equations (Jeppson 1976). The linear theory method transforms this nonlinear relation into a linear equation such as $h_f = [\, K\, Q_{prev}^{\,n-1} \,]\, Q$ where Q_{prev} is previous iteration value of flow and solves it iteratively (Wood and Charles 1972). Similarly this relation is used for evaluating the loop flow correction ΔF in the Hardy Cross method similar to the one shown in section 3.2.2, and for Newton-Raphson method to evaluate the Jacobian matrix.

According to the device centred approach adopted and the formulation described in section 3.2.1, the network components are modelled independently and their behaviour determines the behaviour of the respective streams. Though the component models are well-defined and explicit, the relation between the pressure drop ΔP_i and flow F_i varies from stream to stream and depends on the components forming the stream. For example the stream *s01* (Figure 3.2) consists of components *p01, f01* and *p02* so the equation 3.3 corresponding to *s01* will contain the mathematical models of all these three components whereas stream *s03* consists of only one component i.e *p01* and hence equation 3.3 for this stream will contain the mathematical model of *p01* only.

We aim to: (1) solve the component models and hence the streams as exactly as possible by computing the mathematical models specified in section 2.4, (2) facilitate the user to configure the network and hence define the streams' composition of his choice, and (3) add further network components to the system at later stage whose models are not known at this stage. For the accomplishment of these aims, the explicit evaluation and direct substitution of equation 3.3 into other equations is avoided and instead a hierarchical approach (mentioned in section 1.4.2) is adopted. According to this approach, at the higher level the network is solved for Kirchhoff laws and at the lower level the streams are computed by solving the mathematical models for their constituent components. The interaction between the two levels is only by passing the values of the desired state variables i.e. node pressures and stream flows. This algorithm is explained in the next section, and the reasons why other existing methods cannot be used at least at the higher level will be discussed at the end of next section.

3.3 Primary Algorithm for Air Flow Distribution Computation

Pellet induration systems have by-pass paths for process gas and also the interconnections between the different zones are such that practically loops are inevitable. Secondly, the components are heterogenous by nature, so the assumption i.e. the quadratic

relation used for the derivation of ΔF (described in section 3.2.2) is not valid for general streams comprised of the feasible range of components. The example of single zone although simple provides a clue, that if any real network is reduced to an open network then it can be easily solved for flow and pressure distribution. Further, since it was a closed network using the pressure distribution at nodes the flows in the torn (i.e. independent or chord) streams can be found. Residuals at nodes could then be computed using KCL. These residuals are used systematically for the correction of flow distribution in the network. This procedure of flow, pressure distribution computation and its correction is carried out iteratively until the node residuals become less than the specified tolerance. This converged flow distribution of the network, gives the pressure distribution which satisfies the Kirchhoff's voltage law, as it gives the unique node pressure to every node, independent of the path followed for its computation. The flow distributions can be further used to evaluate temperatures in all the streams of the network.

The algorithm is described in Figure 3.3. It is simple and very efficient. To explain its working, we consider a typical pellet induration system shown in Figure 2.3. Applying graph theoretic notation, the system can be presented as connected graph of E *edges* and N *nodes,* which is shown in Figure 2.5. This connected graph, (E,N), can be partitioned into *tree* and *co-tree* sub-graphs (E_{tree}, N_{tree}) and $(E_{co\text{-}tree}, N_{co\text{-}tree})$ respectively. The edge and node sets are such that $E_{tree} \cup E_{co\text{-}tree} = E$ and $E_{tree} \cap E_{co\text{-}tree} = \phi$ that is an edge can only belong to either a tree or its co-tree; whereas the node set $N_{tree} \cup N_{co\text{-}tree} = N$ and $N_{tree} \cap N_{co\text{-}tree} \neq \phi$ means that a node can belong to either or both of these partitions. With out going into details how these partitions were obtained, which will be covered later in section 3.5, the tree and co-tree partitions of the original graph of the system are shown in Figure 3.4.

Practically, for a network with multiple sources, say n_{src}, the tree structure would in fact be a *Forest,* having n_{src} trees as its components, one for each of the sources. Whereas the *co-tree* would be comprised of all those edges of the network which are not included in any of the *trees.* In the following explanation, the directions of computation from *'sink to source'* or vice versa, would refer to the sinks and source of the respective tree of the forest. Also the

Algorithm for a General Network Computation:

Begin

1.0 **Read-in**
- a) Network Configuration, Components' data and their connectivity
- b) Boundary Conditions: Pressure & Temperature for Source and Atmospheric nodes, and Flows for Sink nodes
- c) Computational controls; *Tolrnc, MaxItr* etc

* 2.0 **Generate** Tree and Co-tree Structures for the network

3.0 **Initialize** Co-tree edge flows $F^0_{Co\text{-}tree}$ and internal node loads $\delta q_I = 0$

4.0 **Compute** Flow in all Tree edges, F^0_{Tree}, satisfying Kirchhoff's node law (KCL), {Sink → Source direction}

ITERATE K = 1

5.0 **Compute** Pressure at all Tree nodes, P^K_{Tree}, using edge flows F^{K-1}_{Tree} {Source → Sink direction}

6.0 **Compute** Flow in all Co-tree edges, $F^K_{Co\text{-}tree}$, using recent node pressures P^K_{Tree}

7.0 **Compute** Residuals for all internal nodes; $f^K_I = \Sigma a_{IJ} F^K_J$ where a_{IJ} is an element of node-stream incidence matrix

If ($|f^K_I| <$ *Tolrnc* **.or. K >** ***MaxItr***) for all internal nodes, **go to 12.0**

†8.0 **Compute** overall error in the network, δQ^K, for Kth iteration

†9.0 **Distribute** the error, δQ^K, proportionately to each of the internal nodes δq_I

10.0 **Compute** Corrective flow δF^K_{Tree} in Tree edges, using δq_I as node loads {in Sink → Source direction}

11.0 **Compute** Flows in Tree edges; $F^{K+1}_{Tree} = F^K_{Tree} + \delta F^K_{Tree}$

Increment K = K + 1; **go to 5.0**

12.0 **Compute** Temperatures for all the edges of network, T, using converged Flows F^K

13.0 **Output** Flow, Pressure and Temperature distributions for the network

Stop;

---

* This algorithm is described separately.
† The respective equations are given in the text.

Figure 3.3 The algorithm for computation of pellet induration networks

mention of tree computation would imply to the computation of the forest, treating each of the trees one by one.

Boyne 1970 described a similar method, firstly he named it as *new method* but later referred it as *Secant method,* as the flow correction was based on the well known secant method. He applied this method to a number of real networks used for natural gas distribution and transmission and claimed that it gave converged results for these tested networks. So we use his proposed flow correction equations. He used Travers algorithm (Travers 1967) for extraction of branch system (or tree structure in our terms) from closed network, which is based on the resistance offered by the respective pipes and the pipes having high resistance are declared as independent (or torn) pipes. In other words, it tries to reduce the distribution network into an open (or tree) network, with minimum flow resistance. For a multi-source network, Boyne declared one source as primary and others as secondary sources and selected the pipes adjacent to secondary sources as independent pipes. He was able to accelerate the convergence of the nodal and loop methods, using extrapolative iterative methods like Newton-Raphson method, as he solved the network as a linear system of equations of the form $\boldsymbol{Ax=B}$, but was not able to accelerate convergence of secant method. Boyne used a process centred approach and stated that the secant method, would have a storage requirement of $\boldsymbol{8.E+6.N.DEG_{max}}$ for a network of $\boldsymbol{E}$ edges and $\boldsymbol{N}$ nodes and $\boldsymbol{DEG_{max}}$ as maximum degree of a node in the network.

The computational procedure according the present algorithm, with the boundary conditions of known flows at sinks nodes, and known pressures at atmospheric and source nodes; is described in the following paragraphs.

In the 'initialization' i.e. step 3.0 of the algorithm (Figure 3.3), we assign an approximate flow to each of edge of the co-tree. Experience has shown that, if these flows are unique to each of the edges, the algorithm converges faster. These flows are treated as loads at the respective nodes whilst computing the flow distribution in a tree. The sink node flows are known as a boundary condition, so starting from the node upstream to the sinks,

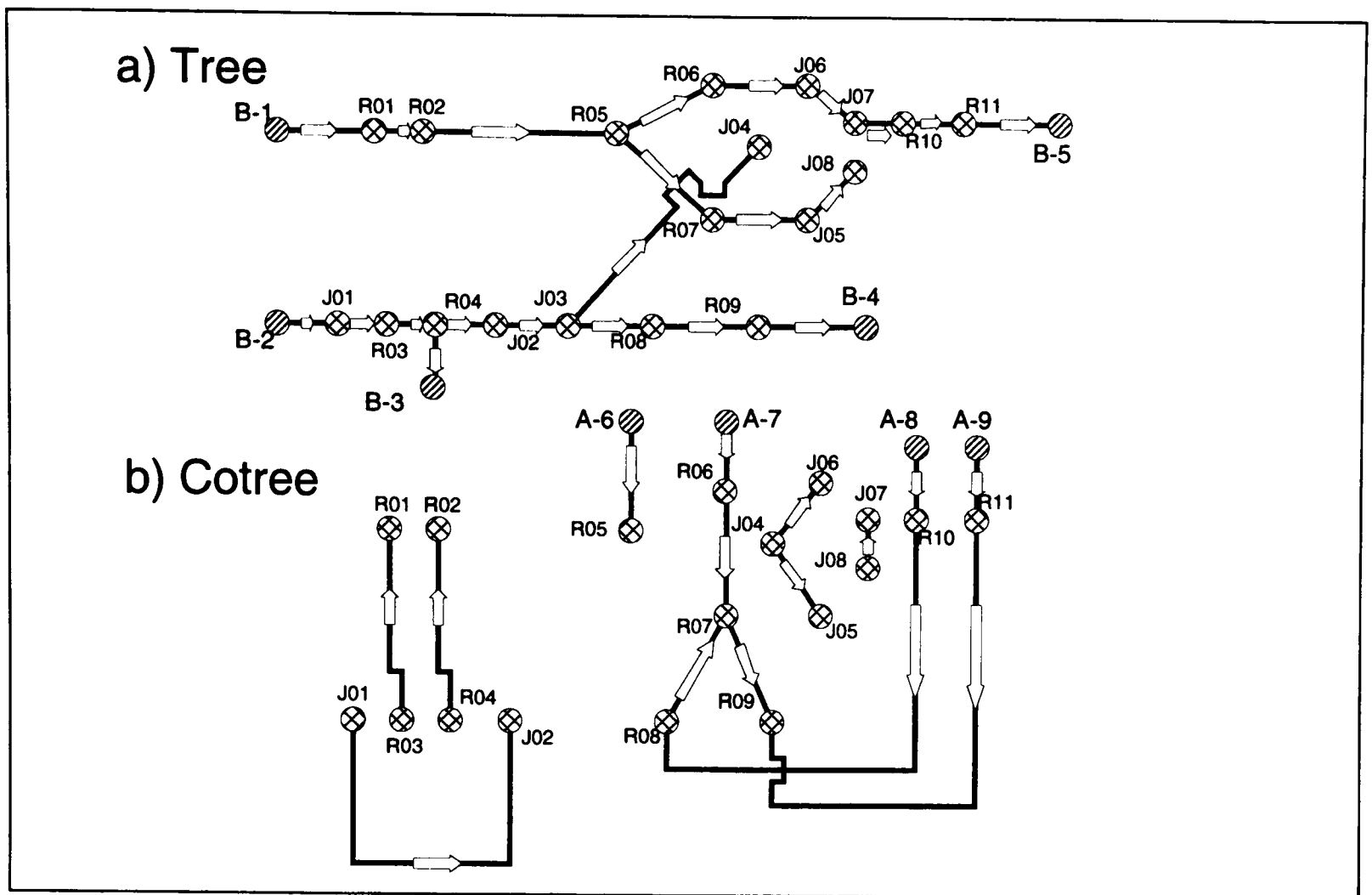

Figure 3.4 The Tree and Co-tree partitions of the described network

we compute each of the tree nodes for Kirchhoff's current law, KCL, and compute the flow in the tree edge entering the node; by the very definition of a tree, there is always one such edge. We continue the computation, until the source node or the root of the tree is reached. This procedure is repeated for each of the trees in the tree structure. Since the source node pressure is known, as a boundary condition, and flow in the tree edges have been evaluated; so the pressure at all tree nodes can be evaluated, by executing the respective tree edges in turn from source to sink direction. Thus at the end of step 5.0 of the primary algorithm (see Figure 3.3), we have computed pressure at every node of the tree. Indirectly the node pressure for each of the co-tree nodes is known (see Figure 3.4), so each edge of the co-tree could be executed, to find its flow, as both of its end pressures are known.

In step 7.0, we use the recent flows for all the edges of the network, (irrespective of their association with a tree or co-tree) computed in previous steps, and evaluate the residual f_i, for each of the internal and atmospheric nodes (i.e. nodes other than boundaries of the system i.e. source, sink). According to KCL these residuals should be zero for the internal

nodes, but if this is not, then it predicts the deviation from the desired flows and can be treated as the error. The flows in a tree are improved systematically so as to reduce this error to zero or to a value smaller than a specified tolerance, say *Tolrnc*. For this improvement we use the equations proposed by Boyne 1970. The derivation of these correction terms is given in Appendix-B.

After evaluating the residuals, at the Kth iteration, f_i^K for all internal and atmospheric nodes of the network, we compute overall error in the network using the following equations

$$\Delta Q^1 = \frac{\sum_{i=1}^{N} |f_i^1|}{2} \qquad for \quad K = 1$$

$$\Delta Q^K = \frac{\sum_{i=1}^{N} |f_i^K| \times \Delta Q^{K-1}}{\sum_{i=1}^{N} |f_i^K - f_i^{K-1}|} \qquad for \quad K > 1$$

where N is number of total internal and atmospheric nodes. This overall error is distributed on the respective nodes using

$$\delta q_i = - \frac{f_i^K \times \Delta Q^K}{\sum_{i=1}^{N} |f_i^K|}$$

where f_i^K is the residual or net flow at ith node at Kth iteration, so it could be either positive or negative. The negative sign in the equation takes care of increase or decrease for the corrective flow in the tree. In step 10 of the algorithm, the corrective flow is computed in the tree by assigning δq_i^K as the load to the ith node of the tree with the usual convention, that is, if it is positive the flow is into the node and if negative the flow is out of the node, and ignoring all other boundary (sink) flows. Using KCL all nodes are executed in the sink to source direction and the incremental flow in each of the tree edges, δF_{tree}^K, is computed.

This is added algebraically to the existing flow of the respective edge, and the corrected flow in tree is evaluated, as given by step 11.0 of the algorithm. The convergence of the method for a real-life system is shown in Appendix-C.

The steps 5.0 to 11.0 are repeated, until a converged flow distribution is achieved. These flows are used to compute the temperature distribution in the network. The computation of temperature distribution uses the same algorithm, as for pressure computation, and treats the temperatures of process gas at sources and in packed beds as parameters for simulation. Like pressure distribution, the temperatures of process gas at nodes, are found using the tree branches. These node temperatures are input to co-tree branches to find the temperature at their down stream ends, which are used to evaluate the net heat entering or leaving the node. The iterations are continued until the net heat energy entering an internal node is zero or less than a pre-set small value. These computed distributions will satisfy the condition, that all edges meeting at a node should have same temperature and pressure values as that for the node.

Using this method, at the higher level only the KCL equations which are linear in flows are solved simultaneously whereas KVL equations (i.e. equation 3.2) which, after the substitution of equation 3.2, become nonlinear in flow are not solved directly. Instead, this method systematically updates the flows in tree branches so that the KVL equations are satisfied. At the lower level streams are computed through either tree or cotree structures i.e. steps 5.0 and 6.0 of the algorithm (Figure 3.3) where nonlinear models of the respective components are solved and the computed state variable (node pressures or stream flows) values are fed back to the higher level. The recent flow values are used to generate the correction term and to update the tree flows for next iteration. The nonlinearity due to loops is resolved by this higher level iteration. The direct substitution of equation 3.3 into equation 3.2 is avoided to conform with the device centred approach and thus for the sake of heterogeneity and genericity of network components and generality of streams. However, this algorithm restricts the use of the other standard methods such as linear theory or Newton Raphson methods at the higher level, as they require the explicit uniform relationship for

stream pressure drop ΔP and flow F. Although, this is a shortcoming of the proposed method but it is outweighed by the benefits provided by it which are described in sections 1.4, 3.9, 4.5 and 6.1.

From the described primary algorithm the significance of tree and co-tree structures is obvious. In the following sections we will first briefly review the state of existing algorithms, generally for loop detection, tree and spanning tree generation; then, why these were not useable in our case; and lastly the development of secondary algorithms for partitioning of network into tree and co-tree structures will be shown.

3.4 Existing Algorithms for Loop Detection and Tree Generation

In the last four decades the graph theory has widely been used in many diverse fields, the algorithms for loop detection and tree generation are the back-bone to the solution approaches. These have been developed for a variety of domains of applications. These algorithms have many common features and a noticeable duplication of implied concepts. It is difficult to make a detailed comparison covering all or most of the related algorithms due to their vast application domains and extensive research therein. Instead some of the main algorithms from few of the fields, which contributed to the development of the ones described in next section, are briefly discussed here.

3.4.1 General Algorithms :

Loop detection and spanning tree generation algorithms are covered by nearly all books on Graph Theory, Combinatorial Mathematics, Data Structures and Network Algorithms. For example Deo 1974, Knuth 1973a & 1973b, Tarjan 1983, Syslo et al 1983 and Ahuja et al 1993 have addressed these topics to sufficient details.

Deo 1974, has given a detailed bibliography on the subject. A network can have many trees and spanning trees. So to impose some restriction in the selection of certain spanning trees, usually a weight is assigned to each edge of the tree. Then a spanning tree having minimum weight is defined as (the shortest or) minimal spanning tree. He stated the Kruskal, Prime and Dijkstra algorithms for the generation of minimal spanning tree and concluded that Kruskal is not as efficient as the other two, because it required the pre-ordering of edges in non-decreasing order of their weights, which is computationally expensive for large networks. Whereas, the Prime and Dijkstra algorithms select the shortest (one with minimum weight) stream among the incident edges of the node. This is done by tabulating the weights in a node-node adjacency matrix and manipulating that matrix. Deo 1974 stated his own five step algorithm for spanning tree generation, which is quite similar to the one given by Traver explained below.

3.4.2 Algorithms related to Directed Graphs :

The directed graphs, where all edges have a fixed direction, arise from many fields, e.g. fluid flow networks (which will be described in next subsection), solution of large sets of algebraic equations, chemical engineering flow sheeting problems and activity scheduling and critical path analysis problems.

Solution of a large set of equations, specially algebraic equations, since the partial differential equations PDEs and ordinary differential equations ODEs also reduce to algebraic equations after discretization, can be obtained by partitioning it into smaller sub-sets. These (sub-)sets can be computed individually, requiring lesser storage because of reduced problem size. Exploitation of consistent solution procedures and which variable to be computed from which equation, depending on the nature of equations can lead to an efficient computational strategy. These sets can be thought as a modules or programming procedures and the variables required and being computed by a module as its parameters. If the number of variables are equal to the number of equations then the module can be computed independently, but if the number of variables, say v_i, are more than the number of equations,

say n_i, then values of the extra, v_i - n_i, variables would be required for the computation of respective module and their availability will be a pre-requisite for the module. The solution procedure can be represented by a directed graph, where the nodes correspond to the modules i.e. sets of equations and the edges showing the variables. The edges directed towards the nodes are the required input variables whose values are meant to be known, either as an initial guess or from previous iteration, and the edges leaving the nodes correspond to the variables computed at the node i.e. output of the node.

The criteria for partitioning of the original set to smaller subsets is crucial for efficiency of the solution and is discussed by Sargent 1978. If the resulting graph is acyclic that is it does not contain any cycles or directed loops then these modules can be computed sequentially and do not require any iteration, but in practice, the network does normally contain cycles. These directed cycles are similar to the cycles occurring in activity scheduling problems, where one node is having input from another which itself is dependent on the output of the considered node. Indeed these cycles point to the strong coupling of the equations involved in the concerned modules. To find out some optimal solution strategy, which gives the precedence order in which the modules or nodes will be computed, first these directed cycles would be torn-off to get an acyclic graph. The set formed by these torn edges is called *tear set*, and the process as *tearing* (Sargent 1978, Motard and Westerberg 1981 and Montagna and Iribarren 1988a) or *decyclization* (Deo 1974) of directed graphs . Then the network can be solved iteratively, by first assigning some initial values to the variables associated with these torn edges, and on computation of respective nodes these variables could be replaced by their computed values in subsequent iterations.

This approach is well used for the solution of chemical engineering flowsheeting problems. Each physical component or unit of the plant, has a mathematical model which has a corresponding set of equations and can be represented by a node. It is usually referred as *sequential modular approach* in the literature as discussed in section 1.3.

The overall computational efficiency of the system is dependent on the choice of the tear set. Similar to minimal spanning tree algorithms a weight, which in fact is a measure of some physical quantity related to computational efficiency say e.g. the difficulty involved in computing the corresponding variable from the respective equation, is assigned to each edge. Lee et al 1966 stated that the tear sets with minimum weight give a precedence order which results in the fastest computation. The tearing algorithms to determine the optimal tear sets, have been discussed by Lee et al 1966, Pho and Lapidus 1973, Sargent 1978, Motard and Westerberg 1981 and Montagna and Iribarren 1988a, 1988b.

The precedence order for computation remains fixed for multiple runs of the process simulation package for a particular plant so the tearing can be done once and stored and read by the later runs. This pre-processing facility can result in significant savings in computational time (Montagna and Iribarren 1988a).

3.4.3 Algorithms related to Fluid Flow Networks :

In reference to fluid flow networks the loop detection and tree generation algorithms have been described by many authors. The storage requirements and computational efficiency of all the methods using loops information for their computation, is directly related to the number and nature of edges (pipes) participating to these loops. The increase in the number of edges will increase the number of non-zero elements in the loop incidence matrix, which will increase the storage requirements. Also there will be more non-zero elements in the related Jacobian matrix for Newton type methods whose generation and solution will require more time. Hence considerable emphasis has been given to the study of the nature of generated fundamental loops and their selection among these generated ones. *Fundamental loops* (or *basic* or *independent loops*) are the loops which does not include any other loop; and these can be generated from the spanning tree and co-tree information.

Voyles and Wilke 1962 tested different loop configurations for the solution of two simple hydraulic networks using the Hardy Cross method and found that the resistance of

pipes common to loops is related to the convergence of network. The loop configurations with common pipes having lesser resistance require fewer iterations to converge.

Daniel 1966 indicated that the basic loops with minimum overlap will provide higher efficiency for Hardy Cross method, as it would decrease the number of terms required for the flow correction for each loop. He generated basic loops from the spanning tree, and indicated that the optimal basic loop for a rectangular grid type network would be the individual circuits but there does not exist any such tree which can lead to this set of basic loops.

Travers 1967 generalised the notion proposed by Voyles and Wilke, and developed an algorithm which generates basic loops for natural gas and water networks, having common pipes with minimal resistance. According to this algorithm all pipes are first sorted out in ascending order of their resistances and put to a list. A pipe is picked up from the list and included into tree if:

- one of its ends is already in tree and the other is not, or
- the two ends belong to different trees then the two trees are merged together to form one tree including this pipe, or
- both ends are not in any of the defined trees then a new tree can be defined.

Otherwise, if both ends of the selected pipe are already in tree then inclusion of this pipe will introduce a loop, so it is declared as *chord* (or *independent pipe*) and placed in co-tree. The executed pipe is deleted from the list and procedure is repeated until all pipes have exhausted. The so generated tree will have low resistance pipes and the non-tree or co-tree pipes will have higher resistance. It would be a spanning tree if the original graph is a connected graph otherwise it would be *spanning forest* i.e. collection of trees and so some authors (e.g. Fincham and Goodwin 1988) have referred it as *Forest Method*. The non-tree i.e. cotree branches, along with the tree branches can define the basic loops. This algorithm is most commonly used for fluid flow networks e.g. Boyne 1970, Osiadacz 1987 and Fincham and Goodwin 1988.

Welch 1966, has given the algorithms for loop generation based on incidence matrix operations and claimed that this is more efficient method for large networks then the methods based on common search.

Yevdokimov 1969, independently developed this approach (the generation of basic loops from incidence matrices) further and has given algorithms for generation of cut-sets and basic loops. He has given the parallel set theoretic formulation of these algorithms which can manipulate directly the edge and node numbers for generation of basic loops and cut-sets. The later formulation is easier to program, reduces the storage requirements and user input data and is faster in computation. Whereas matrix formulation has a sound theoretical base and straight forward proof for the validity of the stated algorithms. He mentioned that efficiency of Hardy Cross method is influenced by the basic loops with minimum overlap and simultaneously the lower possible resistance in common pipes. Whereas the efficiency of Co-ordinate Gradient Method (or Gauss Seidel Network method, which is an improved form of Hardy Cross method) is dependent on minimal (resistance) spanning tree. He gave an algorithm for minimal spanning tree generation using set theoretic formulation.

Fincham and Goodwin 1988 have provided the pseudo code for Traver's algorithm in FORTRAN like IF ... THEN ... statements. They pointed out that the standard methods based on *depth first search* or *breadth first search*, like the ones described by Osiadacz 1987, have a time requirement of $o(n^2)$, and are not efficient for large networks; whereas the Travers algorithm requires computing time of $o(n\ log_2 n)$. Unfortunately, the resulting loops have significant overlap, which is wasteful of storage for loop-pipe incidence matrix and slows down the convergence of Hardy Cross method. They have given a loop reduction algorithm which reduces this overlap between loops. This is done when original loops are generated. As resistance is inversely proportional to the pipe diameter, so they use pipe diameters instead of pipe resistance. They have provided pseudo code for this reduction algorithm and described some interesting results about the dependence of convergence on the original spanning tree generation. For instance the Hardy Cross method converged faster with the original loops produced by some standard breadth first search algorithm then these were reduced using this

reduction algorithm. But the loops generated by Traver algorithm and then reduced proved to be very efficient; i.e. small storage requirement and fast convergence. Most of the programs developed by British Gas use this improved algorithm, that is initial loops generated by Traver's algorithm and then reduced, for the generation of basic loops.

Osiadacz 1987 has given five algorithms for loop generation which are the most widely used by gas network models, including Travers and the two versions of improved algorithm used by British Gas. He computed the generated loops by each of these algorithms and concluded the British Gas algorithms give the best loops with minimum overlap thus resulting in the most sparse matrices and minimum resistance in pipes common to the loops.

Boulos 1989, has stated two algorithms for water networks, both based on breadth first search. He categorizes the pipes as active, the ones whose both ends have degree two or more, and non-active which have either one or both of the ends with degree less then two. During search only active pipes are considered. In the first algorithm, all active pipes are put to a list, the first pipe of the list is selected and its up-end node is marked as a terminating node and down-end node as a start node. The breadth first search is started from the start node and next level nodes are visited, and then from these next level nodes the further incident nodes are visited. This fanning out process is continued till the marked terminating node is accessed. Then the circuit formed is enumerated and the selected pipe, being responsible for this loop is declared as non-active and the degree of its end nodes decreased by one, which would trigger the pipes adjacent to the selected pipe and participating only in this loop also to be non-active. So the list of active pipes is updated and search for next loop is restarted. The procedure is repeated till active pipe list is exhausted. The second algorithm is based on a spanning tree, where a pipe from co-tree is selected and its up-end is marked as terminating and down-end as starting nodes, and breadth first search is carried out on tree branches having active status. Loop is traced when terminating node is visited by search mechanism, and enumerated. Like the other algorithm, the selected co-tree branch is the declared as non-active and corresponding tree branches which participated to this loop will also become non-active. This procedure is carried out for all co-tree branches. The second

algorithm can incorporate the minimum resistance criteria but overlap in loops generated by both of these would require some explicit resolution.

Wang 1982, described tree algorithms for the solution of mine ventilation networks. In these networks the air enters from the entrances, circulates in the mine airways so that it should always satisfy the velocity and concentration standards set by Mine Safety and Health Administration. Each airway leads to a wall face i.e. the working area after that air is pumped out through exits. He defined the set of all wall faces as cut-set and two trees. The up-tree with respect to a reference entrance and down-tree with respect to a reference exit. Using these trees he traced a critical path, which is the longest possible path and offers maximum resistance. He optimized the fan locations and characteristics using this critical path.

None of the above reviewed algorithms is generic, instead these all are specific to the respective application domain, method of formulation and the method used for solution. The characteristics like low resistance spanning tree for natural gas distribution network using Hardy Cross method, are not required by a mine ventilation network which are solved using critical path method. These algorithms were designed to achieve different goals so when applied to the same network, they produced different outputs (Osiadacz 1987). Also, in these described networks all edges (or streams in our terms) had same equation relating pressure drop and flow (only minor variation of some coefficients was needed from edge to edge), hence its inclusion in tree or in co-tree had no significant computational constraints.

As mentioned in sections 1.3 and 3.4, the pellet induration system network is being solved using two level hierarchical solution approach. At the top level the network is solved for coordination or system variables to satisfy the two Kirchhoff's laws whereas at the lower level the streams are computed to satisfy their constituent components' mathematical models accepting coordination variables as parameters. Each stream has a variable number of components, is different from the other streams and has its own computational constraints. Like wall faces of mine ventilation networks, the packed beds in the pellet induration system are important units, we want them to be in tree structure and these offer resistance higher than

other components. As the used solution approach (section 3.4) required computation of flow from a cotree branch when end pressures are given, in consequence the computation of a multi-component streams will be computationally very expensive, so only single component streams could be put in cotree. Likewise the leaks could be efficiently computed for flow, when end pressures are given so these can preferably be placed in cotree, this is a contradiction to Hardy Cross method requirements as leaks offer less resistance than pipes. All these specific requirements for pellet induration networks necessitate the development of another set of domain dependent algorithms. These are described in the following section.

3.5 Secondary Algorithms for Partitioning of Network into Tree and Co-tree structures

The transformation of a pellet induration system (Figure 2.3) to a connected graph of nodes and streams (Figure 2.5) using graph theoretic approach was discussed in section 2.2.3. This graph theoretic representation will be used here, for the development and explanation of the algorithms. In principle, the implementation details will be discussed in the relevant sections of chapter 4, but for the sake of completion these have been briefly filled in where required.

Graph theoretic representation (Figure 2.5) also clarifies the node and stream incidence. The basic definitions of graph theory are intuitive and are given in section 2.2.3. Further the '*level*' of a node is its distance from the source node in terms of streams. Referring to Figure 2.5, the node R02 is at level 2, whereas R06 and R07 are at the same level 4. For a multi-source network, like the one shown in Figures 2.3 or 2.5, the level is sub-network dependent, and is referred to the respective source node.

For ease and computational efficiency the nodes and streams are assigned integral numbers as identifiers. This is done by numbering the nodes sequentially, starting from a source node and following the main paths towards the associated sink nodes. Any strategy for

this numbering could be used. The normal depth first search or breadth first search could be good systematic approaches for the sake of consistency. The algorithms described below for partitioning are independent of numbering strategy. These are mainly dependent on the node-stream connectivity and stream weights and are aimed to result in same tree and co-tree structures for a particular network. Preferably the atmospheric nodes should be stacked at the end of the sequence. This has an advantage of generating smaller arrays for reduced graphs, which will be discussed later.

The streams are numbered in the same order as nodes, keeping the streams comprising of leaks at the end, to generate smaller (computational) loop indices and to accelerate the execution for NO_LEAK option.

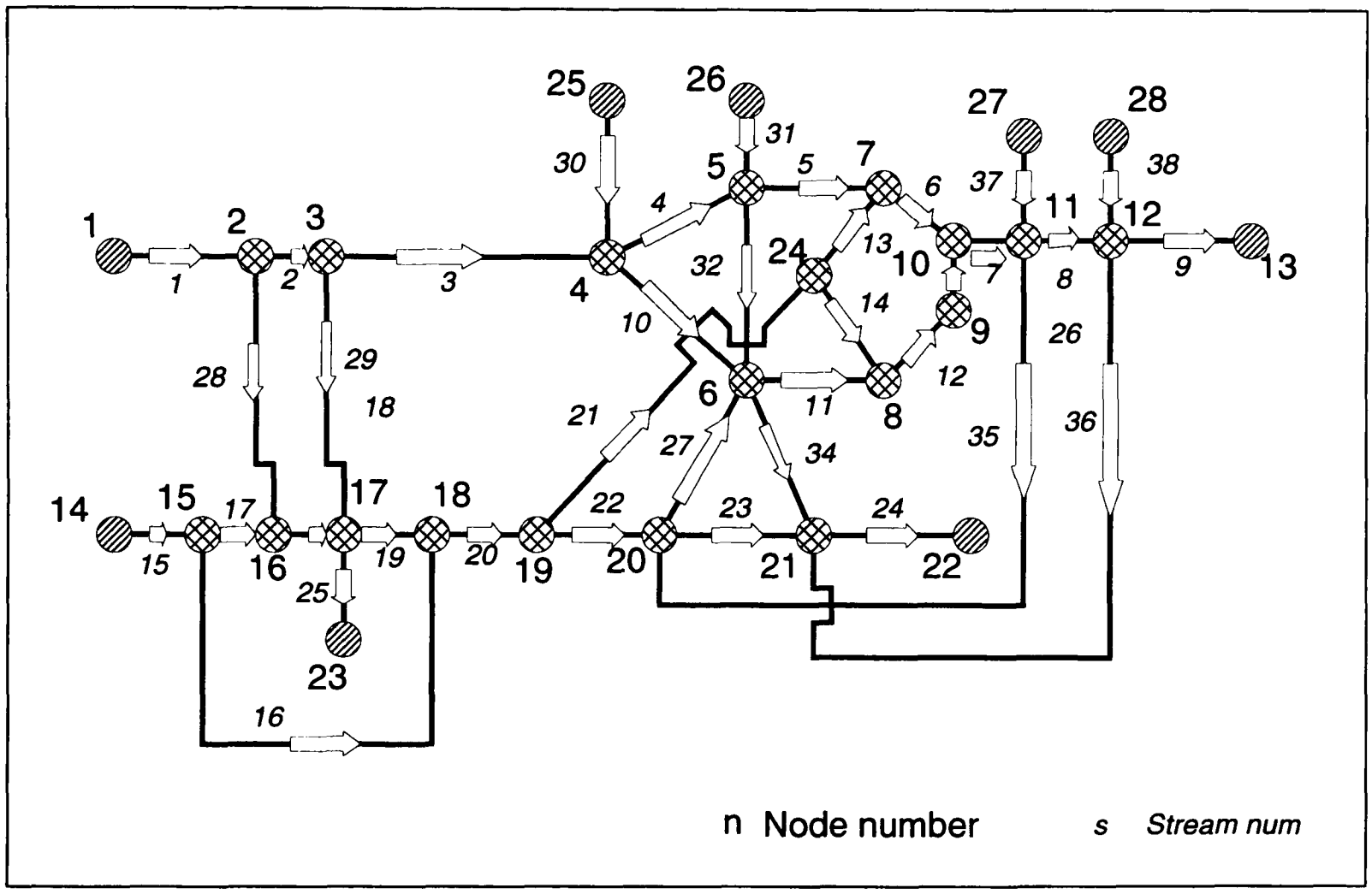

Figure 3.5 Nodes and streams re-numbered sequentially for computation

The numbered nodes and streams is shown in Figure 3.5. These numbers are used as identifiers (or indices in case of linked lists) in all later computation. The primary algorithm for solution of the pellet induration system (Figure 3.3) requires the network to be partitioned

into tree and co-tree graph theoretic structures. The pellet induration networks are not very big as compared to water or gas distribution networks. Although partitioning is possible manually, it would require some insight by the user, concerning which streams should be torn to exclude any cycles in the equivalent connected graph (the one without directed streams) which would then be included in the co-tree structure. The above heuristics and rules of thumb could be embedded into the algorithm to automate this partitioning and free the user from pre-requisite knowledge of mathematical modelling or GASFLO's constraints. Fig 3.6 gives the outline of such an algorithm for tree and co-tree generation.

The stream related data is sufficient for the generation of incidence matrices and, as such, the node data seems to be a duplication. In fact this has been used for validation of the input stream data, otherwise the generated matrices would have been displayed and the user would need to be asked to confirm and trace any of the connectivity errors. This dual entry approach, compels the user to thoroughly sort out the data and check that the components, especially streams and nodes, are consistently identified. For the simulation of a network, this configurational program is run once and the linked lists generated by it are used for multiple number of runs. This configuration needs to be exact, hence it is worth putting slightly more effort in the data preparation, rather than computing or simulating a wrong network.

The stream weight plays a crucial role in the algorithm, it signifies the priority of the stream to be included in the tree structure. In other words it shows, how effective the stream is from the pressure computation point of view, as the tree would be used for computing pressure distribution. The assignment of a weight to streams is also dependent on the implementation state of GASFLO. For example, the present implementation assumes that the cotree streams should be single component streams, later on we can relax this restriction for multiple components cotree streams. This is merely for computational efficiency reasons.

The co-tree branches' execution, computes the flow, for given pressures at the end nodes. It is computed iteratively until a converged value of flow is obtained. If a stream is a multiple component stream, then the pressures for all intermediate interfaces of components

Algorithm for Tree and Co-tree Generation:

Begin

1.0 **Read-in**
- a) Stream and Node data, their composition and connectivity in the Network
- b) Source and Sink nodes' association

2.0 **Generate** Node-Branch & Node-Node incidence matrices of the directed Graph G_d

3.0 **Assign** weight to streams w.r.t. to their composition, connectivity and other constraints

4.0 **Generate** reduced directed graph G'_d from G_d; by deleting zero weight streams (comprised of Leaks only) and, the resulting disconnected nodes

5.0 **Generate** node and stream linked lists for predecessors, as well as, successors of each node

6.0* **Partition** G'_d into disjoint n_s (= no. of source nodes) uni-source sub-networks **use** *Sub-Network Extraction* algorithm

7.0* For each sub-network **use** *Dendrite Generation* algorithm
Extract main pathways by tearing-off any stream introducing cycles
Mark torn streams as co-tree, and non-torn as tree branches

8.0 **Output**
- a) Linked Lists for composition and connectivity of Streams and Nodes
- b) 'NETWORK.INF' binary file having all read-in and generated information, for use of simulation program GASFLO.

Stop;

---

* These would be described separately.

Figure 3.6 The Algorithm for generation of Tree and Co-tree structures from a connected directed graph G_d

would be unknown. Some method would be required, which could assume the consistent pressure at these intermediate end points and compute the flow in the very first component. Then keeping that flow fixed for the stream, the remaining components of the stream are computed for down_end pressures. On executing the final component of the stream, the down end computed pressure is compared with the known pressure and all intermediate pressures

updated for the next iteration. This process of updating would be carried out until the down end computed pressure of the final component converges to the initial given down node pressure of the stream. Active components such as fans, in these multi-component streams, would complicate the situation further. From the fan equation, the pressure gain is inversely proportional to flow passing through it. Whereas in all other passive components like pipes, leaks and beds, the pressure drop is directly proportional to the flow. The computation of multiple component streams as cotree streams, would not only overload the computation, but can also lead to oscillations (in case of streams involving fans) and non-converged results. Hence the alternative, that all cotree streams should be composed of a single passive component, was adopted. This can be regarded as an implementation constraint of the Code. The situations requiring multi-component streams as cotree branches can be easily resolved, by introducing an extra node up stream to the last pipe of the respective stream.

As for cotree streams computation, flow is computed in terms of given (known) end pressures, so the actual equations of the components occurring as cotree streams, are re-written in the form consistent for the computation. This results in a very efficient and fast scheme as it would require just few iterations to converge that set of few equations.

Table 3.1 Rules for weight assignment to streams

Stream Weight	No of Components	Component Nature	Other conditions
0	1	Leak	
1	1	Pipe	Junction Nodes at both ends
2	1	Pipe	Both end nodes are not Junction nodes simultaneously
3	1	Bed	
4	> 1 (multiple)	any valid combination	Multi component stream

Heuristic values for stream weights are shown in Table 3.1. Figure 3.7 shows the excerpts from the output produced by the code showing data for some streams. The stream and node numbers shown here are related to Figure 3.5 for connectivity and can be traced back to Figure 2.3 to see their physical position in the network, via Figure 2.5.

The tree and cotree partitioning algorithms is comprised of three basic steps. The leaks are the best candidate for tearing, as they always exist as single component streams and their computation as cotree stream is simple. Also their removal from the graph will eliminate many loops leaving us with a simpler graph to deal in further steps. In the first step we reduce the graph by tearing off leaks and deleting any of the disconnected nodes. In the second step, this multiple source reduced graph is partitioned to multiple single source networks, taking care that each subnetwork should have at least one sink attached to it. In the third step a tree or dendrite is generated for each of these subnetworks, and all non-tree streams are collated into a co-tree structure. The following subsections explain each of these steps:

Str #	Up_End Nd	C o m p o n e n t s	Dn_End Nd	Strm wtg
1	1	p01 f01 p02	2	4
2	2	b01	3	3
3	3	p03	4	2
.				
.				
.				
24	21	p22 f06	22	4
25	17	p23	23	2
26	9	p06	10	1
27	20	b07	6	3
28	2	l01	16	0

p → Pipe b → Bed f → Fan l → Leak

Figure 3.7 Stream composition, connectivity and assigned weights (Code's Output)

3.5.1 Graph Reduction (or Leaks' Tearing)

In the Graph reduction step of the algorithm, we pick up a stream and check its weight. If it is a 'Leak' stream, having weight zero, then we retrieve its up end and down end nodes, and reduce their degree out_degree and in_degree respectively by one, and mark the stream as 'torn'. After executing all streams of the network, we look for disconnected nodes by computing the total_degree, afresh for each node. The nodes with zero total_degree would be the disconnected ones as they now have zero incidence. The degree of the nodes, which are not incident to any of the leaks, will remain un-affected. Figure 3.8 shows the result of this reduction. The node numbering strategy can help in generating compact incidence matrices, if these to-be deleted nodes are placed towards the end. In this case nodes 25 to 28 were the last nodes.

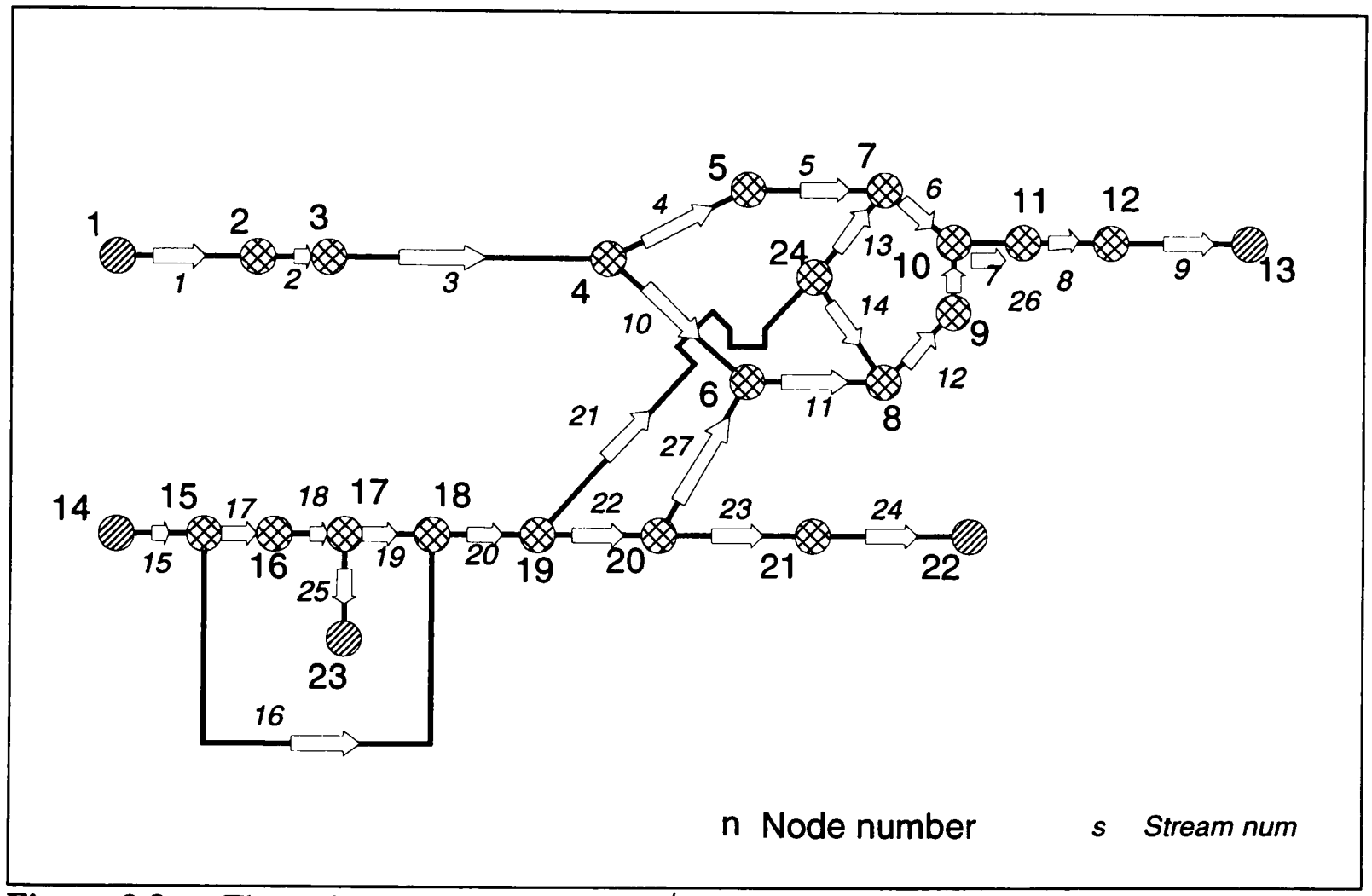

Figure 3.8 The reduced directed graph G'_d, after deleting leak streams and disconnected nodes

Linked Lists Formulation :

The linked lists are very efficient data structures and are well covered in Computer Science literature see for example Knuth 1973a, 1973b, Deo 1974, Syslo et al 1983, Osiadacz 1987, Duff et al 1990 and Ahuja et al 1993. These could be one directional *singly linked lists* or bi-directional *doubly linked lists* providing access to data in forward and backward directions. These could be implemented either using pointers or one dimensional arrays. As FORTRAN 77 lacks pointers, these are implemented using arrays. The assignment of integral numbers as identifiers to nodes and streams, results in integer arrays, which are efficient in storage as well as in computation. The simplest and smallest of the used linked lists, in this network, are for source-sink association. From Figure 3.5, the source node 1, feeds to sink node 13 only, whereas source node 14 feeds to sink nodes 23,22 and 13. This association can be presented by three integer arrays, Src_Nd(), Pntr() and Snk_Nd(). The ith source, with source node number Src_Nd(i); feeds to the sink nodes identified by the nodes Snk_Nd(Pntr(i)) to Snk_Nd(Pntr(i+1)-1). Figure 3.9 shows the linked list structure and its equivalent set form is shown in Figure 3.10. From these figures, the relationship between the second source, node 14, to its sinks, 23,22 and 13 is established by Pntr(2) to Pntr(2+1) - 1, i.e. indices 2 to 4 or array Snk_Nd().

Ind	Src Nd	Pntr
1	1	1
2	14	2
3	0	5
4	0	0
5	0	0

Snk Nd	Ind
13	1
23	2
22	3
13	4
0	5

Figure 3.9 Linked lists for Source-Sink associated nodes.

The step 5.0 of the tree and cotree generation algorithm (Figure 3.6) formulated the linked lists for nodes and stream of predecessors and successors of each node of the network. Since all of the leak streams have been declared as torn streams, they and their associated (disconnected) atmospheric nodes are no longer needed in the algorithm. We concentrate on the reduced graph G'_d, for the generation of linked lists. The further steps of algorithm will be using these linked lists instead of the incidence matrices.

```
Number of Sources : 2

      Source Node Numbers = {  1, 14 }
      Source-Sink pointer = {  1,    2,   5,   0,  0  }
      Associated Sink Nds = {  13,  23,  22,  13  }
```

Figure 3.10 The above Linked Lists represented in set form

The incidence matrices present the complete connectivity of the network. In our case, the maximum node degree is 5, hence in node-stream incidence matrix, a row (associated to the respective node) can have maximum of 5 elements, in respective columns corresponding to the incident streams. Only these elements of the incidence matrix will have +1 or -1 values, depending upon the adopted convention how the incoming and the streams outgoing from a node are to be interpreted, all other columns will have zeros. Each of the column correspond to a stream will have only two non-zero entries corresponding to its nodes, thus in total this $e \times s$ matrix will have only $2s$ non-zero entries. So for larger networks the resulting matrices are sparse, costly for storage and hence undesirable. This connectivity information can be transformed into linked lists, which are well packed, easier to use and can efficiently be ported around among different program units.

In this case the linked lists could be generated either relative to nodes or relative to streams, as these are interconnected. We generate a Predecessor Node List (PNL), Predecessor Stream List (PSL), Successor Node List (SNL) and Successor Stream List (SSL), for all nodes of the reduced graph G'_d.

The procedure is straight forward, we pick up a node, see its incidence, enter the incoming streams and their up_end nodes in the predecessor lists PSL() and PNL(), whereas the outgoing streams and their down_end nodes are entered in successor lists SSL() and SNL(). Independent counters for two categories are maintained for storing or retrieving data from these lists. Obviously the source nodes do not have predecessors and similarly the sink nodes have no successors, so zeros at corresponding locations are substituted.

3.5.2 Sub-Network Extraction

For elicitation of disjoint sub-networks from a multiple source multiple sink network we have to first establish some association between these sources and sinks, i.e. which sinks are being fed by a respective source. As stated earlier that each of these single source subnetworks should have at least one sink. This association is computed from the source-sink information input by the user and network connectivity. According to the assumed flow direction in the streams some of the sinks will be fed in by more than one sources, but from subnetwork's reference a sink can belong to only one sub-network.

For this we first select a source node, such that this is linked to minimum number of sinks, and secondly, the sinks fed by multiple sources should have higher priority to be counted towards the sources with minimum sinks. In fact, this criteria fulfils our aim that each source should have at least one sink assigned to it. For example in the considered case, the source node 1, feeds sink node 13, which is also fed in by the source node 14. Now if node 14 is treated as first source, then the corresponding sub-network will include all its sink nodes, thus leaving the source node 1, with no sink node, this is an undesired situation, which should be avoided.

After deciding the order in which the sources would be selected, we extract sub-networks, from the reduced directed graph G'_d, one for each source node. The algorithm (shown in Figure 3.11) for this extraction consists of two passes. In first pass, we start from the source node and trace its successor nodes and put them to *fan-out* list. In the second pass, we start from the sinks associated to the respective source node, and trace the predecessor nodes and list them into *fan-in* list. The nodes common to both lists constitute the sub-network, which can be analyzed separately for computation to reduce the complexity. The sub-network attribute for these nodes is set up.

For subsequent sub-networks, after selecting the common nodes from fan-out and fan-in lists, the sub-network attribute of each of these nodes is checked to verify that only those which are not already included in any of the previous subnetworks, are included in the present subnetwork.

This process is repeated for all source nodes. The resulting node lists for these sub-networks will be disjoint, but physically these are still connected by some of the streams

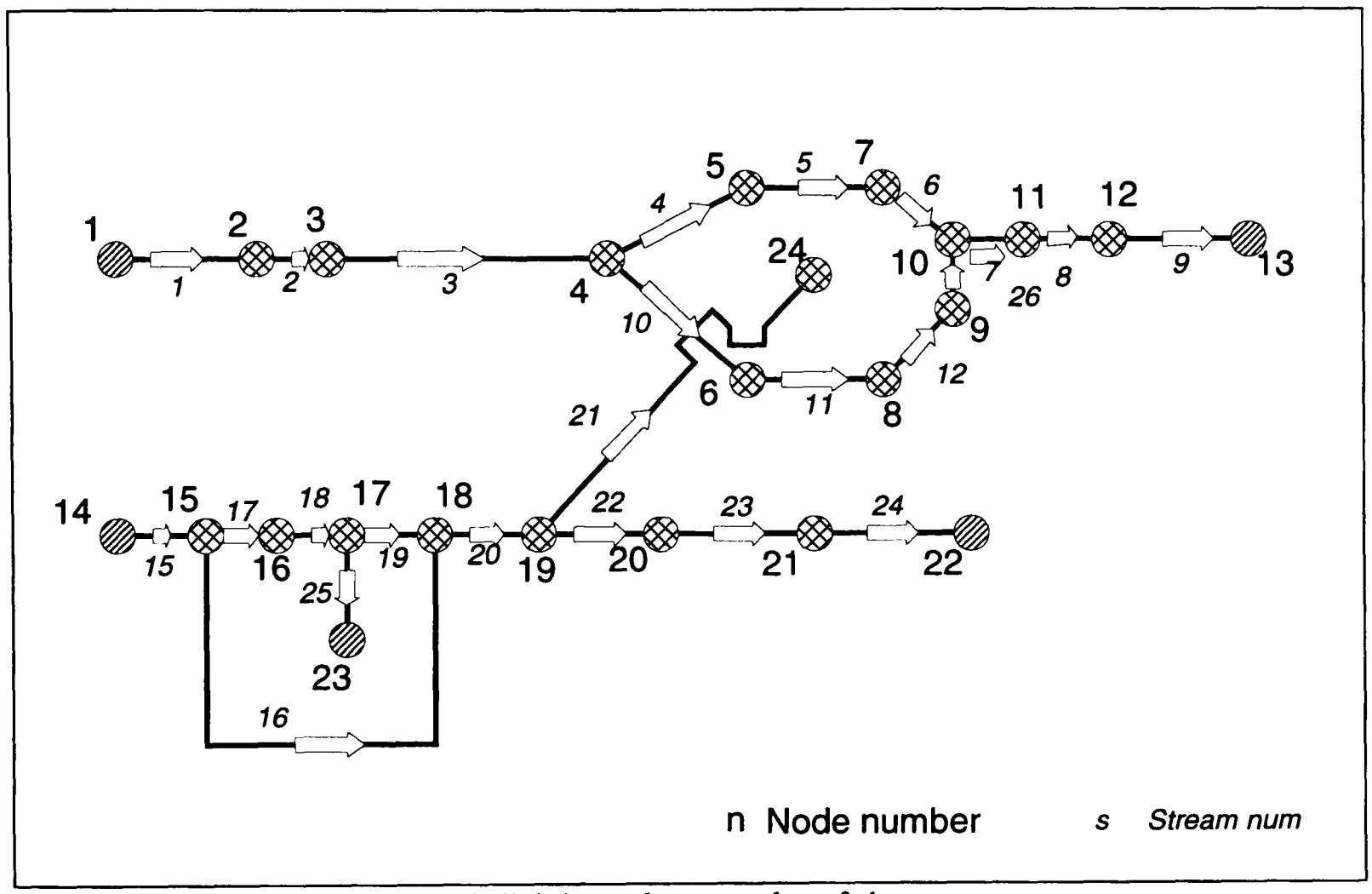

Figure 3.12: The two resulted disjoint sub-networks of the system

joining the nodes from different sub-networks. So as a last step of the algorithm, all the streams (of G'_d) are scanned one by one; if its end nodes are not in the same sub-network, then it is declared as a torn stream. The resulting sub-networks for the considered example are shown in Figure 3.12.

The purpose of extracting sub-networks, from G'_d, was to reduce the complexity in the system. According to the Graph Theory, any multiple source network, of s sources, e edges and n nodes; will have l fundamental loops, where

$$l = e - n + s$$

Putting in the values for our example, Figure 2.5, there will be (38 - 28 + 2 =) 12 loops. For tree generation these loops are to be detected and each one of them is to be torn off, leaving it as a loop-less or acyclic connected graph.

The partitioning of the network to single source sub-networks, reduces the complexity of the problem enormously. From the resulting sub-networks (shown in Figure 3.12), it is not only easy to visualize the loops, but also the node, edge and consequently number of loops have decreased significantly. In these sub-networks the above relation reduces to

$$l = e - n + 1$$

(= 13 - 13 + 1 = 1 for Ist sub-network)

(= 11 - 11 + 1 = 1 for 2nd sub-network)

One could ask that where the other loops have gone, in fact they have been deleted in the intermediate steps; while tearing the leak streams and the streams linking the sub-networks. Instead we have torn more streams than the required, as a consequence, now we have two independent sub-networks, disconnected from each other, whereas the 12 tearing stream restriction would have left us still with a connected network. The independent sub-networks can be dealt with individually.

As mentioned in the last section 3.4, while reviewing existing algorithms, the Travers algorithm resulted in forest of trees only if the original graph of the network was disconnected. Nearly all the authors solve multiple source networks, by treating one of the sources as primary source and use its pressure as reference pressure for computation of node pressures, most of them (e.g. Osiadacz 1987, Boulos and Ormsbee 1991) use loop based

Algorithm for Sub-network Extraction:

Begin
1.0 **Read-in**
a) Successor Node and Stream (linked) lists for all Nodes
b) Predecessor Node and Stream (linked) lists for all Nodes
c) Source-Sink nodes association (linked) lists

INITIALIZATION:
$\mathbf{P}_{res} = \mathbf{P}_{rev} = \phi$; *Fan-in* = *Fan-out* = ϕ

ITERATION:
For each Source Node
2.0 Find *Fan-out* list
2.1 Select Source node, add to *Fan-out* list
2.2 Pick up ith node, **x**, from *Fan-out* list, mark it as 'picked'
2.3 Trace all **y** s.th. **y** = Successor(**x**), Add **y** $\rightarrow$ *Fan-out* list
2.4 **if** (.not. all elements picked) **then**
i = i + 1; **goto** step 2.2
endif
3.0 Find *Fan-in* list
3.1 Select the associated Sink list, L_{sink}, for selected Source node; j=1
3.2 **for all** s ε L_{sink}
3.3 Add s $\rightarrow$ *Fan-in* list
3.4 Pick jth node, **y**, from *Fan-in* list, mark it as 'picked'
3.5 Trace all **x** s.th. **x** = Predecessor(**y**), Add **x** to *Fan-in* list
3.6 **if** (.not. all elements picked) **then**
j = j + 1; **goto** step 3.4
else
Select next sink from L_{sink} i.e. **goto** step 3.3
endif
end for loop;
4.0 Take intersection of lists; $\mathbf{P}_{res}$ = *Fan-out* $\cap$ *Fan-in*
5.0 **if** (.not. First source node) **then**
Delete from $\mathbf{P}_{res}$ the nodes already marked for previous sub-networks
$\mathbf{P}_{rev}$ = Nodes for all previous sub-networks
$\mathbf{P}_{res} = \mathbf{P}_{res} - \mathbf{P}_{rev}$; $\mathbf{P}_{rev} = \mathbf{P}_{rev} \cup \mathbf{P}_{res}$
endif

6.0 Mark all $\mathbf{P}_{res}$ nodes as this sub-network nodes
end For (each Source Node) loop:

7.0 **Tear** off all streams connecting different sub-networks

8.0 **Output** Node lists for each sub-network
Stop;

Figure 3.11 The algorithm for sub-network extraction

methods for formulation of the problem and define pseudo-loops between the different sources or fixed grade nodes. Whereas Boyne 1970, reduced the multiple source network, to single source networks by tearing of the secondary sources. He tried two alternatives; tearing of the pipe midway in the path linking the primary and secondary sources, and the edge adjacent to each of the secondary source; and recommended the latter approach for better results. Like others, he also computed the pressure distribution in the network with respect to the primary node.

Pellet induration networks are different from other fluid flow networks, the main sources (numbers 1 and 14 in Figure 3.8) have exactly similar affect on the pressure distribution of the respective subnetworks, so logically should be treated as primary sources. So the well used concept of single primary source is physically not valid. Hence for each of these extracted sub-networks, the corresponding source should be treated as primary source for computation of pressure distribution. In the following subsection the extraction of tree from each of these subnetworks is explained.

3.5.3 Dendrite Generation

To reduce these sub-networks to tree structure, we consider a sub-network at a time. These could be taken in any order. This algorithm consists of a single pass, and is described in Figure 3.13. We start from the source node, put it in the *list*, select from the *list* first unselected node, look at its successor nodes, if they are not in the *list*, include them in the *list* and mark the respective successor branches as tree branches (i.e. marking the torn status as .false.). If the successor node is already included in the list, then, this implies that the (successor) node has multiple incoming streams, as it has been already accessed through another node. By definition of Tree each node should have one predecessor stream, so a decision needs to be made, concerning which stream should be retained in the tree. Hence, we look at the predecessor streams of this (successor) node, and decide, using the weight criteria, which of these streams should be torn. The one having higher weight, means have higher priority, so should be retained in the tree, and the other with lower weight should be

marked as torn stream. Then the next node of the *list* should be selected and the above procedure repeated. This selection and marking procedure is continued till all the nodes of the sub-network are exhausted.

The step 5.0 of the *Dendrite Generation* algorithm (Figure 3.13), covers the heart of the tearing procedure, so it is described in detail, as compared to others. It also shows how linked-lists can help to implement the algorithm in a more comprehensible way.

Finally, after executing all the sub-networks, all torn streams are collated and these form the Co-tree, whereas the rest would be Tree streams. By Graph theory the resulting tree is in fact a Forest, that is a combination of trees, not a single tree. In loose terms we call it tree. The final result of algorithm i.e. the resulting tree and cotree structures are shown in Figure 3.4, while discussing the computation of the network (section 3.3).

3.6 Algorithm for Leaks' Incremental Variation

The efficiency of the process is dependent on the conservation of thermal energy, i.e. the heat extracted from the pellets in cooling stage should be optimally utilized by the other stages. Ideally, there should be no leaks between the system and atmosphere; but in practice, the gaps between the packed bed and the partitions of the enclosing chamber have to be provided, for smooth functioning of the process. The different zones have different pressures, so the process gas flows between them thus causing leaks between different zones and between the system and atmosphere. The model should be able to optimize the process, by first adjusting the model parameters, like leaks areas whose exact values are not known but approximate ranges are given, and later adjusting the flow and pressure distributions as close as possible to the measured values. Then these distributions should be treated as reference and further the simulations be carried out by varying controllable system variables, like valve opening and fan wattage.

Algorithm for Dendrite Generation:

begin
1.0 **Read-in**
a) Successor Predecessor Node and Stream Linked Lists, related to nodes
b) Sub-Network Node lists c) Source-Sink Node Association
d) Torn State for all Streams

ITERATION :
for all sub-networks

2.0 Initialize *list* = ϕ ; i = 0

3.0 Select the Source node,**s** ; Add **s** $\rightarrow$ *list*

while (.not. all_node_picked) **do**
4.0 i = i + 1 ; Pick the ith node, $\mathbf{x}_i \in$ *list*

5.0 Look at all of its successors and formulate tree:
do j = 1, Out_Degree($\mathbf{x}_i$)
$\mathbf{y}_j$ = Successor_Node($\mathbf{x}_i$)
if ($\mathbf{y}_j \in$ *list*) **then** it has multiple Predecessors, trace them; *Parent* = 0
for all k = 1, In_Degree($\mathbf{y}_j$)
l_k = Predecessor_Stream($\mathbf{y}_j$)
Tear if Single pipe component stream
if (Weight(l_k) = 1) **then**
Torn(l_k) = .True. -- it is marked l_k as Co-tree branch
else this is any other multiple component stream
Torn(l_k) = .False. -- it is marked l_k as Tree branch
Parent = *Parent* + 1
endif
end for
Warn **if** (*Parent* $\neq$ 1) and Flag $\mathbf{y}_j$ and l_k etc
else
$\mathbf{y}_j$ is not present in the *list*, so include it
Add $\mathbf{y}_j \rightarrow$ *list*
endif
end do
6.0 To pick the next node of the *list*; **goto** 4.0

end of for loop:

7.0 Collate Tree and Co-tree structures by scanning Torn state of all streams

8.0 **Output** Tree and Co-tree structure for the whole network

Stop;

Figure 3.13 The algorithm for dendrite generation

A contributing factor to the system complexity is unavailability of complete information about the system. Mathematical modelling is very suitable tool to tackle this problem. For instance in pellet induration system case, we know that the leaks exist at such and such points, these are rectangular in shape, having width as the width of the bed and height about 5-10 centimetres. This height depends on the loaded position of packed bed, it is specific to each leak. In fact the height of the leaks is not known exactly, so we treat it as a parameter in our computation.

In practice the pressures are maintained, so that the input regions should have pressures slightly below atmospheric. Ideally these should be exactly atmospheric to give zero external leaks (leaks between system and atmosphere), but that is difficult to maintain and susceptible to pressure fluctuations. If these regions pressures get higher than the atmospheric, then the process gas will be pushed out of the system, which will not only give rise to heat loss, i.e. system's inefficiency, but can also lead to disastrous \ hazardous situations. This target value of below the atmospheric provides incoming leaks and is also environmentally safe. How these incoming external leaks affect the system's overall efficiency, will be analyzed later in chapter 5.

In section 2.4.2, the mathematical model of a leak was described, physically it is flow through a gap connecting the two adjacent regions. Its magnitude is dependent on the region pressures and the cross sectional area of the gap. As compared to other components, pipes and packed beds, the resistance offered to this flow is negligible, which makes it very sensitive, even very small increases in leak area can give very large values of flows depending upon respective end pressures.

All leaks are treated as torn streams (co-tree branches). Their computed flows are used to correct the flow distribution in tree, which is further used to compute pressure distribution in the network, and in next iteration these pressures are used to evaluate the torn stream flows. Hence if for some value of areas, very large flow for a leak are introduced then

it is possible that this may lead to distribution with non-physical results, from which the system may not recover.

For example consider Figure 2.3, if the area for leak connecting R10 to Atmosphere A-8, is changed say increased, then the flow through this leak coming into the system will increase (assuming the $P_{R10} < P_{atm}$), which will increase all outgoing flows from the region R10, and by KCL execution it will decrease the incoming flow in tree branch i.e. linking R10 to J07. This will decrease flows in the other branches of tree upstream to this point. Similarly this new flow distribution will change the pressures at the tree nodes. Suppose the leak area is such that it produces the incoming leak flow big enough that it reverses the direction of incoming tree branch from R10 to J07. This is non-physical, because the fans 1A and 1B upstream to J07 will not allow that. The flow and pressures form a coupled system of equations and interact to the changes in each other, so in most of the cases the system / code will recover from such unwanted situations, but this recovery will depend upon the input value of area.

To save user from trial and error, the leaks' areas are incremented successively with a fixed step size. The algorithm is shown in Figure 3.14. Initially the program starts with a set of default values for all NT_{leaks} and reads in the value of step size and the final areas to be set, for the respective NV_{leaks}. The subroutine *LEKINI* evaluates the maximum number of steps required by the program to have all leaks set to their respective final areas. First the converged flow and pressure distributions are computed for default leak areas and then the leak areas for corresponding NV_{leaks} are relaxed by a step size each, in the direction of their final values, then using the converged distributions, new flow and pressure distributions are computed. The subroutine *LEKRLX* keeps track of the leaks areas and related statistics, e.g. which are to be increased/decreased, which have achieved their final areas and which are yet to be changed. The computation continues till the converged distributions are achieved for all leaks areas relaxed to the required final values.

A recovery mechanism is also implemented to deal with unsure situations. The flow and pressure distributions and leak areas are temporarily saved for every successfully completed step. If for some step the system fails to recover and comes up with non-physical results, than the previous step saved values are retrieved. As these are already converged values so further computation is not required and so program stops informing user about the failure to reach the final desired values.

The aim of this leak areas relaxation is to achieve a flow distribution as close as possible to the measured results. The experimentally measured result include the flows in streams containing fans, and region node pressures. The so achieved distribution, can be treated as reference distribution and other parametric studies could be carried out.

3.7 Temperature Computation and Communication with INDSYS Code

One of the primary aims of the GASFLO model is to compliment the computation of heat distribution evaluation codes. These codes are now mature and well used in iron industry, two of them, INDSYS (Cross and Englund 1987) and CASCADE (Patel et al 1993), have been developed here, at Greenwich University (formerly Thames Polytechnic).

Heat distribution codes are complex, and meant for microscopic studies. They solve the respective partial differential equations, resulting from conservation laws, and predict the spatial temperature profiles for process gas and solids. They take into account the chemical reactions and heat input to the system by different sources, like burners etc. Due to involved complexity of the process these are compute intensive and modelled separately. Also to minimize the computational load these are applied to the part of the system, rather than the whole system. For example the INDSYS, computes for the packed bed only, whereas CASCADE has facility to compute flow velocity profiles alongside the temperatures, so it can be applied to only few zones of the induration system. For computation, they require the flow and temperatures of the streams adjacent to their domain of application. Presently, in the

Algorithm for Leaks' Area Variation:

Begin

1.0 **Read-in** Number of Leaks' areas to be modified, NV_{leaks}, StepSize, initial and final areas of each of these leaks

INITIALIZE

2.0 **Evaluate** MaxSteps for leaks' area variation by calling
†*LEKINI(... NV_{leaks}, StepSize, $LArea_{ini}$, $LArea_{fin}$, MaxSteps...)*

3.0 **Initialize** Step = 0
LArea = $LArea_{ini}$

ITERATE

4.0 Step = Step + 1

5.0 **Compute** ***Flow*** and ***Pres***sure Distributions in the network;
Use General_Network_Computation Algorithm (Figure 3.3)

if (Results Physical) **then**

6.0 **Backup**
state variables ***Flow*** → ***OldFlow*** and ***Press*** → ***OldPress***
leak areas *LArea* → *OldLArea*

else !non-physical results -- step back to previous step areas

7.0 **Retrieve** the last step backed up variables
OldFlow → ***Flow*** and ***OldPress*** → ***Press***
OldLArea → *LArea*

go to 9.0

endif

if (Step > MaxSteps) **go to** 9.0

8.0 **Compute** leaks' areas, adjusted for the next StepSize, by calling
‡*LEKRLX(NV_{leaks}, Step, LArea)*

9.0 **Compute** ***Temp***erature Distribution; **Use** Temperature_Distribution_Computation Algorithm (Figure 3.16)

Stop;

---

† ‡ The procedures *LEKINI* and *LEKRLX* are subroutines discussed in text

Figure 3.14 The algorithm for Leaks' Area Variation

absence of any code like, GASFLO, the desired distributions are estimated by trial, which is not only a laborious process but also requires a good experience and detailed knowledge of the field.

GASFLO should provide means to communicate with these codes. Since INDSYS is available so Figure 3.15 shows the would-be communication between these two codes. In principle, each one of these should be able to run independently, so the variables expected from the other counterpart are fed in by some guessed values. INDSYS requires the flow and temperature of the streams feeding into the input regions of the induration system. The other data required as input is comprised of heat sources, pellets data and network configuration. It provides the two dimensional (assuming symmetry in third) spatial temperature profiles for gas and solids in the packed bed. These temperatures can be averaged to find the temperature of gas in each of the packed beds, which are required by GASFLO for execution of Ergun equation.

GASFLO requirements are discussed in detail in sections 1.2 and 2.3, briefly these are, system configuration; components' data e.g. pipes diameters, lengths, fans' wattage, beds' areas and pellet related data; boundary nodes data; and temperatures of gas in packed bed. On execution it provides flows and temperatures of all streams and pressures and temperatures at all nodes of the network.

INDSYS was written in 1970s, in BASIC and has captured good international market ever since, whereas GASFLO is still in developing state and is written in FORTRAN 77. Actually, the two should be parts of one code to solve the induration system without any external interruption, but in view of the background sited above, that is not possible and will require good amount of work. Alternatively, an external shell with filter like utilities, can be written which can execute each one of these programs and the filters can parse the output files to extract the required information, feed that as input to the other code and the other code is executed. This cycle is carried out till overall convergence is achieved. Before indulging into the proposed shell writing, the strategy was tested manually and found that

only 4/5 iterations are needed for the overall convergence of the combined system, so even the effort of writing shell was not thought to be worth while.

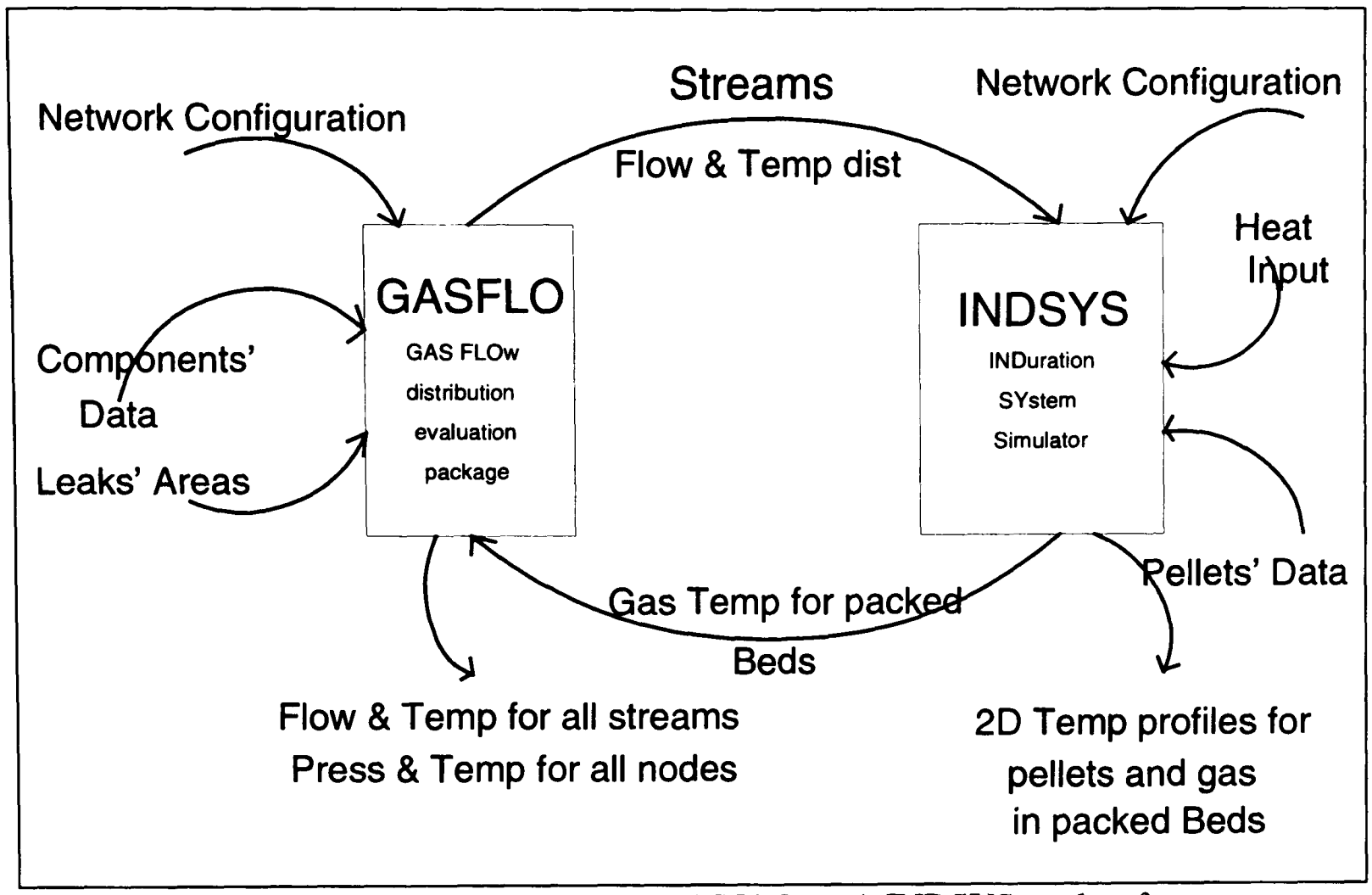

Figure 3.15 The data transfer between GASFLO and INDSYS codes for temperature computation

In the following section the algorithm for computation of temperatures in the GASFLO will be discussed.

3.8 Algorithm for Temperature Computation

In section 3.1 it is described that the temperature computation is dependent on flows, but instead the flows have a weak coupling with temperatures. Fincham and Goodwin 1988, also suggest that the equations for natural gas pipe networks should be decoupled, the internal loop should compute flow and pressure distributions, whereas the external loop should

compute temperature distribution. Same strategy is used for temperature computation in GASFLO. Figure 3.16 shows the respective algorithm.

In external loop, the program INDSYS is executed, using guessed flow distribution, and it gives the temperature profiles for gas and solids in packed beds. These can be averaged for each of the beds and imported into the GASFLO. In internal loop, the converged flow and pressure distributions are found using *Leaks_Area_Variation* algorithm (see Figure 3.14) for specified leak areas, then temperature distribution is computed.

For temperature distribution, the component models (given in section 2.4.2) show that the nodes and pipes play an active part. Although packed beds are the most active components, but their temperature computation is done in INDSYS. This will cause a discontinuity of temperature in the streams containing the beds, i.e. the temperature of process gas entering the bed will be different from the temperature of gas leaving the bed. At internal nodes of the network, i.e. junctions or regions, the streams with different flows and different temperatures meet, so analogous to KCL, the heat energy should be conserved. Using this energy balance equation the node temperature is computed, which is assigned to all streams leaving the respective node. Though this will give a unique temperature to each node, but there is nothing equivalent to KVL, as different streams entering a node can have different temperatures. As compared to pressures, the node pressures completely specified the pressure distribution in the network, but for temperature distribution, each stream's up-end temperature can be same as the up-end node temperature, whereas the down-end temperature of a stream may have different value then the down-end node temperature.

For this computation, we follow a scheme similar to one used for flow distribution computation. Temperatures at source nodes and in packed beds are known, so temperatures are computed from source to sink direction in tree. While executing components, the corresponding temperature equations for each of the component will be computed.

Algorithm for Temperature Computation:

Begin

1.0 **Read-in** Temperature at source boundary and atmospheric nodes; parameters $Tolrnc_{int}$, $Tolrnc_{ext}$, *MaxItr* etc.

2.0 **Obtain** Converged ***Flow*** distribution in the network for pre-set Leaks' areas;
Use Leaks'_Area_Variation Algorithm

EXT-ITERATION

3.0 **Import** from **INDSYS**† the process gas ***Temp***erature for all packed beds

ITR = 1

INT-ITERATION

4.0 **Compute**‡ ***Temp***erature distribution in tree i.e. for all internal nodes and tree branches (in Source → Sink direction)

5.0 **Compute** ***Temp***erature distribution in co-tree

6.0 **Evaluate** $ERROR_{int}$ (i.e. relative error using $Temp_{prev}$ and ***Temp*** at internal nodes)

if (ITR ≥ 2 .and. $ERROR_{int} < Tolrnc_{int}$) **go to** 8.0

7.0 **Update** temperature distribution $Temp_{prev}$ = ***Temp***
ITR = ITR + 1
go to 4.0

8.0 **Evaluate** $ERROR_{ext}$ (comparing the computed packed bed gas input temperatures for current and previous iterations)

if ($ERROR_{ext} < Tolrnc_{ext}$) **go to** 11.0

9.0 **Export** converged ***Flow*** and the recent ***Temp***erature distributions to **INDSYS**

10.0 **Execute INDSYS** with new ***Flow*** and ***Temp***erature distributions
go to 3.0

11.0 **Output** ***Flow***, ***Pres***sure and ***Temp***erature distributions of the network

Stop;

---

† **INDSYS** is an other code for computing heat concentration in pellet induration systems.
‡ See text for details

Figure 3.16 The algorithm for Temperature Computation

After evaluating tree (node as well as stream) temperatures, the co-tree branches are executed. The co-tree branches, by definition are single component (see section 3.4.1) streams, mostly comprised of leaks and few are of pipes and beds. The proposed temperature equations for leaks are simply assignment of up-end node temperature to down-end of the leak stream. This assignment is very logical and enables us to have the actual stream temperature for all incoming streams, e.g. if an external leak is coming into the system (i.e. $P_{Reg} < P_{Atm}$) then it will have atmospheric temperature, but if it is going out of the system then it will have respective region's (i.e. up-end node) temperature as its temperature. The packed bed down-end temperature is also an assignment to the temperature computed by INDSYS and fed in to GASFLO as parameter. The pipe co-tree branches, will assume the up-end node temperature as temperature at their input end, and would compute by executing their temperature model, to find the exit end temperature.

The *ERROR* is computed by comparing the present iteration node temperatures to the previous iteration ones. If *ERROR* is larger then the specified tolerance, then temperatures in tree are re-computed, but now using the last iteration co-tree temperatures. So the computation of temperature in tree and co-tree, steps 4.0 and 5.0 of the algorithm (Figure 3.16), is continued until the convergence is achieved, which completes the internal iteration loop.

These converged values for the flow and temperature of the streams feeding into the input regions of the induration system are exported to the INDSYS, where these are used as input. This step needs the manual editing to the data files for INDSYS, and requires some understanding of INDSYS how it represents the network configuration and different connecting streams.

The external iteration is carried out until the bed temperatures provided by INDSYS for successive iterations become constant. It was experienced that about 4/5 external iterations worked for the tested data sets. This interaction of two codes is further elaborated in section 5.5.3.

3.9 Advantages of Present Approach

The developed code GASFLO is based on the unit based approach and it used generic algorithms for the evaluation of airflow distribution and network partitioning. This approach has the following advantages over the classical network solution methods:

- Each of the system component categories are modeled as independent modules, which are extensible e.g. the mathematical model of any of the components can be refined and changed by modifying the respective module.
- The heuristics based on working experience or for computational efficiency reasons, can be embedded into the algorithm for network partitioning, by assigning weight to the streams of the network. This reduces the network into unique tree and co-tree structures, which are used in further computation.
- Each component is picked up and executed in order of its connectivity in the network. The component can interrogate themselves for their different states and execute accordingly, e.g. the execution of *nth* pipe will first verify its inclusion either in tree or in co-tree structures, if it is in a tree branch then it will accept ***Flow*** and $\boldsymbol{P_{in}}$ as inputs and compute $\boldsymbol{P_{out}}$ as its output, but if it is a co-tree branch then it will accept $\boldsymbol{P_{in}}$ and $\boldsymbol{P_{out}}$ as its inputs and compute ***Flow***.
- More components could be added, either by enhancing the functionality of the existing modules or units, or, by making them as independent new modules, hence the overall model can be enhanced or refined. Their physical connection to streams and nodes be defined in the network configuration.
- Each unit or module is solved as an independent module in terms of the parameters provided by the neighbouring units of the network, and it computes the locally converged values of state variables, which are then, passed onto the next unit in the sequence. This contributes towards the overall convergence of the system and reduces its computational time requirements.
- Each module can use different numerical scheme, suitable to the nature of its equations, for local convergence. Presently, *One Point Iteration Method* (see section

2.6) was used, for all of the modules, as it suited best to the used equations. Provisions for different views of computation of a module, i.e. computing different parameters in terms of the others, as mentioned above for a pipe in tree or in co-tree, could be managed by re-writing the pressure-flow equation in different forms.

- Flexible graph theoretic algorithms, which give insight to the problem and its solution. These were very helpful to the evolving nature of the code and to the complexity of the application domain. For example, the leaks areas relaxation and temperature distribution computation were easily implemented. After depicting the goal, what is needed to be done? the solution strategy provides a way out how it can be elegantly done.

Due to presence of all the above advantages, the resulted code GASFLO was able to cope with varied situations. These included the addition of Valves to the model; enhancing the role of packed bed as co-tree stream along with being a tree stream, to allow the '*cross-flow*' situation in the network; incremental variation of leak areas; and replacement of zone unit by its component fundamental units, regions and packed bed. These implementations will be discussed in detail in Chapter 4.

Chapter 4
Application of Software Engineering Concepts to Simulation Tools: GASFLO - A Case Study

4.1 Software Engineering (SE) Concepts and Techniques

The term 'Software Engineering' was coined as a result of a conference held in West Germany, on software crises in 1968. It implies that software should also be developed like any other engineering product, as it has similar usage, and manufacturing constraints. A successful software project has to meet time and budget constraints (Bentley 1987) and the factors like re-use, skilled manpower and efficient tools also contribute in a similar way towards higher productivity as in other engineering disciplines. The attributes of high quality, reliability, robustness and ease of use imply the same meanings to software as to any other engineering product. Hence, the approaches used by other engineering disciplines, to achieve these attributes and satisfy these restraints, could analogously be applied for software development.

Most mechanical products, especially the components used by these products; for example peripheral devices of a Personal Computer (PC) namely printers, mice, keyboards, monitors, disk drives, even central processing units and chips; are produced by different vendors and assembled together by the manufacturer. Each one of these components, is

complete in itself, performs the intended function and conforms to a standard interface, which enables it to fit in consistently to its target product. In case of malfunction the faulty component is tracked down and swapped by its counter part, without effecting other components of the product.

Accordingly, to induce reliability and re-usability in software, this sector needs to learn from the experience of other engineering disciplines, so that the components or modules of a software should behave like black boxes and have qualities to:

- Perform their well-defined intended functions;
- Work independent of each other; and
- Conform to some pre-set standard interfaces.

The increased availability of computers at relatively low prices with higher computational power has led to their wide spread use in diverse application fields, which suggests that the future software demands would be even higher than today. The gap between demand for the software and the software developed would ever increase unless some disciplined approach is adopted. From the software developer's point of view to handle this challenge, the process of software development should be:

- **Efficient** to meet time and budget constraints;
- **Productive** to respond to these increased future demands;
- **Error-free** or have least minimal possible errors, which necessitates that more attention should be focused at the design stage. Since the rectification of an error after implementation/coding is 30 times more expensive than if it is fixed at design stage; and
- **Maintainable** to fix the bugs as well as to respond to the changing needs of the users and varying specifications required by the clients.

These objectives for software quality and development process can be accomplished by adopting appropriate Software Engineering (SE) principles and techniques. The commercial or non-scientific domain have widely benefited from these SE techniques and from the well known outcome the object orientation (OO) technology, which is claimed to be the latest '*silver bullet*' to software crisis (Cox 1990). A detailed discussion on OO will be given later, in section 4.6. Interestingly, about +80% of the literature cited in context of OO discusses benefits of the object orientation without mentioning how it can be achieved or implemented. By contrast, here we will be applying the SE techniques to resolve the problems encountered in the development of software in scientific domain. We will see that the finally resulting code, GASFLO, will have all the properties promised by software engineering community and it satisfies the initial objectives laid out in section 1.4. The intermediate benefits to the software developer (though may not be mentioned explicitly, but) will be evident as it provides a well defined systematic approach rather then the conventional trial and error approach to software development.

4.1.1 Software Quality Objective :

The modules are likely to achieve the above stated 'black box' qualities, if they have the following properties:

Information hiding: A software module should have access only to the data needed by it. All information relating to the methods used to transform its input to output is encapsulated inside the module and kept hidden from other modules. The user of the module should only know about its input and output i.e. interface with other modules.

Cohesion: Modules should be cohesive and preferably perform a single well defined function. In case of more functions, these all should be complementary and supportive to each other.

Uncoupling: The modules should be uncoupled so these can function independently. Their dependence on each other should be explicit and minimal.

Manageable: They should have a reasonable size, neither too large to understand nor too small as it will increase the number of total modules present in the software and make their management complex.

The information hiding principle was proposed by Parnas 1972, and it is the key concept to software engineering and object orientation.

Above are the main properties which every software product should be expected to have, irrespective of its field of application. The complete list of properties for an ideal software product, or the famous 'ilities', can be found in any standard text on software engineering (e.g Pressman 1988, Sommerville 1989 and Fertuck 1992). Additional properties specific to an application field can be seen in the respective application area literature. Petridis et al 1991 and Knight and Petridis 1992, have given one such list for computational fluid dynamics (CFD) software which includes; efficiency, correctness, robustness, extendibility (covering design simplicity and modularity), reusability, compatibility, portability, verifiability, integrity and ease and efficiency of use.

4.1.2 Software Development Process Objective :

From the software developer's view the produced software should have minimal overheads and be completely documented. It should satisfy some quality as well as productivity metrics. The process should support the team work, where the personnel related to software development could swap their roles, like any other manufacturing industry. Some tool or language which could provide clear communication among different groups of humans concerned with the software; designers, developers, programmers, users and clients; is required. Such tool or language should be versatile enough to record all important phases of the Software Development Life Cycle (SDLC).

The design process should support iteration, that is, it should have minimum cost to accommodate changes, which could be either bug-fixes or changed requirements or

specifications. The graphical language with set of rules and different notations for different phases is an ideal solution. It provides simple, clear, unambiguous and easily comprehendible communication and encourages all concerned to participate fully in the process.

To achieve these objectives SE provides a disciplined approach to all phases of SDLC, which enhances the understanding of the problem domain and its solution and clearly records the process how this solution is achieved.

4.1.3 Software Life-cycle and SE Techniques :

The software life-cycle, in general, can be partitioned into the following four stages:

- **Analysis** - to analyze the problem domain and determine what is required.
- **Design** - how the software should be designed so that the resulting code could fulfil what is required and include the most of the qualities mentioned in section 4.1.1.
- **Implementation or construction** - transformation of designed software into a tangible code using some computer language.
- **Maintenance** - has varied meanings from different perspectives. It is the effort required to keep the code running for its later life. For large commercial software, it is the support provided to the user after delivery of the code, mostly bug fixing (Wilde and Huitt 1992). It could be to meet the initially set goals or to improve efficiency or even to incorporate the creeping features which are required by the user lately after using the program.

Most of the authors agree to the above categorization of life-cycle; but some do extend it to as many as seven stages treating feasibility, problem specification and testing into separate stages; whereas some like Jackson 1983, contract it to only two stages of 'specification' and 'implementation', although his two stages span the above mentioned four stages of SDLC. The mutual boundaries or interfaces of these stages are not exactly defined and also vary from author to author.

The terminology jargon is quite common in SE literature, even the definition of the term 'software engineering' differs from author to author (McDermid 1990). The other such term is 'methodology' which is stated as miss-used word instead of method' by Jackson 1983, whereas Holloway 1991 comments that any method supporting at least one stage of SDLC can be regarded as methodology.

A range of methods (or methodologies) exist providing techniques to carry out these stages, Gane and Sarson (1979), Yourdon (DeMarco 1978), JSD (Jackson 1983), Structured System Analysis and Design Method SSADM (Ashworth and Goodland 1990), Object Oriented Analysis and Design OOA/OOD (Coad and Yourdon 1991) and Modular Approach to Software Construction Operations and Test MASCOT (Moses and Jackson 1991) are some which are mostly used. These all have their own advantages, inclinations and disadvantages; for example SSADM fully supports the first two stages and the last two are partially supported, JSD is mostly used in the domains where the dependence on time has prime importance for instance control systems. These methodologies have been well used in commercial, systems analysis, database design and information systems environments and large benefits have been claimed by their use.

The methodologies themselves are collections of a few basic techniques. All of these techniques concentrate on different aspects of the system manipulating the underlying data model of the analyzed system. These have been proposed by the experienced software developers and have been developed in parallel course of time, originating from different application fields and so have different terminology and graphical notations. For example SSADM calls the Entity Relationship model a Logical Data Structure and uses different notations, whereas the same information is presented by Entity Relationship Diagrams ERDs by other authors (e.g. DeMarco 1978 and McDermid 1990).

Ironically, the application of these software engineering concepts to scientific environments and especially for simulation tools, covering all the stages of the life-cycle, is not previously addressed in literature. The applications using one or two techniques do appear

in recent literature; for example Petridis et al 1991 and Knight and Petridis 1992 have used an entity relationship model in the context of a CFD to partition the problem and solution domains, Wilkinson and Byers 1993 also used an ER model to analyze complex engineering systems; but they did not benefit from other techniques to cover the whole SDLC.

4.1.4 CASE Tools :

Proper tools always contribute towards the product quality and enhance user productivity, the Computer Aided Software (or Systems) Engineering (CASE) tools are no exception. CASE tools are designed to support the techniques spanning all the stages of SDLC.

CASE is a generic word with a variety of implications. All tools ranging from analyst's workbench having basic facilities to draw Entity Relationship Diagrams (ERDs), Data Flow Diagrams (DFDs) and Structured Charts (SCs) etc; to code generators; Integrated Program Support Environments IPSEs; configuration management and version control systems; and further to Validation, Verification and Testing VV&T systems, all come under CASE tools. Their complexity is dependent on the facilities they provide. For example VV&T tools can generate the test data from program design and apply it for testing phase, in fact they ensure that the delivered software has been developed correctly, performs according to specifications and is suitable for its purpose. All CASE tools have three basic parts:

1. **Human Computer Interface HCI:** This is usually a Graphical User Interface (GUI) developed on some WIMP (Windows, Icon, Mouse and Pull-down menus) environment and generates WYSIWYG (What You See Is What You Get) output. The user of the tool always interacts to the tool through this HCI, and so most of the time thinks in terms of the graphical notations provided by the tool.
2. **Object meta model:** This is internal database of the tool, where all the data relating to different templates, graphical notations, consistency and integrity checking rules are stored and inferred when required. The user of the tool can only use this internal database but he cannot modify or update it.

3. **Repository:** This is where all the input by the user is stored. The tool captures sufficient information from the user about the entered objects so it can carry out the internal validation for its consistency and integrity. Here the user has full access to this database i.e. to read, write and modify. If the tool provides some import/export facility then this data could be accordingly formatted from/to the target port. The repository is also known as Data Dictionary and have similar meanings as would discussed later in section 4.3.3.

In further discussion the mention to CASE tool will refer to the ones having basic facilities to support analysis and design stages of SDLC. The main features of these tools are:

- Graphical diagramming support;
- Consistency checking within a phase according to pre-defined rules in meta model. For example it would not allow to pass data between data stores in data flow diagrams, or interlinking of two relations in entity relationship diagrams;
- Integrity checking between different phases of the life-cycle;
- Automation and ease of use are the main features, the templates for the respective diagrams automatically pop-up and the user anchors them where he desires. The tool prompts for their names and allocates a unique sequential number to each of them by itself. The data flows and the externals for a process are automatically drawn for lower level DFDs, and the user is warned if he attempts to delete any of these flows at a lower level. Also changes made in one phase, say in structure charts are automatically reflected in other diagrams on which it depended say in data flow diagrams (Fertuck 1992);
- Take away the drudgery of drawing process;
- Support the teamwork, a group of people can start a project jointly, partition and work independently, and at the end of a phase/project they can merge their work together (ASCENT2, 1993);
- Provide choice of notations for all the phases of the life-cycle, the one opted once is followed consistently throughout;

- (some of these tools) Can generate code for the modules which are described to sufficient details;
- Capture sufficient information for the user that is required for different in-built checks; and
- Provide complete documentation of the development process, comprehensible to all concerned, with least re-production costs. This can be easily modified to reciprocate any bug-fix or changed program specifications.

Apart from all these benefits, these tools have sharp learning curve, cannot be used unless one is familiar with SE concepts and techniques. Good tools with comprehensive features are expensive, dependent on operating systems and hardware, and are biased towards a specific audience e.g. Software through pictures (IDE 1992) is UNIX based and generates 'C' code; Excelerator (Intersolve 1992) and ASCENT are DOS and MS-WINDOWS based and generate COBOL and Pascal codes respectively. Excelerator does have screen and form generation facilities for prototyping and is widely used in commercial environment. ASCENT (Automated Strict Case Environment at Teesside) is an affordable, good and speedily improving tool available from an academic (University of Teesside) vendor. It supports Gane and Sarson, Yourdon, OOA, SSADM, MASCOT and PERT notations. Its version 2.0 (still in beta test) was available and has been used partially to draw Data Flow Diagrams for this work.

These tools enable their user to concentrate on the real work avoiding, the drudgery of the drawing process and consistency checking, but still they require more then the user would normally put in if he doesn't use a tool. For example, for the best results it is recommended that the user should draw all the diagrams in rough and sort out the names for all data flows and objects before hand, otherwise any wrong name associated with a data flow at a lower level of a DFD can over-write the names of corresponding data flows at higher levels. Nevertheless, these overheads are negligible as compared to the gains.

Holloway 1991, has given a detailed comparison of CASE tools, by categorising them into nine different types according to their functionality. He has compared the available tools of each type and has pointed out what features they offer. Also, he has quoted the reasons for CASE failure in past; and what steps should be taken for successful implementation and how real gains from CASE could be achieved.

4.1.5 Application of SE Techniques for Simulation Code :

Another side effect caused by a lack of disciplined approach for present day software development is, an absence of confidence in the product, which is very rightly pointed out by Thimbleby 1993. Thimbleby has quoted the excerpts from 'disclaimers' of some well-known large software products and compared them to the 'guarantees' offered by the manufacturers of other engineering products who not only take all the responsibility for any malfunction of the product but also offer some compensation in return for it.

Wiegers 1993 has quoted the benefits of increased user satisfaction, productivity, encouraging development environment and reduction in maintenance costs to 12% by the use of SE in a small software development team, and concluded from the results of 35 monitored projects. His experience indirectly points out that the use of SE can increase the confidence in the produced software.

One of the initial tasks in software development process is *abstraction* of the process i.e. to identify all the important entities and related functions which can fully simulate or model the actual process and ignore others which are not relevant. Isner 1982 mentions three types of abstractions used for the development of scientific (or FORTRAN specific) environment programs:

Control Abstraction where the attention is focused on flow of control in program. The most commonly used representation for this purpose is flow charts. Which are still widely used by scientific community programmers despite of the facts that flow charts:

- Require relatively involved information about the problem as pre-requisite, which is rarely available to start with;
- Are more formal and understood by a limited audience i.e. programming community only; and
- Portray the problem as a whole and thus are not conducive to its further partitioning.

Procedural Abstraction concentrates on the procedures performed by the system. This is the widely used conventional approach presently adopted by the community. It ultimately leads to global data sharing and distributed decisions in the produced software which renders it to an unmaintainable code.

Data Abstraction focuses on the data and its related procedures. It implies Parnas 1972 *information hiding* principle to localise the design decisions along with the data. The data abstraction results in well maintainable code for complex systems. Parnas 1972a has compared two programs for a complex operating system, one using a conventional approach and other using data abstraction and concluded in favour of the latter. Also the SE proponents strongly support the data oriented or data driven approach, as it concentrates on data. According to JSP (Jackson 1975), the structure of a program can be determined if the input/output data streams (or indirectly the underlying data structures) are known.

Isner 1982 has proposed a three stage methodology for FORTRAN programmers based on data abstraction. At the *design* stage he writes down all the needed operations and data structures informally, which are then refined at the *specification* stage. For specification he used the 'state machine' concept i.e. where a system is comprised of finite states, and each operation either modifies (or 'Operates on') or enquires (or 'Visualises') these data structures. Isner categorises these operations into O-operation and V-operation groups and writes down their formal specifications using the actual argument or variable names. In the *implementation* stage these formal specifications are coded and the related data structures and operations are encapsulated together. He opted for the state machine concept rather than other formal (rigorous mathematical) methods because it was simple and more suited to practitioners.

Colbrook and Smythe 1990 chose more formal Z type mathematical language and defined a schema for all considered operations to achieve data abstraction and implemented using FORTRAN 77. Their claim for this choice was that the state machine concept is not powerful enough to deal with real life problems. From their results it is obvious that the formal approach is complex enough to be comprehended and used by practising programmers, secondly as pointed out by Isner, it is more suitable to the environment where the software is to be automatically validated.

The present work is based on a data abstraction philosophy. Due to incapability of state machine and complexity of formal approaches, here we will use the well established SE techniques; Entity Relationship Diagrams (ERDs), Attribute Analysis (AA) and Data Flow Diagrams (DFDs); to achieve data abstraction. Instead of committing to any particular methodology, the underlying techniques would be applied in logical order for the first three stages of analysis, design and implementation (see section 4.1.2) to achieve a tangible code.

Software engineering methods are well accepted in database design, information systems and other business environments. Nevertheless, the problems in scientific and non-scientific domains are inherently different and have different performance requirements. The scientific problems are comparatively well defined, their input and nature of the expected output, are known; the relationships between input and output variables are fixed by a mathematical model. Whereas in non-scientific case, for example an information system, this relationship is very vague, even the user is not sure what output he is looking for. Many times the required output is decided by studying a series of outputs produced by the program and then analyzing the trends of some of the variables. The non-scientific programs deal with huge amount of data so efficient retrieval and storage have higher priority, whereas the scientific programs are compute intensive and the faster algorithms to reduce computational times have higher priority.

Due to the different nature of two domains, the order of application of these SE techniques for non-scientific environment in not suitable to our problem. For example the

information systems methods (McDermid 1990) start with data flow diagrams to understand the process and to communicate with the user for further elicitation of knowledge i.e. to gather more system related data to define the problem. Whereas our problem is already defined to some extent, so we will start with writing down the problem specifications (section 4.2). The order in which these SE Techniques will be applied to the present problem is shown in Figure 4.1.

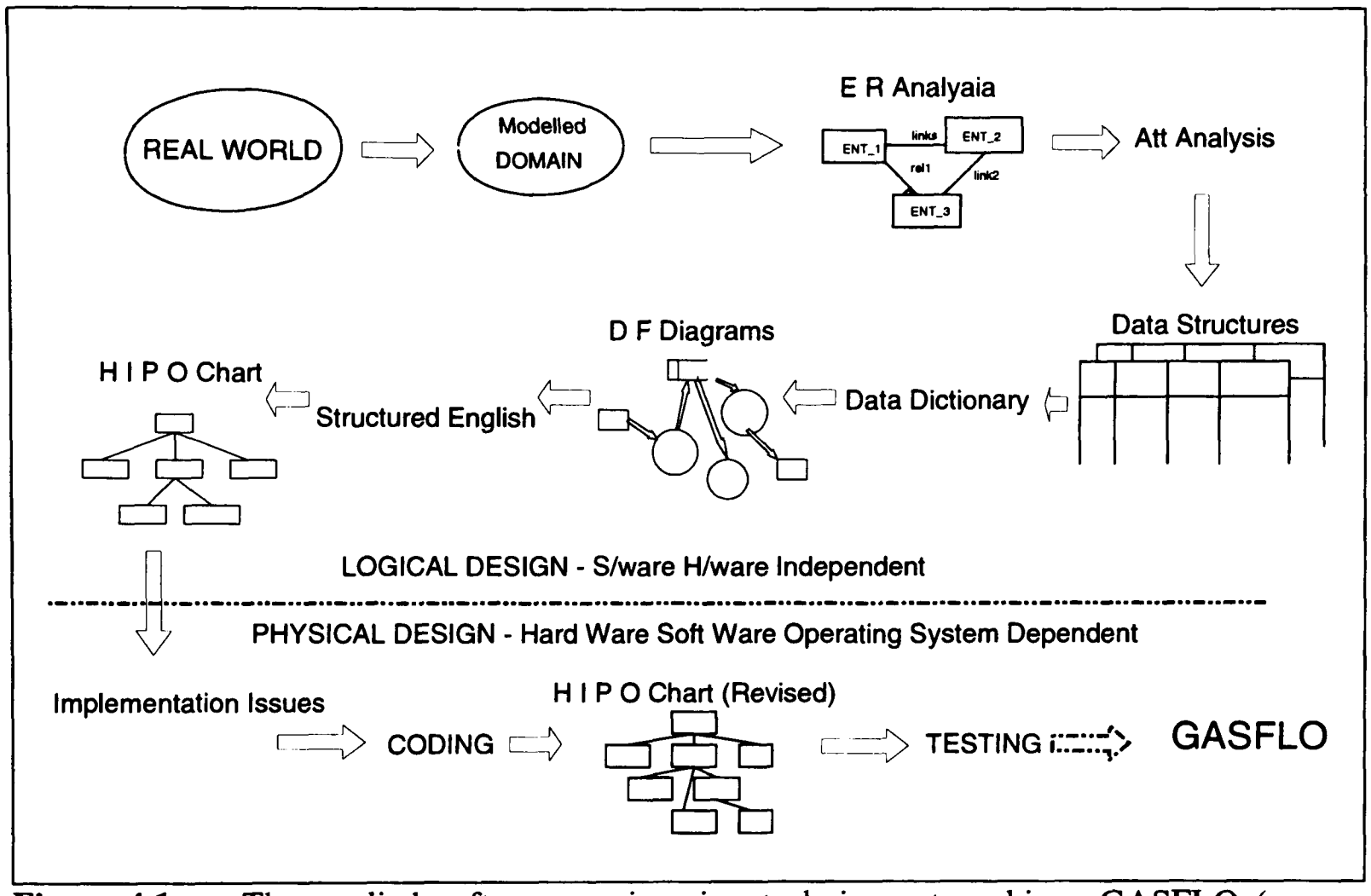

Figure 4.1 The applied software engineering techniques to achieve GASFLO (arrows show their order of application)

In section 4.3 the use of an entity relationship model and attribute analysis will be discussed to acquire the data structures for these entities. These data structures will represent problem data and will be used in data flow diagrams, which will depict the process of their transformation from input data to output data. In our case the process mainly refers to the computational procedure. The use of a data dictionary and structured English will also be discussed and the expected hierarchy of software modules will be shown as a Hierarchical Input Process Output (HIPO) chart. These steps will conclude the first two stages of the software development life cycle and it will be referred to as Logical Design, due to its

complete independence from all hardware and software constraints. Section 4.4 will consider the implementation decisions, and how the logical design is converted into physical code, this will conclude the physical design or stage three of the life-cycle. The qualities of the resulting code with reference to the maintenance costs and the extent to which the code conforms to the object oriented paradigm will be discussed in section 4.5.

4.2 Problem Specification

The problem specification includes a clear statement about the aims and constraints of the software, that is what client (the one who ultimately pays for the software) or user (who uses it) want to do with the code, what outputs they expect it to provide and what inputs they would be supplying to run the program. The hardware and software constraints are also mentioned. The problem specification is very vital as it is; the goal for the developer to achieve, wish list for client/user and a standard for the critic to compare the program with (for evaluation). In practice the problem specification keeps on changing. For development it should be agreed upon at least by the client and developer, written down and frozen, till the coded executable program appears. These should be written clearly to avoid any miss-interpretation. After initial agreed specifications, all the required changes should be fully documented.

For pellet induration network airflow distribution simulation, we are required to develop a simulation tool, GASFLO, which should determine airflow, pressure and temperature distributions in the network, for known :

- Components and their inter-connections;
- Parametric data for components i.e. friction factors, lengths and diameters for pipes; cross sectional areas and discharge coefficients for leaks; lengths, widths, heights and pellet related data for packed beds; valves' types; and controllable component parameters like fan characteristics, leaks' area variation and valves' openings. The data related to air, the process gas, is also known; and

- Given sets of boundary conditions or loadings i.e. known pressures and temperatures at source and atmospheric nodes and known flow rates at sink nodes. The temperatures of air coming out of all packed beds is also treated as fixed and known.

Only those network components which contribute towards the airflow are included in the model. The heat generation, chemical reactions and fuel sources etc are not taken into account. The simplifying assumptions for the individual components are discussed in section 2.4.1, though in the drying stage of the induration process water vapour is present but for simplicity single phase one dimensional steady state flow is assumed and modelled. Figure 2.3 may be reconsidered to identify network components which will be referred to in further discussion.

4.3 Analysis and Logical Design

The SE techniques namely entity relationship model, attribute analysis, data dictionary, data flow diagrams, Hierarchical Input Process Output (HIPO) charts and structured English are applied for data abstraction. The intended software modules are presented as modules referred to in a HIPO chart and described in structured English. This will complete the logical design of the sought GASFLOw model. These techniques contribute towards the understanding of the problem and are complementary to each other. The theoretical background and details of their application procedure are well covered in the cited references. For space limitation reasons, their description have been kept to the minium possible; these all have been discussed though some very briefly to give a flavour how these helped to reach the final (logical) design of software. The realization of this design into code will be discussed in section 4.4.

4.3.1 Entity Relationship (ER) Model :

This model was proposed by Chen 1976, for database design. He illustrated that ER model could simulate any of the then existed three data models: relational model, network model and entity set model for the database design purposes. McGee 1976 has given a

comparison of these three models and a criteria to select which would be the best for what domain. The ER model includes the advantages of all of these three models and in addition it is simple, based on set and relation theory, and has its own graphic representation. Knight 1983 has given the mathematical basis of the ER model and has supported its use for the scientific domain.

According to ER model the modelled system which is a subset of real world system as mentioned in section 2.3.1, can be represented by a set of *entities* and a set of *relationships* among these entities. An *entity* could be anything physical or non-physical, but of interest to the modelled system and have some data associated with it. A *relation or relationship* is a link between two (and or more) entities showing their inter-dependence. The number of entities linked by a relationship is called the *degree* of the relationship. In the original paper P P Chen represented relationship with 'diamond' notation which included the name of the relation and the lines linking it to the respective entities were indorsed by their participation, or cardinality ratio. This notation is still widely used in the database field (e.g. Elmasri and Naventhe 1989). DeMarco 1978 and others used crowfoot notation for relations with a mix of solid and dotted lines to represent participation. By convention the total participation of an entity to a relationship is represented by solid line and partial participation (i.e. where the relation exists without the participation of any of the instances of the respective entity), is presented by dotted line. This will be explained shortly. SSADM introduced the assignment of two names to the same relation to improve readability.

Figure 4.2 shows the Entity Relationship Diagram (ERD) for our proposed GASFLOw model. The entities shown actually represent the entity sets of corresponding component e.g. 'Pipe' as entity in Figure 4.2 represents all pipes of the network. The ends of a relationship drawn with dotted or solid lines show the optional (or partial) or mandatory (or total) participation respectively. The crowfoot end represents the cardinality ratio i.e. contribution of more than one instance of an entity to the relationship. The figure illustrates the following information:

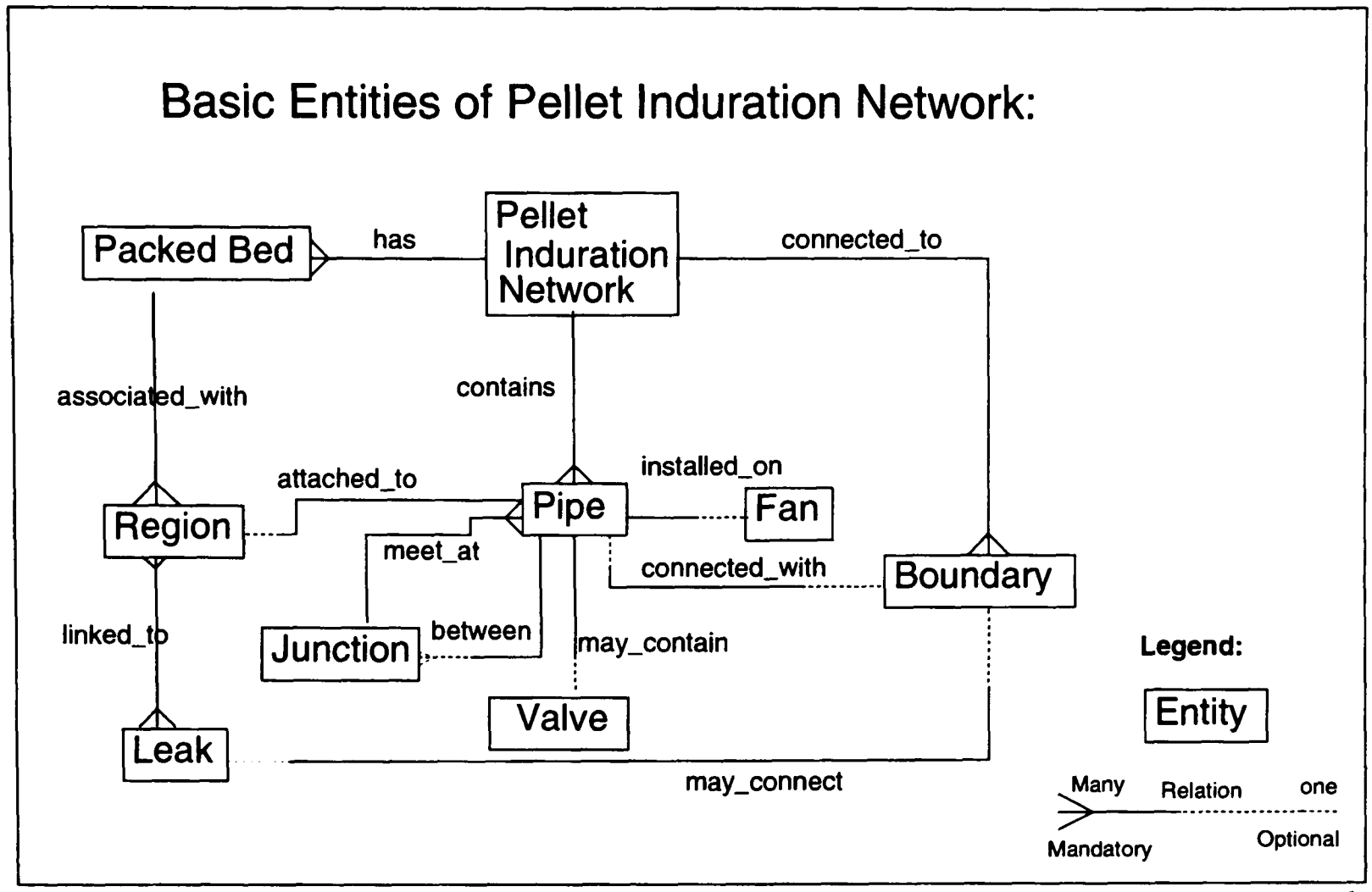

Figure 4.2 Entity relationship diagram showing basic entities of pellet induration network for GASFLOw model

- A pellet induration network *has* many packed beds, it is *connected_to* many boundaries, and *contains* many pipes.
- A packed bed is *associated_with* many regions.
- Each region is *attached_to* a pipe, but every pipe may not be *attached_to* a region. Note that the dotted line is at region end. By convention the nature of participation is determined by looking at the opposite end entity. The relationship *attached_to* is one to one, with region having total participation (looking at pipe's end) and pipe with partial participation. The same information could be conveyed more clearly in another notation by placing 1 at region end and 0/1 at pipe end, and *attached_to* could be specified as a (1:0/1) relation.
- Each fan is *installed_on* a pipe, whereas each pipe may not have an associated or *installed_on* fan.
- Many pipes *meet_at* a junction and a pipe may or may not be *between* many junctions.
- Many regions are *linked_to* a leak and many leaks are *linked_to* a region.

- A leak *may_connect* to a boundary, when it is external leak, or it may not when it is internal leak, so the relationship is dotted on boundary end. Viewing from boundary reference, it *may_connect* to a leak, in case of an atmospheric boundary, or it may not be connected to a leak for source sink boundaries, which renders the leak end also dotted. Similarly the relationship *connected_with* between pipe and boundary is also optional (dotted) on both ends. Again, it is dependent on the nature of the boundary, if it is source or sink, then it would have an associative pipe but not when it is an atmospheric boundary.

ERDs clarify the mutual dependence of the entities and provide insight for their data sharing. For example one can argue that the pellet induration system has only one boundary, that is atmosphere. If that view is adopted then it will change the relations; *connected_to* from (1:M) state to (1:1), *may_connect* from (0/1:0/1) to (M:0/1) and *connected_with* from (0/1:0/1) to (M:0/1); to not only a more complex state, but will also require that there should be only one associated pressure with the boundary which negates the physical situation as the pressures at suction ends i.e. B-1, B-2 (referred to figure 2.3) would be different than at exhaust B-4 and B-5 ends. This suggests that the entity boundary should have more than one instance, each representing the atmosphere at a point adjacent to the network locally. This single boundary would have also increased the number of loops in the network (section 3.3) and so increased computational complexity.

As mentioned in previous chapters and in sections 1.4, 2.4, and 3.3, in the context of the used solution method, the network is computed as a two level hierarchical network. It is resolved into a network of nodes and streams using graph theoretic approach (section 2.2.3). Then it is solved for the system variables at higher or coordination level to satisfy the Kirchhoff laws. These system variables are used for the computation of components at the lower level. The introduction of new abstract entities, node and stream, partition the existing entities into two separate classes which also effect the existing relationships among the components of these two classes, which are now redirected through these new entities.

Figure 4.3 shows the effect of the introduction of the two abstract entities. Now the pellet induration system is *comprised_of* many nodes and it *contains* multiple streams. The node *links* one or more streams and also a stream *links* to more than one node i.e. *links* is a many to many relationship. A node *may_be* either a boundary, a region or a junction, this is same as .exclusive.or of predicate logic. Similarly a stream *could_be* either a packed bed, a leak, or it is *made_of* multiple (one or more) pipes. The multiple pipe stream *can_include* a

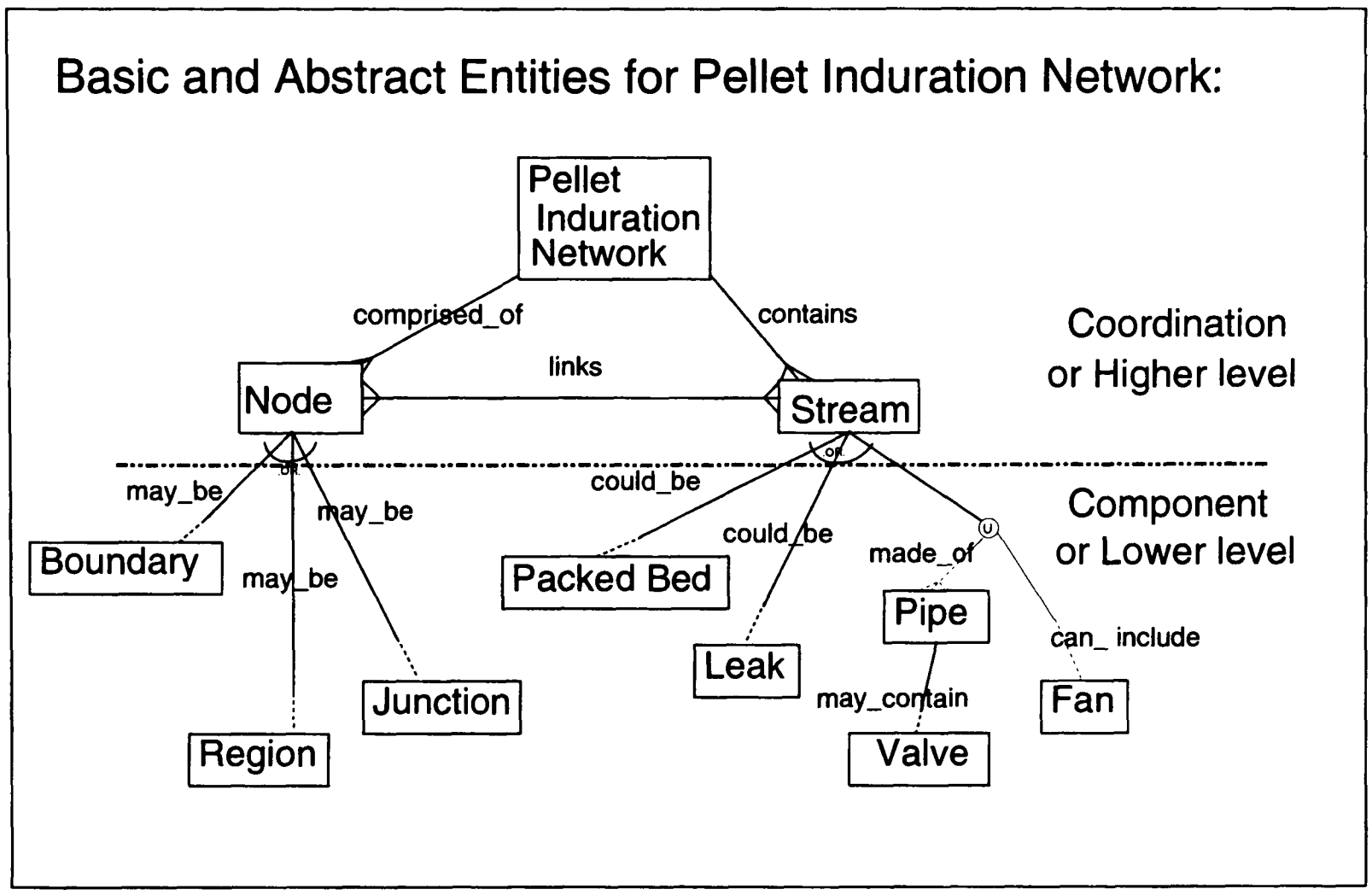

Figure 4.3 Modified ERD with two additional abstract entities

fan, the encircled 'U' correspond to the Union, usually used in Enhanced Entity Relationship (EER) notation. The *can_include* is also (1:1/0) stating the fact that there could be a multiple pipe stream which does not include a fan, whereas every fan would belong to one of the multiple pipe stream. The relationship between pipe and valve remain unchanged.

The graphical representation provides abstraction, enables an efficient and easy communication mode for all individuals related to the software (members of development

team, clients and users). It also helps in avoiding the ambiguities of textual documentation which is context sensitive and can easily be interpreted differently by different classes of the readers.

4.3.2 Attribute Analysis and Data Structures :

Attributes are the measurable properties or qualities of the entities. Some authors from database domain, using EER notation, put these assigned attributes adjacent to the respective entities on the ER Diagrams (Elmasri and Naventhe 1989), which provides complete information about the data corresponding to each of the entities, e.g. primary and secondary keys and how these can be accessed. In our case the attribute lists are comparatively long for each of the entities and so are shown separately in Figure 4.4.

The attributes of an entity for GASFLOw model are the required constants or variables associated with the computation of that entity. These are dependent on the computational scheme and mathematical model of each of the entities (section 4.4.4).

In Figure 4.4 the abstract entities node and stream contain all information relating to the connectivity of the network which is required for computation at a higher or coordination level. Each stream also contains the information about its constituent components. Some of the attributes though could be acquired from the attributes of other entities, and are repeated for computational efficiency reasons. For example the information about stream node connectivity is available in the node attribute list but it is copied to stream attribute list also to avoid extra computation. In a non-scientific environment it is emphasised that such data redundancy should be avoided for storage and maintenance requirements as it is expensive to maintain all copies up to date, but in this case it is justified for computational efficiency reasons and secondly, the amount of data also is not large.

Each entity is in fact an entity set, that is it contains many instances. So if these assigned attributes are filled in with their values then each entity will become a table (or matrix) where each row, or *tuple* will correspond to an instance of a respective entity, and a

Abstract Entities

Node: Node#, Node_Name; In_Degree, Out_Degree, {Associated_Stream#s}; Pressure, Temperature, {Stream_Flows}

Stream: Stream#, Stream_Name; UpEnd_Node#, DownEnd_Node#, No_Of_Comps, {Comp_Ids}; Status, Press_In, Temp_In, Flow, Temp_Out, Press_Out

Basic Entities

Boundary: Bdy_Id, Bdy_Name; Bdy_Nature(Source | Sink | Atmosphere); Press, Temp, Flow

Junction: Jun_Id; ; Press, Temp

Region: Reg_Id; ; Press, Temp

Fan: Fan_Id, Fan_Name; Discharge_Coef, Efficiency, Wattage; Press_In, Temp_In, Flow, Temp_Out, Press_Out

Leak: Lek_Id; Width, Height, Discharge_Coeff; Press_In, Temp_In, Flow, Temp_Out, Press_Out

Pckd Bed: Bed_Id; Length, Width, Height, Voidage, Pellet_Diameter; Press_In, Temp_In, Flow, Temp_Out, Press_Out, F_Known

Pipe: Pip_Id; Diameter, Length, Thickness, Conductivity, Fric_Factor, Effi_Fac, Vlv_Id; Press_In, Temp_In, Flow, Temp_Out, Press_Out, F_Known

Valve: Vlv_Id; Valve_Type, %_Opening, Equivalent_Length;

{ } Multiple valued attribute
(|) Attribute having one of the specified values

Figure 4.4 The attributes assigned to abstract and basic entities of pellet induration network.

column will contain the values of the associated attribute for all instances of the entity.

The very first attribute or component identification number forms the primary key for each of these entities and it must be unique. It is generated systematically. The additional component names mentioned for some of the entities, are the names commonly used by plant

engineers in their diagrams and correspondence, e.g. the Fan_Ids would be f01, f02, f03 etc, but their names in the provided diagrams and data are 3A, 3B, 1A etc, so these are retained for output and cross referencing purposes.

The hierarchical nature of the network, and introduction of node and stream as abstract entities in Figure 4.3, shows that the information available for node and stream could be shared by the entities below them. For example a node can either be a boundary, junction or region, so the In_Degree, Out_Degree and corresponding stream numbers need not be defined for these lower level entities (unless there is some special reason for it) and can be passed on to junction or region nodes at the time of computation of the respective component.

By analyzing the nature and values of the attributes assigned to each of these entities, we can conclude that each of the attribute lists can be split into three types of data:

(a) Identification data - which includes the component identity and component name if it exists;

(b) Connectivity or material or geometric property data - which is specific to the component; and

(c) System variables data - which carries the values of system variables that is pressure, temperature and flow variables relative to the respective component. The stream type components (leak, fan, packed bed and pipe) have a constant flow, so a single value of flow is associated with them alongside the temperature and pressure values at their ends. Whereas the node type components have temperature and pressure values only.

Computation wise, the data of first two types (a and b) would be read in by the program and will remain constant during the computation. It will be used for the computation of the third type (c) i.e. the system variables data, which are variable and are computed by the program. Then these attribute lists could be presented by a generic data structure comprising of these three types of data for each of the entity. This is shown in Figure 4.5.

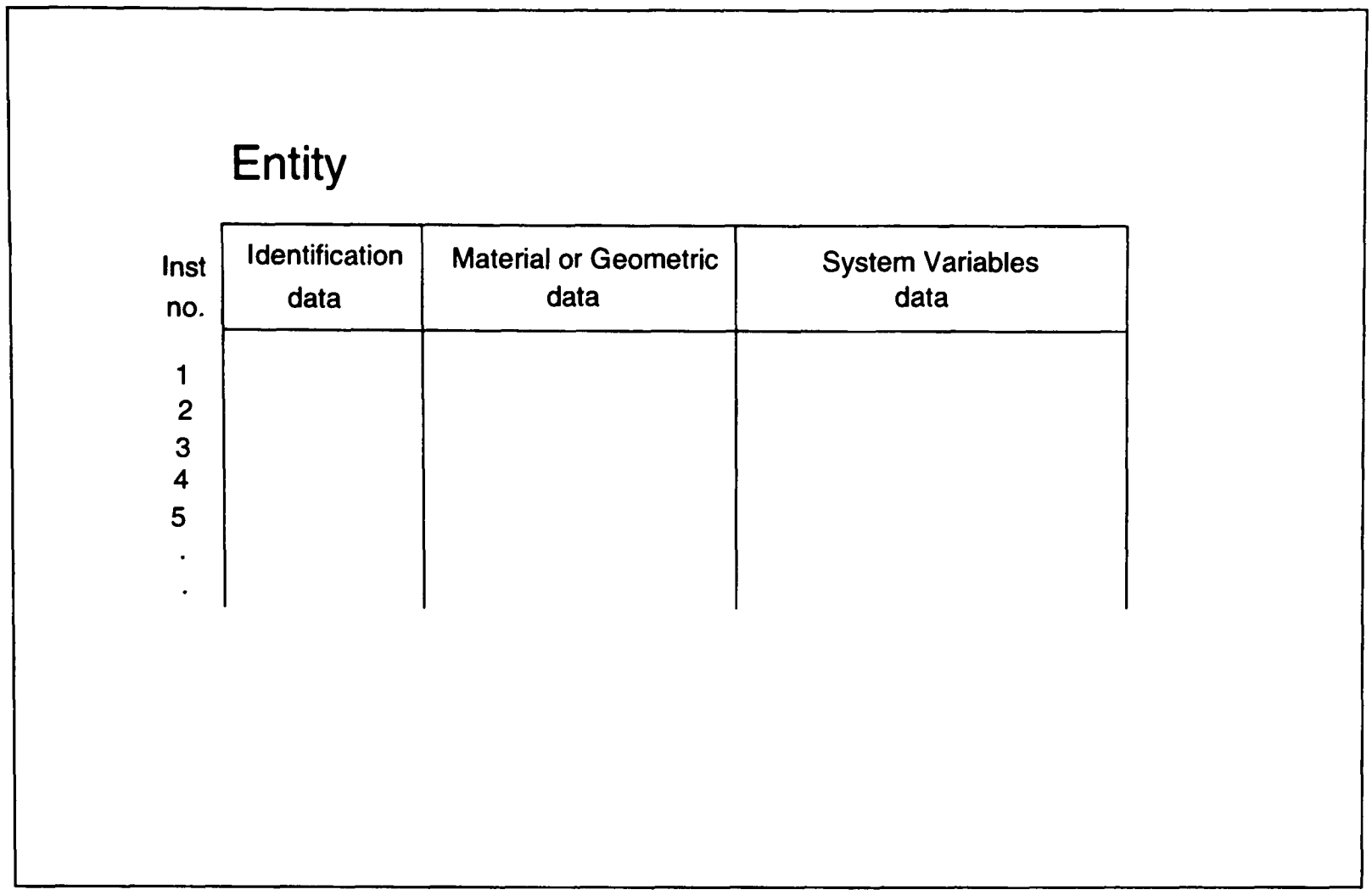

Figure 4.5 The generic data structure for an entity

Practically a program is supplied with some known raw inputs and it is required to output the desired unknowns. In terms of the above mentioned generic data structure, the input fills in the first of the two parts, whereas the program computes the equations of the components of the network and fills in its third (the variable) part. Of course, the main function of a program is to fill-in the corresponding data structures.

4.3.3 Data Dictionary :

The Data Dictionary (DD) records the information about the data, its type, nature and how it is modified during different phases of the software execution. Also known as meta data or simply data about the data. All data structures are composed of atomic, multi-valued or composite attributes. Overall storage requirements by a data structure is dependent on the data types (integer, real, double precision or character etc) of its constituent attributes and the total number of instances of each respective entity (or dimension). The information where and which part of a data structure is defined, where it is used (read only) and where it is updated

(read and written) is very important for design, function, debugging and maintenance of the software.

As mentioned in section 4.1.4, DD is a one of the main features of present day Computer Aided Software Engineering (CASE) tools. These tools help the user initially to specify the basic information about the atomic attributes, and later these could be combined into composite data structures. CASE tools provide an integrated environment for all aspects of software development depending upon the type of the tool, full use of DD is made to achieve the intended functionality. Data dictionary provides an integration layer for communication of various techniques (ERDs, DFDs and Structured charts) supported by the respective tool.

Manually maintenance of DD is cumbersome, but still possible and useful. It is developed in parallel with other techniques. It can be started after the specification of data structures, by assigning unique variable names and data types to their constituent attributes. The definition, usage and modification of each of these variable names is recorded in the form of a matrix or a table, with a row representing the modification history of a variable name (or a data structure) and column representing different processes of DFD where it is being referred. The *ij*th element of this DD matrix, would imply the state of access or modification the *j*th process has on *i*th variable name. Finally in the resulting code, the processes of DFD will transform to the respective coded modules of software so then this DD matrix would provide the information where the respective data structure has been accessed and modified. Whereas if program is fully developed using a good CASE tool, the clicking on a variable name from attribute list, can give information about all processes of DFDs and modules of structure charts where the variable was referred. This is an invaluable information tool for program maintenance purposes.

Most of the relational data base management systems use same kind of tables for data dictionary also as for the storage of ordinary data. They define separate tables; for entities, attribute and relations definitions, one for each category; and access data from these tables

using same query language as for other data, which makes the type checking and other validations of data very simple (Fertuck 1992).

4.3.4 Data Flow Diagrams :

Data Flow Diagrams (DFDs) describe the process of transformation of data from its input state to final output state. These are very versatile and used for process modelling in a variety of forms. System analysts use DFDs to model the existing systems to study their requirements, work out the system specifications and how the automation or computerization will effect its future working environment.

For our problem we use DFDs to describe the process of computation. How the raw input; i.e. known data about components, their connectivity, boundary conditions etc; is used to compute to the required output namely pressure, temperature and flow distributions in the network.

Data flow diagrams have four components; terminators, processes (or bubbles as DeMarco calls them), data stores and data flows. Each one has its own graphical representation, though it varies from author to author e.g processes are presented by soft (rounded corners) rectangles in Gane and Sarson notation, whereas DeMarco represents them with circles. We will use the Gane and Sarson notation for DFDs.

DFDs conform to the information hiding principle, the name of each process describes what it is doing, but NOT how it is being done. These are hierarchical by nature and can been exploded to further levels to describe the details of any component process of the DFD. These are flexible and depend on the user how he defines and names them, he can put as many as seven processes in a single DFD. Thus the same information would be presented differently by different users. DFDs are a good tool for refinement of ones own thoughts and an aid to communicate with the users and others. They simulate flow of data and do not show control or temporal constraints.

The boundary of DFD demarcate the range of interest at respective level. All the components inside the boundary are considered and studied whereas the components outside the boundary are 'externals'. Externals are treated as black boxes and only data flows coming from or received by them are of interest at this level. Each process is linked to its neighbours by data flows, and should have an input and output. Every process is assigned a unique number which links it to other level processes and used in other documentation for cross-referencing.

Data flows show the transfer of data among the processes and they work as a stimulus for the activation of a process and carry its response as output from the process. They link the other components of DFDs. By convention, each data flow should be given a name which will specify the input and output of the process. The data stores represent the data structures so the data flows entering or leaving a data store can have its name and can convey the desired information. Data flows always start and/or end at a process, they cannot link two data stores or a data store to a terminator or an external.

Terminators are the objects which are outside the modelled systems' domain. These are either providers of input to the system or receivers of its output. Data stores denote files or data structures. These can also be used to isolate the linked processes. For example the two non-interacting processes, one passing on its output to the other, can be delinked by storing the output of first process to a data store and then reading input of second process from that data store, just as happens for batch processes, and both can work independently.

Figure 4.6 shows the context diagram, which is top or zero level DFD. It states that 'user' feeds in raw data to the 'GASFLOw computation' process which passes on the processed output to the user. The Context diagram has a single process and no number is assigned to it by convention.

At next level this process can be expanded. Figure 4.7 shows the first level DFD, here 'user' becomes external and placed outside the boundary. It describes that 'GASFLOw

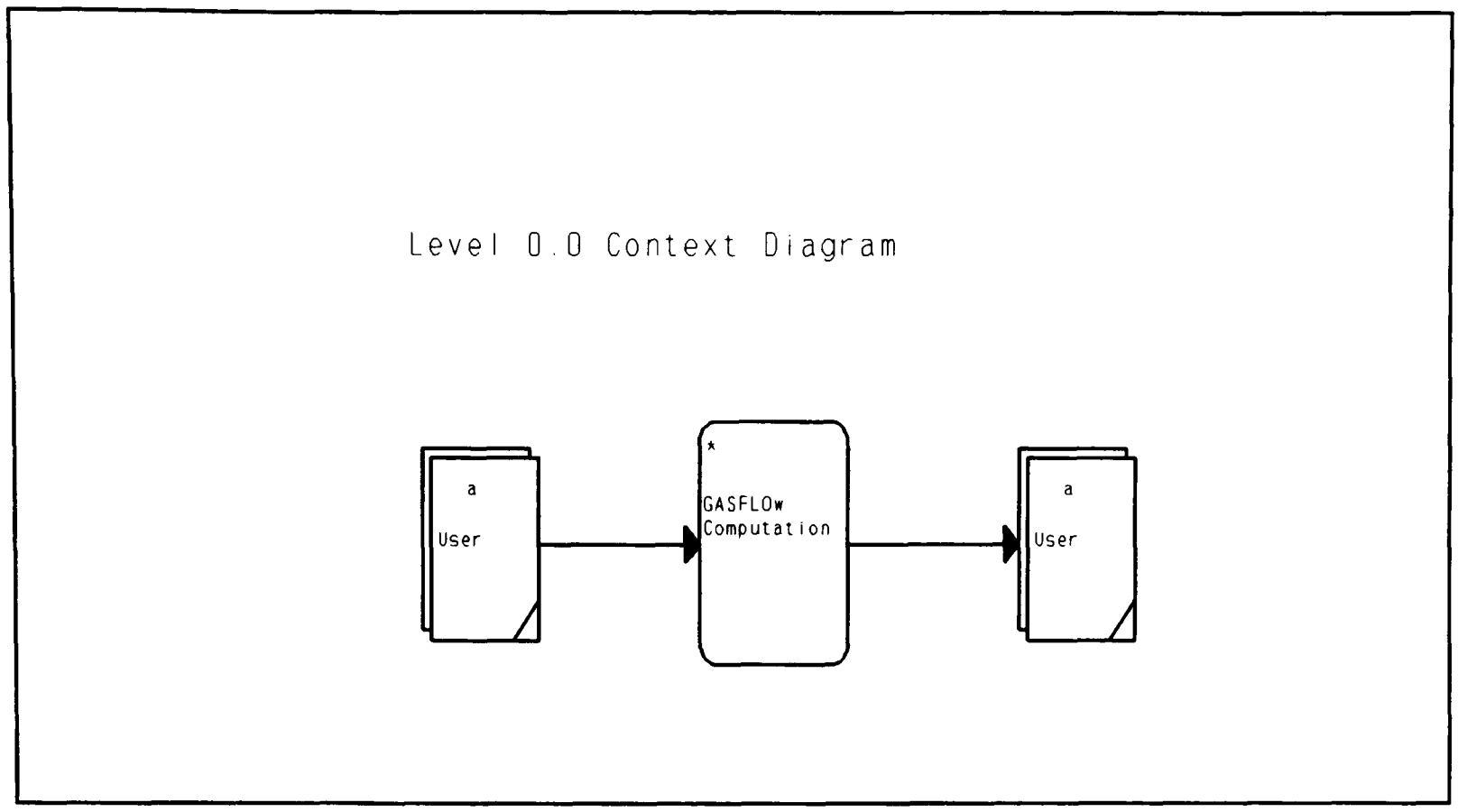

Figure 4.6 Top level (context) data flow diagram

computation' process, it is comprised of two processes; process 1 'PRePare NETwork' reads in the components connectivity data and formulates the network. The process 2 'CoMPute NETwork' computes the network using component related data from the user and network related data fed by the other process.

The externals could be terminators, data stores or processes are drawn with dotted lines showing they are not considered here and are inherited from the previous level. In Figure 4.7 the output of process 'PRePare NETwork' is stored to a data store 'Network info', instead of being passed on directly to other process. Both of the processes are linked to user to receive required inputs and pass on the computed outputs. Obviously, these data flows are specific to the respective processes. The '*' with process numbers indicates that respective process has been expanded to include further details.

At level 2.0, both of these processes will be expanded. Figure 4.8 shows the expansion of process 1 'PRePare NETwork', and describes how it formulates the network connectivity. The data flows linking the boundary of the DFD at this level are the same as the ones linking the process at the previous level. Now this DFD has five processes known as sub-processes

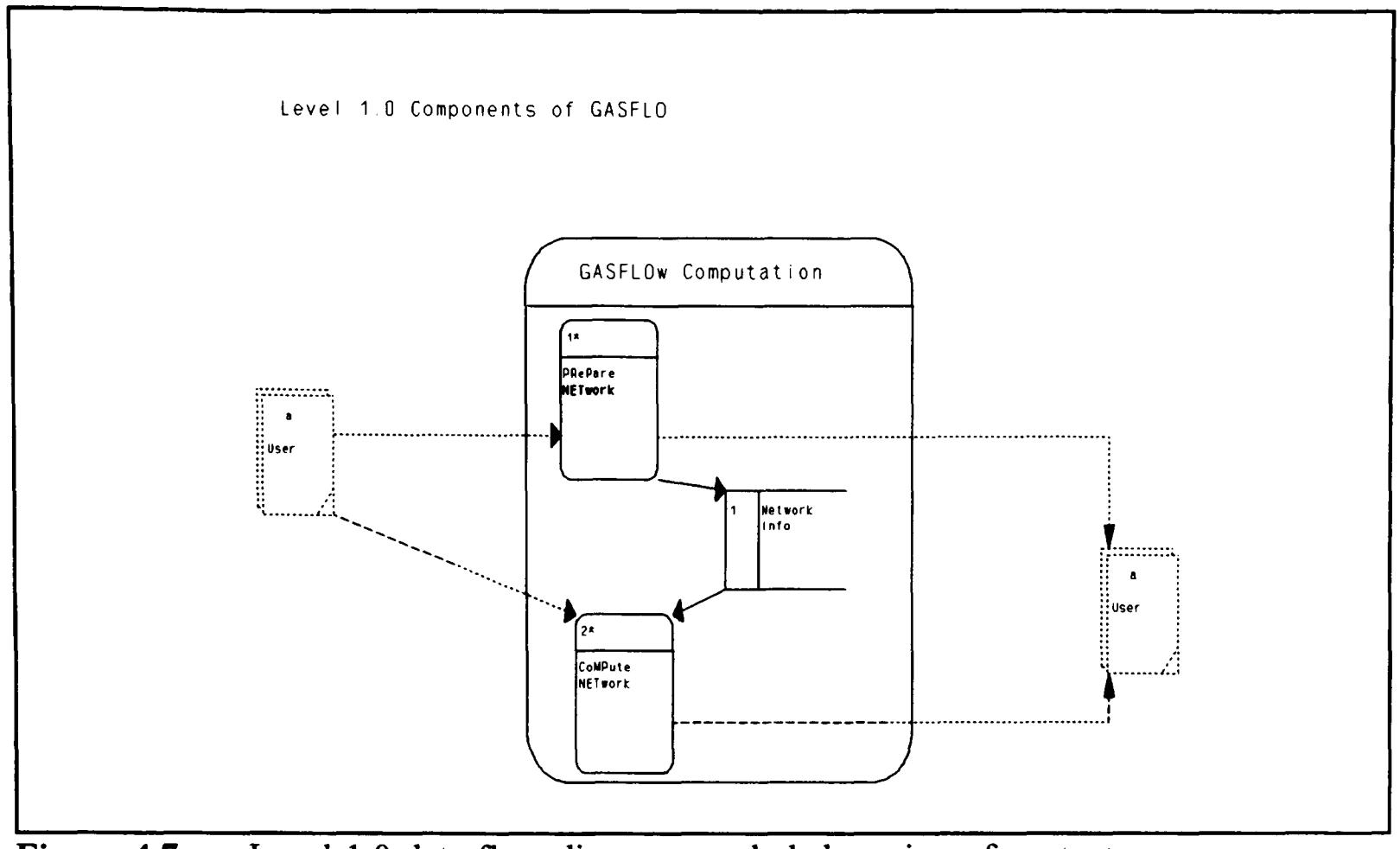

Figure 4.7 Level 1.0 data flow diagram, exploded version of context process

or children, numbered sequentially from 1.1 to 1.5 and five data stores. The assigned process numbers have no association with their computational order.

Figure 4.8 describes the process of formulation of network information and can be correlated to the algorithm (Figure 3.6) defined for this purpose in section 3.5. This DFD states that the read input is sorted out by process 1.1 'input-1' into stream and node data, and stored in 'Stream info' and 'Node info' data stores respectively. The process 1.4 'Generate Network' reads in from these stores, generates temporary network information and stores it to 'Temp Network info' data store. The bi-directional data flow to 'Temp Network info' store implies that the data is updated i.e. read and written. One of the claimed advantages of the graphical notation is that they are self-descriptive, so the working and data exchange of other processes in the figure is assumed self evident. Unfortunately, the used version of ASCENT was still in beta test and generated wrong numbers for DFD processes and sub-processes. However, correct numbers for these processes have been mentioned in the text.

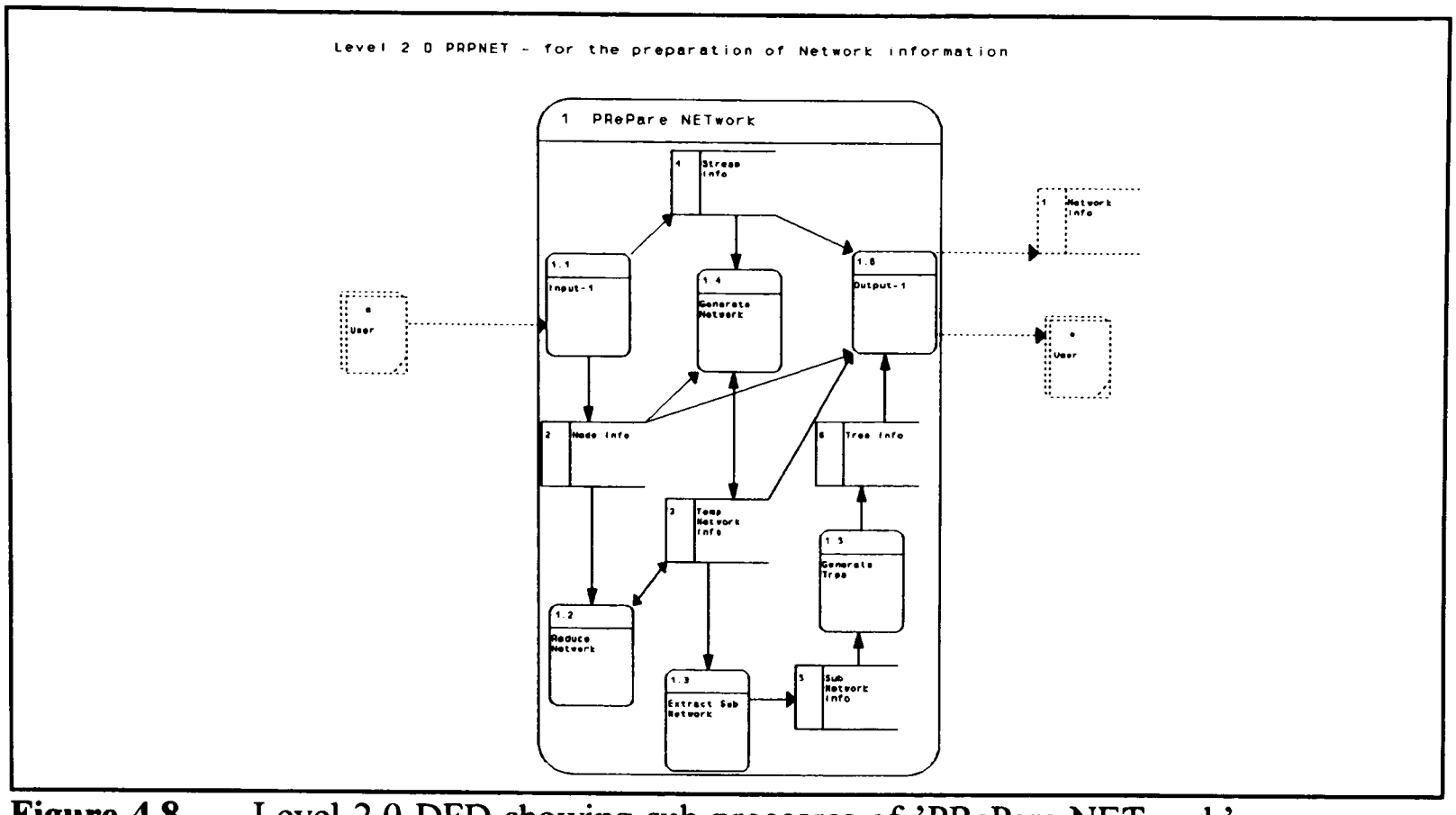

Figure 4.8 Level 2.0 DFD showing sub-processes of 'PRePare NETwork'

Refinement of process 2.0 'CoMPute NETwork' is shown in figure 4.9. The input by

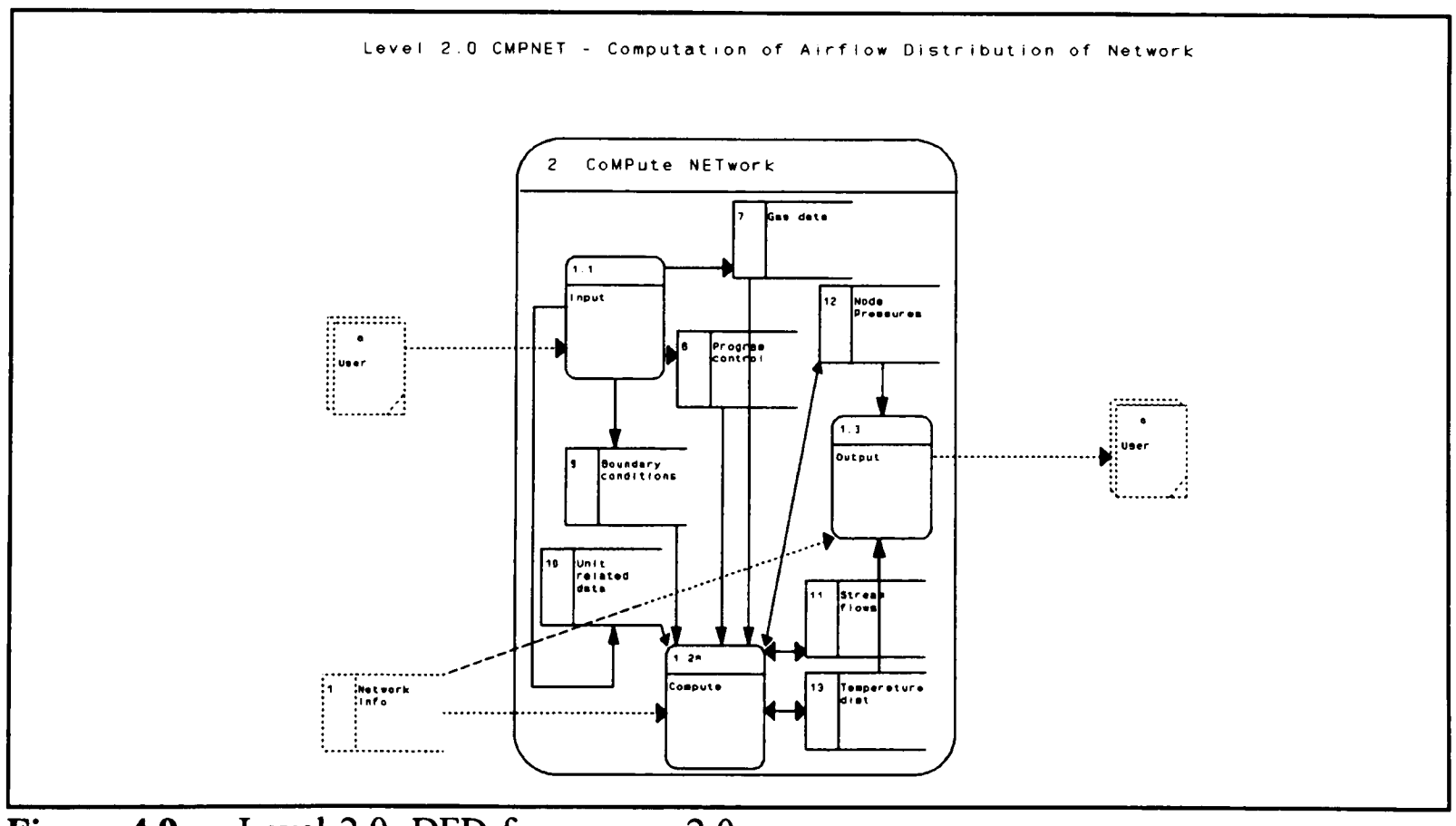

Figure 4.9 Level 2.0, DFD for process 2.0

the user is filtered out by the process 2.1 'Input' and stored to respective data stores, from where it is read by other processes. Further expansion of process 2.2 'Compute' is shown in figure 4.10, which explains how the computation is performed. Note all the data stores linked to process 2.2 at the previous level are now external to this process; and the data store 1 'Network info' provides network related information to all the three sub-processes 2.2.1, 2.2.2 and 2.2.3, as it is needed for their computation.

These processes can be expanded to any desired level. According to DeMarco 1979, this refinement should be continued to such a level where each process performs a single task i.e. it is reduced to a basic process. Whereas for a problem like ours, this refinement can be carried out to the point where the main computational steps or the purpose of the process is obvious. The functionality of these basic processes can be written down in structured English

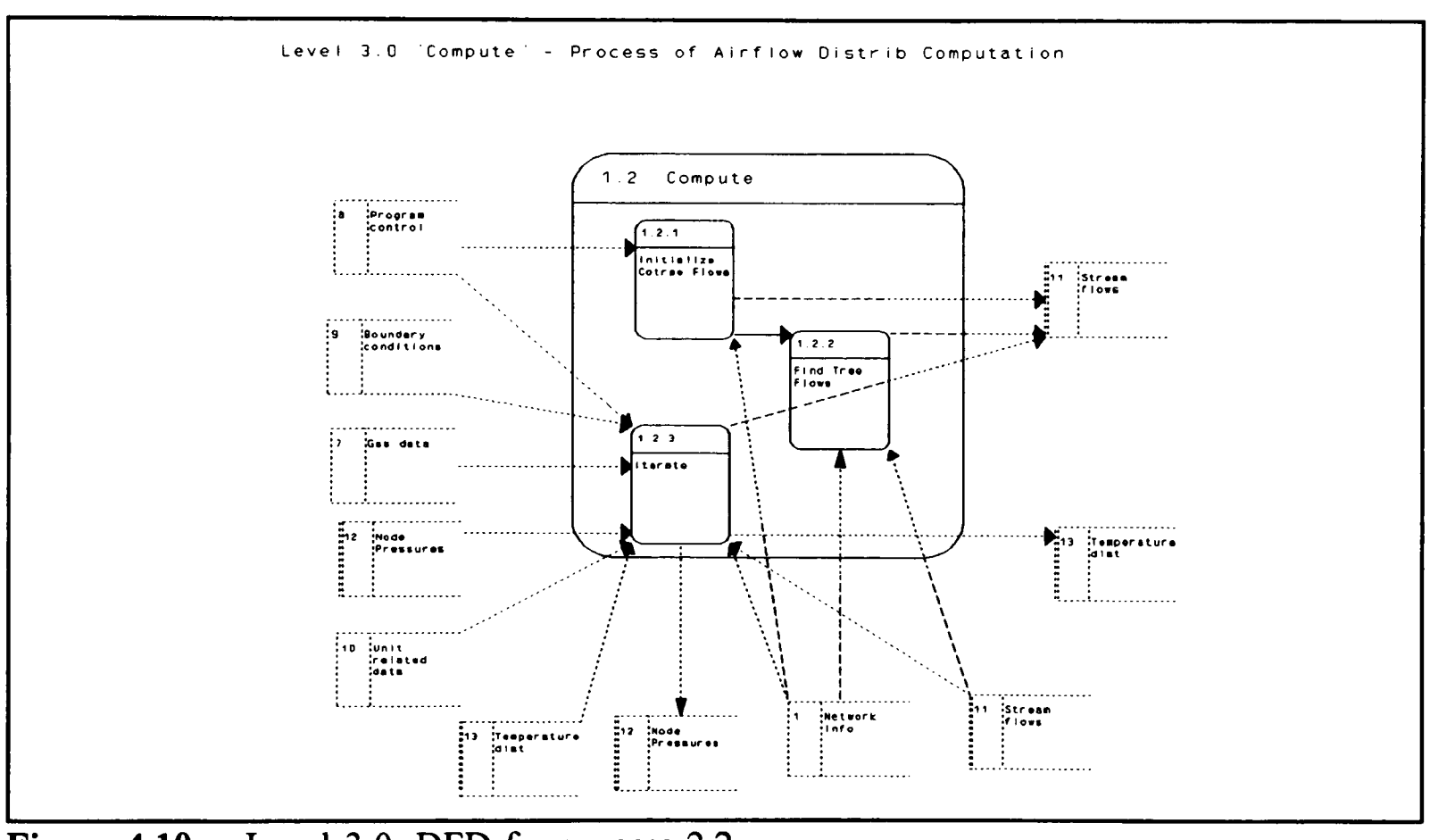

Figure 4.10 Level 3.0, DFD for process 2.2

(to be described later) and logic could be explained in terms of a decision tree or decision tables (Fertuck 1992). SSADM provides a special 'Elementary Function Description' form for

this purpose, in which along with the narration of the process, the corresponding DFD, data structure and structure chart references are also filled in.

Data flow diagrams describe the functionality of the system or process in top-down way. The Information hiding principle helps in concentrating on the components inside the boundary of DFD. These indirectly help in accomplishing the cohesion in the software modules. The insertion of data stores enables the partitioning of the process on the basis of data rather than on control or functionality which decreases overall coupling. For example in level 1.0, the output of process 1 was stored into data store 'Network Info' which delinked the two processes. This information or data has been used by many sub-processes of the process 2 but these processes have no functional coupling and so the whole process GASFLOw can be decomposed into two sub-programs PRPNET and CMPNET which can work independent of each other, the latter reads the network related information computed by the former.

In further discussion the term 'GASFLOw' though implies the overall computation of the model including PRPNET and CMPNET, but would refer in particular to CMPNET i.e. how the network airflow distributions are computed.

4.3.5 Hierarchical Input Process Output (HIPO) Charts :

Hierarchical Input Process Output (HIPO) charts (Martin and McClure 1985) show the organizational structure or the architecture of the software and are also called (with slight variations) as Structure Charts (SCs). These show how the modules comprising the software are linked, their span of control and which module controls or calls which. The relationship between, a controller module usually called the 'master' and the ones called as 'workers', is local. A worker could be a master for some other modules if it is somewhere in the middle of the hierarchy.

There is more than one notation to represent these charts. Fertuck 1992, preferred to present them horizontally to fit them on a standard sheet of paper and for ease of printing

with the convention that the modules on left side control the ones on the right. Whereas Jackson 1983, DeMarco 1978 and proponents of HIPO charts use a vertical format and claim that it is more expressive as the top level controls the lower levels.

In contrast to DFDs which are an analytical tool to describe the functionality of the modelling process, the SCs are representation of actual software and they refer to actual source code modules. These modules are represented by rectangles and the links between them describe the control or call lines. The data and control information required to realize this call are presented by the side of link. The decisions regarding the execution of these modules are taken at this time and control information included in parameter list.

Warnier-Orr notation is another horizontal presentation, which uses braces rather than straight lines for links and does not put data or control information along the call links. This notation is easier to draw and comprehend the structure of program, but conveys less information.

We use HIPO chart notation with some improvements to the standard. The flow of data and control information are shown, but their names are omitted to avoid cluttering the figure. We also draw the data structures to show their access by different modules to infer how they are accessed and filled in.

The design of structure charts, is indirectly dependent on the information available from the application of all previous techniques. The expertise of the concerned individuals also plays an important role. The relationship between processes of DFDs and modules of Structure Chart (SC) is not one-to-one, instead it could be many-to-one. The conversion of each process of a DFD into a software module could result in a large number of modules which would ultimately increase their mutual coupling and thus hinder their independence. Alternatively, the groups of DFD processes, performing similar functions and sharing data, could be identified and combined together to form a module of a SC. This may need merging of processes from different levels of DFDs and consequently it may modify the data flows

for some processes, which would be reflected back in DFDs iteratively. Moreover, the DFDs described the functionality of modelled process but these do not include any information about error checking, data validation, iteration or control. In fact these should be carried out by the software and so these additional requirements are fed into the SCs in the form of extra control information. The software may require some service routines or library modules, which could also be shown in SCs.

Finally, these modules of SC could be transformed into the modules of the software so the properties of cohesion, coupling, manageable size, information hiding, independence, explicitness and comprehension may once again be reconsidered and the proposed SC be tested against the set criteria. It is in fact an iterative process. For example to achieve the goal of **high cohesion, low coupling and moderate size**, we can try different groupings of the DFD processes. These properties of cohesion, coupling and module size are inter-related e.g. high cohesion would demand decreased module size, whereas low coupling will require an increased module size. In other words; if the module size is large, it will include more functions and will have lower cohesion thus its size should be decreased for high cohesion. However, if the module size is small then it will increase total number of modules which will increase the overall communication between the modules and hence the overall coupling of the software. To lower coupling the module size should be increased. So the balance between the two is sought. This may require few iterations but it will save a lot of labour at later stage if the same changes are done after implementation stage. Fertuck 1992 and others give guidelines for the design of SCs from the information available from previous steps.

Figure 4.11 shows the initial HIPO chart for GASFLOw model. This shows the airflow distribution computation and assumes that the data related to network connectivity has already been computed by the other part and written to 'Network info' file. The main controller module 'GASFLO' calls three modules 'INPUT', 'COMPUTE' and 'OUTPUT'.

The 'INPUT' reads in; the network related data from 'Network info' file (or data store), and other data either from files in 'BATCH' mode or from the user interactively using

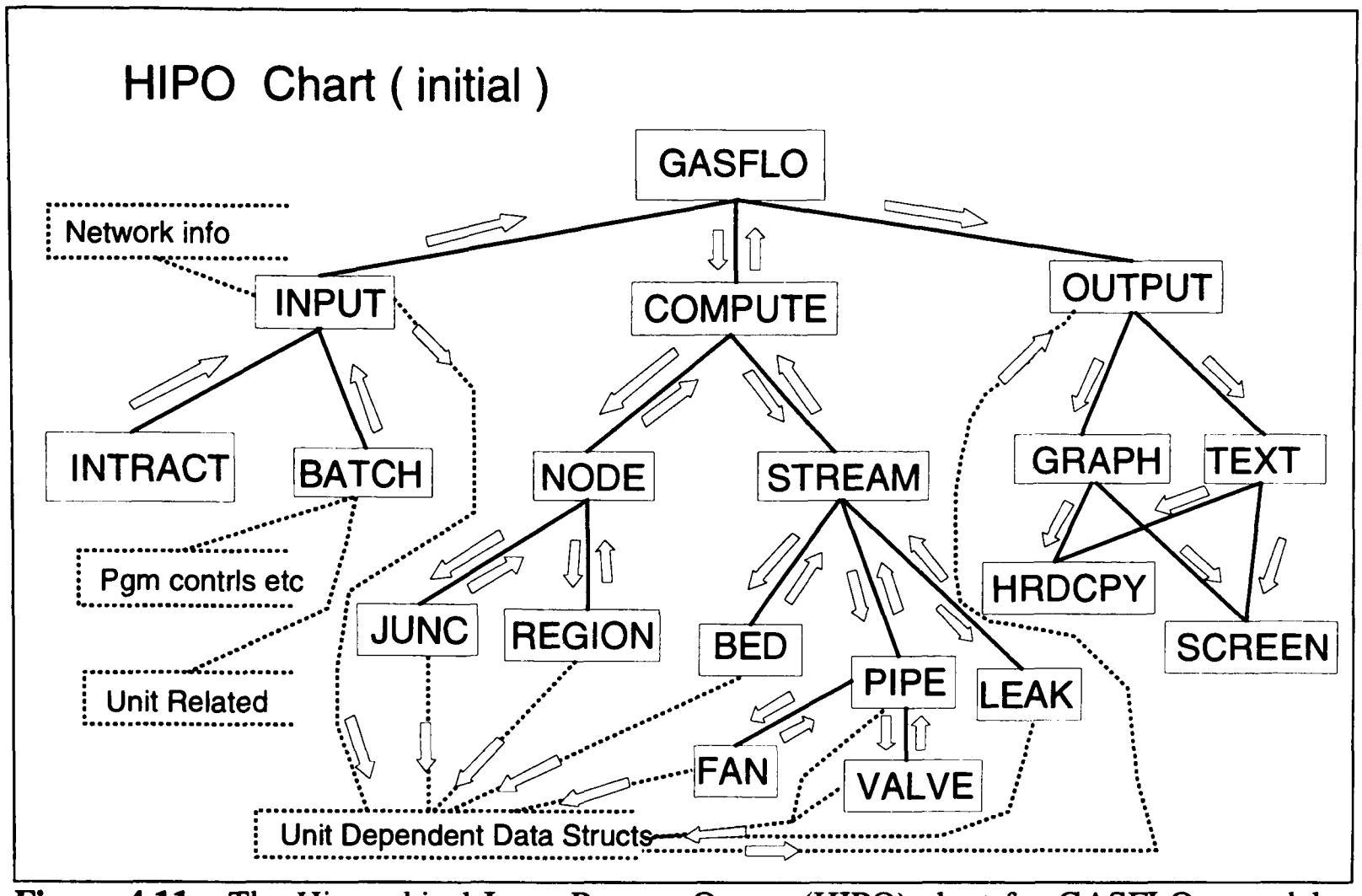

Figure 4.11 The Hierarchical Input Process Output (HIPO) chart for GASFLOw model

'INTRACT'. Reconsidering the Figure 4.5 for generic data structure, the first two parts of this data structure are constant and comprise of input supplied by the user, so 'INPUT' also fills in these two parts for each of the unit dependent data structure, which is shown as data store at the bottom. Program Controls, to control program execution and other read in data is passed onto the main module 'GASFLO'.

'GASFLO' executes the basic worker modules 'JUNC', 'REGION', 'BED', 'FAN', 'PIPE', 'VALVE' and 'LEAK' using intermediate controlling modules 'COMPUTE', 'NODE' and 'STREAM'. All these basic modules compute the airflow distributions using respective mathematical models and fill-in the third part of their data structures. A copy of these computed system variables is also passed on to the controlling modules as 'STREAM' computation needs node temperatures and pressures, and 'NODE' computation needs stream flows.

The 'OUTPUT' module receives; control and network related data from 'GASFLO' and reads in unit related data from the respective data stores, transforms it into graphical or tabular form using 'GRAPH' or 'TEXT' modules respectively as desired by the user; and finally displays it using 'SCREEN' or prints it as 'HRDCPY'. The 'HRDCPY' or 'SCREEN' should not be confused with the physical devices, these would in reality be software modules like device drivers capable of presenting output in either graphical or textual format.

The HIPO chart shows the organizational hierarchy, the module communication with their immediate bosses or workers and their control span. For complexity reasons, a module should not control more than seven modules; if others are required then another layer of controllers should be introduced.

Now the internal data structures for each of the modules or basic entities are known from ERD and Attribute Analysis; the functionality of each module is obvious from its corresponding group of DFD processes; and the data and control flows being passed up and down along the connecting links in the HIPO chart concludes the 'parameter list' or 'interface' for its respective module. These three parts of information completely specify the modules and structured English for each of the modules can be written.

4.3.6 Structured English :

Structured English is a subset of standard English with few constructs, which can describe the functionality of a module in a simple, concise and unambiguous way. The modern computer languages, e.g. Ada, provide constructs to describe the algorithms or functionality of such modules. Like other SE techniques the structured English is another aid for increased comprehension and communication among different team members. These module descriptions are written at design level using information available from data structures, DFDs and HIPO charts, usually by different people. These modules are coded by the programmers at later stage; so these descriptions ought to be in a simple, understandable language and very clear to avoid any misinterpretation.

The structured English or specification of a module is in fact the step-wise description of the method used to transform the input of a respective module to its output. While coding each of these steps requires a block of some high level computer language code, these structured English statements can serve as comment lines thus providing an overview of what is being done by the subsequent block of the code.

Jackson 1983, Pressman 1988, Sommerville 1989 and Fertuck 1992 discuss in detail the constructs used and advantages of structured English. The algorithms described in chapter 3 used similar constructs. Further in Figure 4.12 (in section 4.4.4) showing sample code, the comment lines starting with 'C-*', initially presented the structured English for respective module, later on the statements corresponding to each block were filled in. Indirectly the structured English works as an outline for the programmer.

ERDs, DSs, DFDs, HIPO charts and structured English complete the logical design of the software. Up till now, we have postponed all decisions regarding hardware, operating system or implementation language. As a result the logical design is independent of these constraints. It can be implemented for any operating system, on any hardware using any programming language. The next section on physical design and implementation will consider these issues.

4.4 Physical Design and Implementation

In this section we will be considering the practical problems and their solutions for converting the logical design into tangible executable code which satisfies the initial requirements as stated in section 4.2 and fulfils the criteria of quality software. The hardware/software dependent decisions have been delayed as much as possible to benefit from the reusability of design e.g. the change of implementation language at some later stage, would require the repetition of coding part only and could reuse the analysis and logical design which has already been completed.

4.4.1 Implementation Decisions :

Depending on the selected hardware and software any later change would require a significant amount of work and hence an increased cost, so these required changes should be well planned and documented. Some of these decisions are dictated by the client according to his available resources and preferences; e.g. what hardware he has, which operating system is being used, with what existing codes the new software has to co-exist or communicate and his preference for some computer language. The developer has to abide by these constraints and they might be included as an essential requirement, but being an expert himself, has to use his foresight for the future of the product and should warn the client of any risks associated to the choice.

For GASFLOw the following choices were made:

1. **Hardware platform - IBM PC[1] Compatible:** The main reasons being their falling prices, open standard and enormous computational power. The most important of all, the high end personal computers (486s) are available at every office or organization (Chansler and Rowe 1990) including pellet induration sites. These high end PCs can compute substantially large fluid flow networks, as reported by many authors e.g. Gomasta and Devi 1989. This provides confidence that these could also well serve the GASFLO needs.
2. **Operating system - MS-DOS[2]:** This is the operating system available on the PC platform, in fact it is the consequence of first choice. It is not only widely used but dominant operating system on PCs.
3. **Implementation language - FORTRAN 77:** FORTRAN is the most widely used programming language in scientific domain since its inception in mid 1950s (Metcalf 1985, Wilkes 1993). The initial design objectives; numeric efficiency, clear syntax,

[1] IBM PC is registered trademark of International Business Machines corporation for their Personal Computer

[2] MS-DOS is registered trademark of Microsoft Corporation for its Disk Operating System

natural and simple data structures; promoted its use and widespread acceptance among the engineers, scientists and technologists which helped the present day dominance. About 95% of the scientific programs have been written in FORTRAN, in some fields like finite element methods and structural analysis this ratio is as high as 99% (Filho and Devloo 1991). After FORTRAN 66 and FORTRAN 77 ANSI (American National Standard Institution) standards, now FORTRAN 90 standard has been approved but its compilers would take some to appear in market. This standardization process resolves portability related problems, enables users to learn just one language and help them benefit from the ever improving computer hardware. Because of compute intensive nature of scientific problems, FORTRAN still has no potential competitor (Metcalf and Reid 1990) and it will remain an ideal solution. The huge investment in the form of FORTRAN code is an other factor which ensures that this language will continue its dominance in the scientific domain.

Apart from these and many other positive points, FORTRAN is an old language and lacks explicit constructs to support the modern SE concepts of 70s and 80s. FORTRAN 77 is mostly criticised for the following shortcomings:

- Fixed syntax and format, which is inherited from its 'card punched input' age;
- Lack of block structured constructs e.g. REPEAT .. WHILE loops etc and explicit constructs for information hiding, data abstraction, inheritance and dynamic binding (which would be covered in section 4.6.3);
- Lack of dynamic memory allocation, which restricts the program to define all storage requirements at compile time;
- Lack of recursion and pointers and better input/output modes which are provided by other high level languages e.g. 'C';
- Some very powerful and widely used constructs like GOTO and COMMON (whose advantages compelled every real user of the language and was supported by texts e.g. Metcalf 1985) are regarded 'dangerous' by software engineers from maintenance and data sharing points of view respectively. For

example Fertuck 1992 assigns the worst (i.e. the highest) coupling to a module as 'Common Coupling' and denounces the use of COMMONs;

- Inability to use duplicate names for subprogram names;
- Lack of constructs for composite heterogenous data structures and user-defined abstract data types.

The above shortcomings have been addressed in the recent FORTRAN 1990 standard (F90). F90 provides the proper constructs for information hiding e.g. now abstract data types can be defined. It has improved syntax; better I/O facilities, variable names up to 31 characters, multiple statements on same line, imbedded comments and free format source code can all improve the readability. For numerical efficiency; recursion, pointers, array and matrix operations are provided which will lead to an efficient, compact and easily parallelizable code. Block structured constructs, and CASE statement has been provided. Dynamic memory allocation is possible. Complete F90 language has been discussed by Metcalf and Reid 1990, and Reid 1988 explains its usage for large problems. F90 is superset of F77 unlike other language extensions e.g. C++ is not proper superset C (Jones 1993), so all programs written using F77 will run on F90 by default. Although these F77 programs may not be as efficient as if they are written in F90 which exploits the hardware architecture of the used machine. It is hoped that with this new standard, FORTRAN will continue its superiority in numerical, scientific, engineering and technical fields (Reid 1992).

4.4.2 FORTRAN 77 - SAVE and ENTRY Constructs :

These two constructs of standard F77 have played a key role in the development of GASFLO, for data abstraction and encapsulation. FORTRAN has had a facility for the definition of local variables in subroutines and function subprograms since its conception. FORTRAN uses the 'transfer by reference' mode for the transfer of dummy arguments to subprograms which is computationally efficient. This mode provides the address of the corresponding dummy variable to a subprogram rather than its actual value. Hence the values of dummy variables are undefined when control returns to the calling program. Another problem is that on return of control to the calling program the local variables of the

subprogram are undefined or 'forgotten' (Metcalf 1985), so on re-entry to the subprogram the values of local variables established at a previous visit are not available.

SAVE statement cures this 'forgetting' problem. All the local variables are saved with their existing values and on re-entry these are available. This enables us to maintain the data structures by defining them locally in respective modules. Advantages of this statement has led many vendors to include it as a default facility for their implementations of F77 standard whereas other compilers provide it when requested by the user. It can be argued that it is good practice to use SAVE explicitly in all the modules, and specially where the re-use of the local variables is intended, so that for every compiler the SAVE option is activated. Otherwise the same software which was running smoothly on one compiler may produce unexpected results on an other compiler which does not SAVE by default.

ENTRY statement was introduced in FORTRAN 77 to re-use certain code fragments of subprograms. In function subprogram the ENTRY name should be assigned a value and single value will be returned, whereas in subroutine subprogram ENTRY has same general nature as subroutine itself, it can return any number of values to calling program. Multiple ENTRY statements could be included in a subprogram, thus providing a multiple number of points where control can enter into the subprogram. This is disliked by software engineers as well as the FORTRAN practitioners due to maintenance and code comprehension problems. Metcalf 1985 has suggested the use of COMMON (which is even worse) to share data between the modules rather to use ENTRY statement. Its intelligent use however, not only outweighs the pointed out disadvantages, but also opens the doors for object oriented programming to F77 users as demonstrated by Isner 1982, Meyer 1988, Corbin and Butler 1989 & 1990, Butler and Corbin 1989 and Colbrook and Smythe 1990.

In the following sections the use of these statements will be explained and how the shortcomings mentioned in respect of maintenance could be overcome.

4.4.3 Coding Guidelines and Testing :

Like software engineering methodologies, coding guidelines are also application, language and working environment specific. Software houses and professional developers have their home grown guidelines, which have evolved with time. These guidelines help new comers (novice programmers) to understand the coding practices and serve as a disciplined approach to be adopted by the programmers. Strict adherence to the guidelines is always beneficial, resulting uniformity in approach, better readable code and decrease of maintenance costs.

The research environment is different from the software house environment it has its own aims and objectives, though quality software is one of the common aims between the two environments. In research the individual involved, the researcher has to play all the roles as problem initiator, the client, an engineer, mathematician, numerical analyst, software designer, developer, programmer, and first-time user. So the implementation of such guidelines is not really possible due to the overheads involved. Fitzsimons and Greenough 1993 have presented a set of guidelines which could be easily followed and conform to engineering and scientific environments; but unfortunately these guidelines cover the coding or implementation stage of SDLC only.

Another approach called 'Literate Programming' is now getting established among many practitioners, this is based on Prof Knuth's notion that the 'programs should be written for *humans* not for *machines*' (Knuth 1984 and Levy 1993). This idea is supported by a variety of public domain freely available tools with which a programmer can write the programs as 'stream of consciousness' in a text file, which can later be manipulated by the provided tools to extract compiler dependent code and produce the pretty printed documentation. Majority of these tools are based on public domain T_EX software and are compiler specific. Now some other tools like CLiP are available which are word processor and compiler independent (Ammers and Kramer 1993). The present state of literate programming restricts itself to coding phase of SDLC only, as only text could be handled with these available tools. Some recently proposed tools (Shum and Cook 1993 and Lougher

and Rodden 1993), which can handle graphics and manipulate hypertext; show the potential to cover the other stages of SDLC.

In general, all texts on FORTRAN programming e.g. Metcalf 1985, and Ward and Bromhead 1989 discuss the ethics of good programming. Collins and Miller 1991 has pointed out common mistakes, mostly due to lack of understanding of the working of FORTRAN language, and suggested their remedy which in result can improve the program efficiency and overcome portability problems.

Due to space limitations we will restrict to the following principles in using the SAVE and ENTRY constructs:

- The source files corresponding to basic entities of the model contain their own unit dependent data structures which would be maintained incrementally;
- These source files encapsulate all related modules using multiple entry points;
- All source files (in principle), and especially those which are having multiple entry points, must include SAVE statement;
- Each of these basic entity source files should have at least three modules; ***INI for initial reading of input and initialization, ***COM for computation of mathematical model, and ***OPT for output; where *** refers to the three characters of respective entity name e.g. BED for packed bed, PIP for pipe etc. ***INI should be the first and subroutine name and ***OPT be the last entry name. If required more entry points could be inserted in between these two;
- Each entry and subroutine corresponds to its own RETURN. The code segments between SUBROUTINE ... RETURN and ENTRY ... RETURN are disjoint, that is each call to a respective module either through subroutine or entry executes its own code. This has been recommended by other proponents of multiple entry points and leads to safe program execution and easier maintenance;
- For clarity and readability the same variable names have been used globally for physical constants, system variables and computational controls. Adherence to F77

required maximum length of six characters, so GASCNS is R the gas constant, DENSTY is ρ the density of process gas, ACCURC is δ the local tolerance of convergence, TIN is T_{in} the temperature at the incoming end of the unit etc.

- The data about process gas and program controls is constant and read in by *INPUT* module (Figure 4.12 in section 4.4.4) and supplied to all these basic entity modules once through ***INI, by FORTRAN constraints, on return of control to the calling program the dummy variable names would become undefined and will not be available. So to resolve this situation, slightly different names have been used in dummy argument lists and then in ***INI they are copied to local variables having the same global name, e.g. gas constant comes into BEDINI subroutine in the guise of AASCNS and then it is copied to a local variable GASCNS, which is used later on for computation of the mathematical model. This copying could be done either by simple assignment statements or using an internal scratch file;
- Re-considering the data structures for respective entities, the mathematical model, computational algorithm and the present architecture of the basic entity modules, the parameter list for each entry point is decided;
- All modules of the same category i.e. ***INI, ***COM or ***OPT should have a uniform interface. For example, the BEDCOM, PIPCOM and LEKCOM all compute the system variables using their own respective mathematical models, but their interconnection with other network components is generically of the same type. Hence for any of these modules the software interface would be:

```
ENTRY UNTCOM( UNTID, TIN, PIN, FIN, POU, TOU, FLGTRE, FLGTMP)
```

Where UNT corresponds to any unit, UNITID is the identification of an instance of the unit, and TIN, PIN and FIN are temperature, pressure and flow at upstream or incoming end, and POU, TOU are pressure and temperature at the down stream or outgoing end. FLGTRE and FLGTMP are to control the respective module's computation for tree or temperature options.

4.4.4 Architecture of Unit (Basic Entity) Modules :

The similarity between data structures and the functionality of basic entities lead to quite a similar module structure and generically similar code. Figure 4.12 shows the partial listing of code related to packed bed entity. The packed bed is an important entity, its mathematical model is given in section 2.4.2, here the software module simulating packed bed is described. Other basic entity modules like fan, pipe and leak have a similar structure. This module is comprised of three parts as described previously;

BEDINI (or ***INI) is subroutine name, its main functions are:

- Import of related data which includes; network configurational data e.g. total number of beds (or the respective units) in the network; the process gas required by the mathematical model of the unit e.g. dynamic viscosity, gas constant etc; program control data e.g. input channel for reading data, maximum number of iterations and convergence criteria for local computation etc.;
- Explicit declaration of all the variables for a unit whether they are dummy or local, static or dynamic;
- Reading and validation of the bed (unit) related geometric and parametric data for each of the packed bed (i.e. for each instance of the unit), in the network either through a file or interactively from the terminal;
- Computation of constants occurring in the mathematical model of the unit and save them as local variables for later use.

The unit dependent data structures are defined as local one dimensional arrays, one array for each attribute of type LOGICAL, CHARACTER, INTEGER or REAL depending on its nature. The SAVE statement retains the values of these local variables or unit dependent data structures on RETURN from a module and provides them on re-entry. After completing all above tasks the control is returned to the calling module.

```
      SUBROUTINE BEDINI(NEWSET, IN1, MBEDS, MAXITR, AYNVSC, AASCNS,...)
C-*  Variable Declarations
C * Internal Dummy File for formal to local arguments transfer
        CHARACTER    FDUMMY*80, STRING*80
C * Real constants
        REAL     DYNVSC, GASCNS, DENSTY, PARDIA, ...
C * Bed dependent Geometric Data Structure
        REAL     BDAREA(NTBEDS), BEDENT(NTBEDS), BEDOUT(NTBEDS), ...
C * Bed dependent variables
        REAL     TBEDIN(NTBEDS), PBEDIN(NTBEDS), FBEDIN(NTBEDS), ...
           SAVE
C * For storage of formal arguments these are converted to locals
        WRITE(FDUMMY,'(2I4,5E12.5)')MBEDS,MAXITR,AYNVSC,AASCNS, ...
        READ(FDUMMY,'(2I4,5E12.5)')NBEDS,ITRTNS,DYNVSC,GASCNS, ...
C-* Read in Packed Bed geometeric or parameteric data from terminal or file
                .
                .
C-* Work out and save the Recurring constants
      DO IBED = 1, NBEDS
          CONST0(IBED) = (1.0 - VOIDAG(IBED) ) / PARDIA
          CONST1(IBED) = 1.75*GASCNS*CONST0(IBED) / (PRSATM*VOIDAG(IBED)**3)
      END DO
      RETURN
C *=============================================
      ENTRY BEDCOM ( IBED1, TIN, PIN, FIN, TOU, POU, TREBRC, TMPCMP )
C-*  Compute equations of respective bed for required variables
C    in terms of given parameters, as dictated by TREBRC and TMPCMP flags
         IF ( TMPCMP ) THEN
C-*     Temperature Distribution Computation
          ELSE
C-*     Flow and Pressure Distribution computation
         IF ( TREBRC ) THEN
C-*        Normal Forward computation, Known PIN & FIN to find POU
           G0 = FIN / BDAREA(IBED)
           CONST2 = CONST1(IBED)*BDTMPS(IBED)*BEDHYT(IBED)*G0*G0
           POU = PIN - CONST2
         ELSE
C-*        TORN stream computation i.e. known PIN and POU and to find FIN
           CONST2=ABS(POU-PIN )/CONST1(IBED)/BDTMPS(IBED)/BEDHYT(IBED)
           G0 = SQRT (CONST2)
           FIN = G0 * BDAREA (IBED)
         END IF
         END IF
C-*  Initialization of Dynamic Data Structures
100    TBEDIN(IBED) = TIN
                .
                .
       RETURN
C * ============================================
      ENTRY BEDOPT ( INTIAL, IOUT1, ITR )
C-* Output:
C * INiTIAL = .TRUE. outputs Geometric & Parametric (Static) Data Structures
C *         =.FALSE. outputs ITRth iteration values of state (Dynamic) variables
```

Figure 4.12 The partial listing of module relating to packed bed

BEDCOM (or ***COM) is the main part where computation of the mathematical model for the respective unit takes place. The parameter list includes the unit identification, coordination level (or state) variables and some flags required for the computation of respective instances of the unit. As seen in section 3.3, on the solution algorithms, the computation is split into two stages. First the pressure and flow distributions are computed and then using these at second stage the temperature distribution is computed. Further the computation of a component is dependent on its position in the network i.e. its occurrence in a tree or cotree. For tree branches the pressure at down stream end POU is found using known pressure at up stream end PIN and flow FIN, whereas for cotree branches flow FIN is found using known end pressures PIN and POU. For each of these alternatives, a corresponding set of equations are used and the flag TREBRC distinguishes between these two. The other flag TMPCMP selects between the temperature distribution or pressure and flow distribution computations.

Computation of the linear equations requires simple rewriting of the variables to be evaluated in terms of the known, whereas nonlinear equations would be computed iteratively using any suitable method, one such method viz. the *one point iterative method* was discussed in section 2.5.

Finally the coordination level variables are copied to the local dynamic part of unit dependent data structure e.g. TBEDIN(IBED) is initialized with TIN where TIN and IBED were brought into BEDCOM through the parameter list. This way complete track of variation of state variables can be kept which is useful for debugging and final detailed output of the network, because for multi-component streams the information at the interfaces of the components will not be available at the coordination level. That is why the data structures corresponding to pipe, bed, leak and fan being stream type components do not inherit the system variable attributes from the stream entity (Figure 4.3).

On return from ***COM the converged values of the desired variables are fed back to the calling program.

BEDOPT (or ***OPT) is for the output of the unit, to the channel connected by IOUT variable. The present implementation outputs for all instances of the unit. It can be easily modified for a single instance but that does not seem that useful as this part is usually called at the end of execution. The flag INITIAL controls the choice between initial i.e. static and final i.e. dynamic data structures of the respective unit.

All modules relating to basic entities are coded using the SAVE - ENTRY architecture which changed the initial HIPO chart (shown in Figure 4.11). This architecture and analysis of the coded modules and the data structures also suggested that some of the modules should be merged further due to their data coupling. These changes were carried out which resulted in the changes in calling hierarchy and is discussed in next section.

4.4.5 Revised HIPO Chart :

Figure 4.13 shows the revised HIPO chart which resulted due to the implementation of data structures using SAVE - ENTRY architecture.

The mathematical model for a junction or region (section 2.4.2) is comprised of a single equation (2.7) i.e. Kirchhoff's first or current law. According to the solution strategy discussed in section 3.3 (algorithm given in Figure 3.3), the junction or region (the same as an internal node) will be executed to find the flow in the tree branch coming in to the node in terms of all other flows connected to that node. From the data structures shown in Figure 4.4 for these two components, it is clear that the majority of the attributes are inherited from 'Node' data structure. Now if their separate modules are written then the code relating to mathematical model computation would be just duplicated, and secondly there would be more of data transfer rather than computation, which would penalise the computational efficiency of the code, so instead, the junction and region related code and data structures were maintained inside NETINI and equation computation is carried out therein by the entry point

NETCOM which computes the flow in tree. The execution of streams is also carried out at coordination level, but since these comprise multiple distinct components so each one is computed in turn as shown in Figure 4.13. The coordination level computation is included in NETCOM. Other modules related to coordination level computation are NETINI, NETFLO, NETCRC and NETOPT; these all are placed in the unit called NETWORK.FOR. Each module performs a single task.

As mentioned in section 2.4.2, valves are modelled using the equivalent length concept i.e. the resistance offered by a valve is converted into an equivalent (length of) pipe and this length is lumped to the pipe's actual length for pressure drop computation. Thus the valve computation was included in the pipe's computation. Although the entity fan has a similar relation with the pipe entity as the valve, fan has a complete mathematical model and being an important entity, is modelled as a separate entity.

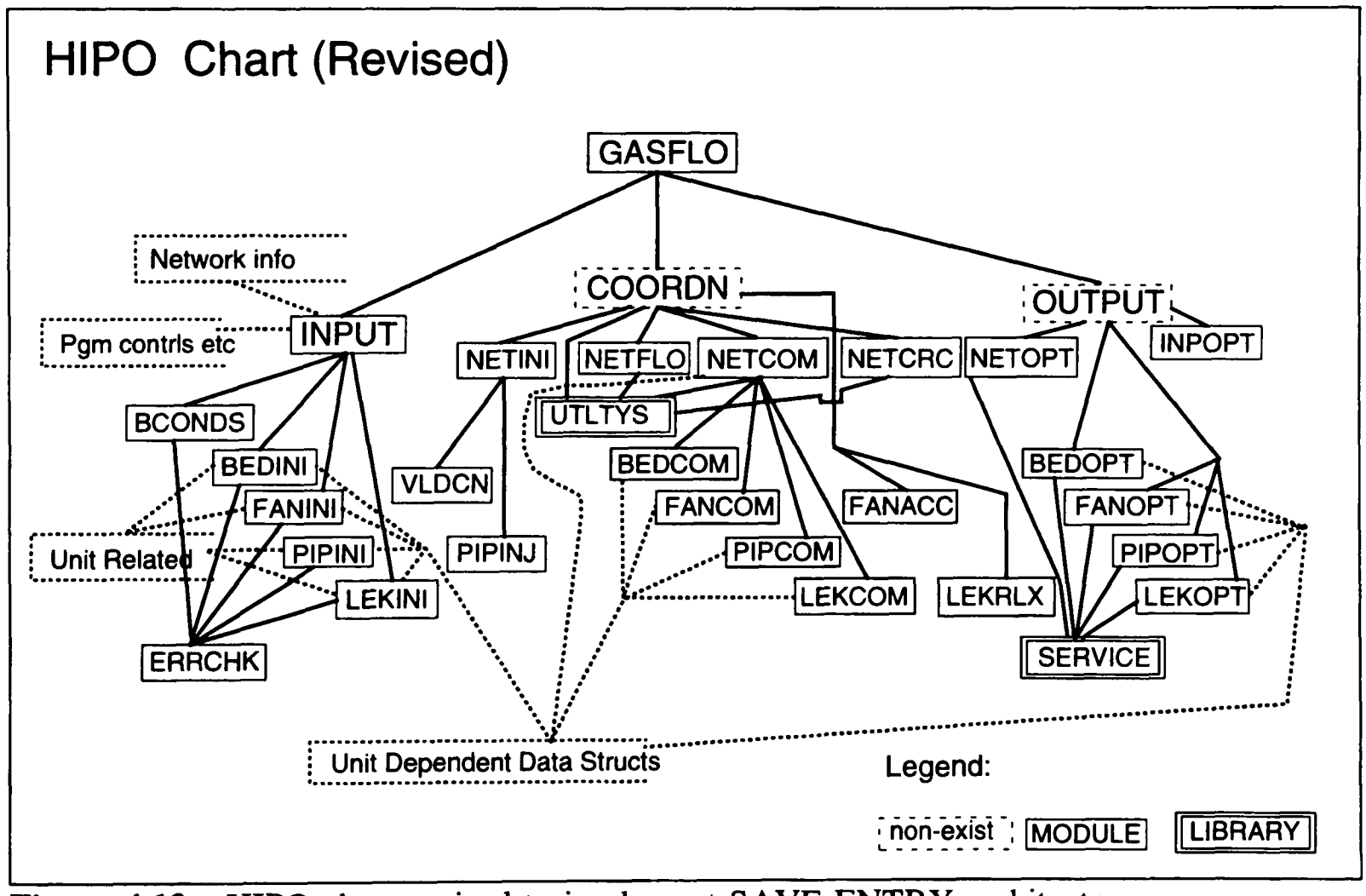

Figure 4.13 HIPO chart revised to implement SAVE-ENTRY architecture

In Figure 4.13, the data flows and control flows have not been shown for clarity reasons, but they are almost same as in Figure 4.11. Here ***INI modules read in the unit specific data and initialise the Unit Dependent Data Structures UDDSs, e.g. BEDINI reads in data for all the beds present in the network and stores it into the static part of the respective UDDSs.

NETCOM computation is controlled by a flag which selects from; tree flow, tree pressure or cotree flow distribution computation. NETFLO computes error at each of the internal (i.e. junction and regions) nodes, whereas NETCRC corrects the tree flow distribution in response to the computed error.

Modules named COORDN and OUTPUT are in fact part of the main program, here they represent the coordination level computation and output part respectively. The ***COM modules use the static part of UDDSs, compute the mathematical models of respective entities and after computation write down the dynamic part. The ***OPT modules access these completed UDDSs and output them in the format and mode, chosen by the user.

All modules whose names have the same first three characters are placed in one source file to enable them to use the same data structures. The source file NETWORK.FOR contains modules NETINI, NETFLO, NETCOM, NETCRC and NETOPT, each one has its own specific task but they share the data. ERRCHK is for the validation of data so it is used by all the ***INI modules. SERVICE is a library of utility routines to output data of multiple instances of an entity simultaneously, these are used by ***OPT modules.

F77 lacks standard functions to access operating system facilities like time, date and CPU time etc, and these are provided by the compiler vendors. Hence for portability reasons i.e. to insulate GASFLOw from an operating system, hardware and software dependencies are confined to a library UTLTYS which uses different names for these utilities and calls these compiler specific functions underneath. UTLTYS also contains some other functions for the ease of user I/O e.g. WRTYNO, and for debugging the subroutines WKSHED and WKSVAL

write to a file in worksheet format which could be directly viewed and manipulated by some worksheet program. These library functions are being called by COORDN i.e. main program.

4.4.6 Implementation and Testing :

The revised HIPO chart given in Figure 4.13 clarifies the calling structure of all modules, whereas the data structures, mathematical model, solution algorithm and program interfaces provide sufficient information to code each of these basic entity modules.

The utility library modules relating to the user I/O and data validation are written first and thoroughly tested, as these are to be reused by other modules. Their extensive re-use saves time, leads to standardisation and increases productivity. These libraries are evolutionary and incremental by nature and should be designed to be as general as possible. Any part of the code which is expected to be required by other modules also, can be designed with a general interface and written as a subprogram which after thorough testing can be placed in the library.

Earlier phases of the software development life cycle corresponding to the logical design (section 4.3) were top-down; we started from an overall general problem and refined it to finer detail and special cases. Now in the implementation stage we used bottom-up strategy.

The modules shown in HIPO chart (Figure 4.13) are written, putting all modules related to an entity; say packed bed, fan, pipe and leak; in same source file. Then each of these source file can be referred as a *unit.* For each of these units; the ***INI module is kept as first because it includes all declarations. These units are thoroughly tested. Testing includes the verification as well as validation. Verification is the process of ensuring that units perform all the functions (i.e. all corresponding modules are executed) correctly or as defined by some authors that 'is the product correctly built?' Whereas validation is to evaluate that the units perform as sought by the mathematical models that it conforms to or simulate the physics of

the process involved, and some authors define it as to ensure that 'the correct product is built'.

All mentioned software engineering references and Henderson-seller and Edwards 1993 cover the testing in detail and discuss the two major strategies for basic testing:

Clear or White Box Testing is to verify that internal algorithms are accurately implemented and the software behaves according to its specifications. It is comprised of:

Peer reviews where the other peers examine the design and specifications of the code;

Structured Walk Throughs where a group of people, involving designer, developer, programmer, user and client, are explained and guided through the different stages of SDLC. They point out different errors or mistakes which programmer has to fix later on;

Code Inspections of the produced code by other programmers and point out any miss-matches between the code and initial specifications.

Black Box Testing is to verify the module interfaces, their external communications and specification errors. It is carried out by using the executable code and comparing the outputs produced by the code with that expected. This is a widely used method although it can not pick up all the errors in the program. This strategy is strictly dependent on the test data provided. The purpose of test data is two fold, one to find where it breaks the program and to see that if it works then to see that are the provided results same as desired. Experienced programmers automate this phase by writing test programs, which should test every segment of these programs. In context of 'The errors of T_EX' for writing such test programs Prof Knuth writes:

> ' ... I generally get best results by writing a test program that no sane user would ever think of writing. My test programs are intended to break the system, to push it to its extreme limits, to pile complication on complication, in ways that the system programmer never consciously anticipated. To prepare such test data, I get into the meanest, nastiest frame of mind that I can manage, and I write the nastiest code I can think of; then I turn around and embed that in even nastier constructions that are almost obscene. The resulting test program is so crazy that I

> could not possibly explain to anybody else what it is supposed to do; nobody else would care! But such a program proves to be an admirable way to flush the bugs out of software. ... '
>
> Knuth 1989.

For GASFLOw, the first strategy of clear box testing could not be applied as it was an individual and specialised research work, so more effort was spent on black box testing.

Another FORTRAN constraint that these written units cannot be individually tested as stand alone programs. So a 'driver' to serve as master for these units was written, which called these modules and executed them. The uniform interfaces enabled the single driver to test or verify the functionality of these modules. Each unit represents an object with multiple instances (or a class with multiple objects) so the selection and execution of different instances of the unit and similar tests verified the argument lists and correct access of data structures.

For validation the test data was generated using the physics of the modelled process. For example for pipe, the down end pressure should be less than the upstream end pressure for positive flow and it would be reverse for negative flow; increase in thermal conductivity of pipe material would dissipate more heat energy and cause higher temperature drop. Similar test for fan, packed bed and leaks were designed to validate that the units correctly execute their mathematical models and simulate the physics of the process.

After unit testing, integrated testing of system was carried out, by extending the driver so that it could call more than one units in series. This is equivalent to the computation of multiple component streams. Initially the junction and region units were written separately and tested in combination with other units. Later the region and junction units were merged together as an internal node.

Some examples of integrated testing of units are shown in Afzal 1991, the initial prototype from which GASFLO was developed and is skipped from here for space limitation reasons. The test data used for integrated testing was again based on simple physical viable values, as no experimental data for any test level small networks was available. Results of full

scale pellet induration plants are discussed as case studies in chapter 5, but these were only possible after resolving all the errors during testing.

4.5 Characteristics of Implemented Code

The code was implemented based on the above software engineering principles. It resulted in two parts. The first part PRPNET prepares the network for computation by reading in the connectivity and nature of all components of the pellet induration system, connecting them into a network. It then partitions the network into a collection of acyclic trees or *a forest* and a set of cotree *or coforest* networks. This information is further transformed into linked lists and finally stored into 'network.inf', the network information file for later use. The second part, the operation part of GASFLO, is actually called CMPNET for the computation of the network. It reads in the network connectivity in linked list form from 'network.inf' file and boundary conditions and other needed data from the user either interactively or in batch mode.

The resulting code comprises of 14, F77 source files comprising of 88 modules (including utility modules) and about 6,200 lines of code, out of which about 52% are comment lines. Briefly the main properties of the code are:

- The basic entities (or components) of the network are modelled as independent units, the related data structures and operations are fully encapsulated and completely insulated;
- The data structures of a unit are completely hidden from outside and can only be accessed by defining an explicit method or operation for that purpose;
- Multiple entry points and exits enable access to unit's data structures which reduce argument lists to the minimum possible and the uniformity of these argument lists or interfaces avoids the common argument miss-match errors;
- Each of the basic unit modules performs a well defined task and operates as a worker only, so their argument lists have exactly those parameters which are required for the

functionality of respective module, this avoids *stamp* (which is due to transfer of complete data structures although only a part is needed) or *tramp* coupling which is due to transfer of some parameters which though not required by the considered module are required for the calling of another module which is being called by this module;

- Each of the units completely *abstracts* the actual physical unit in its functionality and communicates with others through a well defined interface;
- The units are extensible in structure, any number of required entry points can be defined for a respective unit when desired, though the modules ***INI, ***COM and ***OPT form an initial template;
- The units are well insulated from each other, this facilitates debugging and maintenance and errors should be easily tracked down by testing the individual unit with a driver as stated in section 4.4.5;
- Errors of a particular unit remain inside that unit and are not communicated to other units, except through system variables, which can always be checked;
- For any of these units the mathematical model as well as the solution method for computation could be replaced or modified independently. This would not effect any other module and proved to be a very useful facility for mathematical modelling studies;
- The units, especially the most used ones for computation of the mathematical model have well defined interfaces which makes their calling procedure error free;
- Has minimum storage requirements; the code does not need any substantial work space as in matrix base network solution methods (those mentioned in section 1.1) and as a result, would be able to compute comparatively large networks;
- The algorithms and solution strategies could be implemented as if visually i.e. Figure 4.14 shows the code for execution of a stream which could have multiple components, where the nature of each component is identified from its 'Comp_Id' and respective module executed;

- The modules are quite small for all the units, the component related decisions about its computation and choice of model etc are included inside the module so the code is logically less complex and easier to understand and manage as well as to maintain.

In the next section, we consider the object oriented paradigm and in the end we will evaluate the developed code against OO standards. The capabilities of the developed code, GASFLO, from the user modeller and the end-user prospective are discussed in section 5.4.

4.6 Object Oriented Paradigm

Object orientation (OO) is the latest proposed solution (or *Silver bullet* as called by Cox 1990 and Duff and Howard 1990) to 'software crisis' and hence a key to produce quality software which is accurate, maintainable, quantitatively sufficient to users increasing demands and meets the time and budget constraints. OO is a vast subject fully covered in many texts like Cox 1986, Meyer 1988 and Wimblad et al 1990 and in wide variety of research papers spanning nearly every field where computers are used. Baker et al 1993 has given a good review on the object oriented paradigm and compared the present day Object Oriented Programming Languages (OOPLs). In the following subsections different aspects of OO are briefly discussed, sources for further information are provided where necessary.

4.6.1 Some Clarifications about OO :

For the expected gains 'object orientation' has become a buzz and selling word. Every thing from operating systems to databases and programming languages is getting object oriented. Programmers are learning Object Oriented Programming Languages (OOPLs) to face the forthcoming challenge. Software houses and other organizations are looking forward to implement OO for the development of their software as soon as the Object Oriented Technology (OOT) matures. Its wide range projection has caused some confusions; such as OOT is revolutionary, it is entirely different approach and would need different people to implement etc. In fact these notions are not true and need clarification:

```
                    .
                    .
                    .

C*- For FORWaRD i.e. PRESSure Distribution computation
       DO 1800 I = 1, NTNDS
         I1  = NDPNTR (I)
         J = 1
C -      BDY/JUN/REG node execution, Skip incoming But slct+exct
C        outgoing streams
 1100    CONTINUE
           IF ( INOUT(I1).LT.0 ) THEN
C -        it is outgoing stream of the node (BDY/JUN/REG)
             ISLCT = IOSNUM ( I1 )
             IF ( TORN(ISLCT) ) GO TO 1600
C -           This part executes the selected tree stream ISLCT, and
C            modifies the Down Node TMPR and PRESS values
             PIN = PRESS ( I )
             TIN = TMPR ( I )
             FIN = FLOW ( ISLCT )
             DO 1400 K = SFIRST(ISLCT), SFIRST(ISLCT+1)-1
                CALL EXTRCT ( UNNUM, NODNAT, SCMP(K) )

                IF ( NODNAT.EQ.IPIP ) THEN
                   CALL PIPCOM ( UNNUM, TIN, PIN, FIN, TOU, POU,
     +                           FOU, FLKNOW, HLCMPT )
                   GO TO 1200
                END IF
                IF ( NODNAT.EQ.IFAN ) THEN
                   CALL FANCOM ( UNNUM, TIN, PIN, FIN, TOU, POU,
     +                           FOU, HLCMPT )
                   GO TO 1200
                END IF
                IF ( NODNAT.EQ.IBED ) THEN
                   CALL BEDCOM ( UNNUM, TIN, PIN, FIN, TOU, POU,
     +                           FLKNOW, HLCMPT )
                   FOU = FIN
                END IF
 1200            CONTINUE
                 CALL CONVRT ( OUTS, INS, .NOT.REVRSE )
 1400          CONTINUE
               PRESS( DNNODE( ISLCT ) ) = POU
               TMPR ( DNNODE( ISLCT ) ) = TOU
           END IF
 1600      CONTINUE

C -        The following part is invoked for all streams incident to the
C          node; incoming, outgoing, torn or tree branches
           I1 = NXTSTR ( I1 )
           J = J + 1
           IF ( I1.LT.0 .OR. J.GT.IDEG(I) ) GO TO 1800
           GO TO 1100

 1800  CONTINUE
                    .
```

Figure 4.14 Partial listing of module 'NETWORK.FOR' showing stream computation

- OO is an evolutionary approach, supported by software engineering concepts, just as in 1970s and 1980s were structured programming, structured design, structured analysis, CASE tools, Fourth Generation Languages (4GLs) and artificial intelligence. Each one of these helped in resolving the software crisis to some extent but not as successfully as was promised by their proponents; (Duff and Howard 1990, Brereton 1993);
- OO is a complete way of thinking (or philosophy) rather than simply coding programs using some Object Oriented Programming Language (OOPL). It is cleaner and easier to implement an Object Oriented Design (OOD) using OOPL, but it is equally possible to simulate OO in any of the existing languages like Ada (Corbin and Butler 1990 and Corbin et al 1993), C (Duff and Howard 1990), Fortran (Meyer 1988, Corbin and Butler 1989 & 1990 and Colbrook and Symth 1993), Pascal (Jacky and Kalet 1987) and SIMULA (Dahl and Nygaard 1966);
- Unlike structured methods where data and procedures are separated using either traditional process-modelling or fairly recent data-modelling approaches, here in OO paradigm data and functions are tied together;
- In OO the attention is focused on the product rather than on the process producing it (Cox 1990). It is analogous to the ideas discussed in section 2.3.2 for mathematical modelling using device-centred or unit based approach rather than the process-centred approach (Babrow 1984, DeKleer 1984 and Afzal and Cross 1992);
- The OO paradigm is another evolved phase of software engineering techniques and uses same underlying concepts; e.g. data abstraction, modularity, partitioning and conscious deferral of design decisions; to achieve black box type software modules.

4.6.2 Objectives :

The main objective of OO is to resolve the software crisis by producing high quality, low cost software and providing a system to manage the software development process. There are other secondary indirect benefits. For example OO provide mechanisms to:

- Quickly respond to user's changing needs and requests for enhanced features;
- Fully utilise the hardware as well as software resources. For example the graphics capability of present day high end personal computers and workstations and available WIMP (Window Icon Mouse and Pull-down menus) environments;
- Adapt to the changing trends in users interaction with software, e.g. as the Graphical User Interfaces (GUIs) have completely transformed the way of users thinking and interaction to scientific programs (Filho and Devloo 1991, so now for wider acceptance and even for existence such programs must have GUI;
- Deal with 'Megaprogramming' or programming of large problems like Geographical Information Systems (GISs) which are now possible by the fusion of many related technologies;
- Harness the availability of immense compute power now available due to technological advances e.g. GFlops parallel processing or connection machine technologies, which enable to simulate 'real physics' problems which previously could never be deemed off (Boghosian 1990 and Chandra et al 1992);
- Model complex engineering systems, the complexity and inter-dependence of whom components restricted the understanding about their nature previously (Wilkinson and Byers 1993).

These are few of the domains where OOT have been applied and significant gains have been reported. Many potential users are still waiting for the technology to mature and to benefit from others experiences, so that in the meanwhile all implementation problems are sorted out. Instead, it is quite clear that the success of OOT depends on its application and usage.

4.6.3 Basic Features :

The list of basic features whose conformance leads to OO varies from author to author and is also domain specific. It is generally agreed that following four basic features are necessary for a software to be object oriented:

Abstraction,

Encapsulation,
Inheritance, and
Polymorphism.

Abstraction or 'Data Abstraction' is to consider only those data attributes and functions (or procedures) for an object, which are required with reference to the desired objective of the system; and ignore other data attributes and functions which are irrelevant to the desired objective. In other words that software object in reality should simulate its actual physical counterpart in data as well as in functionality.

Encapsulation is the embedding of data attributes and functions (also known as methods) related to the object in the same software module (or source code file). This feature distinguishes OO from structured methodology where data and functions are kept separate from each other. In fact it is just an implementation of *information hiding principle* (Parnas 1972). The object data or functions are not visible or available to other objects until access is explicitly provided by the object itself.

Inheritance is an ability to inherit data attributes and functions from other objects. The objects behaving similarly are grouped together and placed in general *classes*, which can be refined further to form *sub-classes*. These sub-classes can inherit data attributes and functions from their base or parent classes. This feature enables the user to redefine or specialise the functions for the objects of sub-class and thus provides a mechanism to deal with the complexity of the system. Most of the OOPLs like SmallTalk, Objective-C, Object Pascal allow only one parent class to sub-classes, whereas some languages like C++ and Actor allow *multiple-inheritance* where a Sub-class can have more then one parent classes (Duff and Howard 1990).

Polymorphism, 'message passing' or also known as 'operation or function overloading' allows different objects to respond to the same message but each object interprets it according to its own context. For example, *Draw(Object)* could draw a 'line', 'circle',

'triangle' or a 'rectangle' depending on the value of 'Object' each one using different number of arguments and a different function, but this would be invisible to the user of Draw(). Polymorphism is implemented using two ways: *early or static binding* and *late or dynamic binding*. In static binding, the respective functions are assigned to the objects at compile time by say looking at the nature of their arguments; FORTRAN 77 generic functions are an example of static binding e.g. MAX(X) can bind to any one of the functions MAX0, AMAX1 or DMAX1, depending on data type of X whether it is integer, real or double precision. For dynamic binding the function assignment is carried out at execution or run time.

Baker et al 1993 has mentioned that some authors stress that other features, like garbage collection (i.e. to recover used memory from deleted objects), object persistence, concurrency and exception handling should also be available in perfect OOPLs.

OO programming promotes high cohesion through inheritance and polymorphism allows more general and finer grain code. Encapsulation insulates data and functions of an object from the outside world, which promotes loose coupling. These high cohesion and loose coupling were the desired characteristics for quality code mentioned in section 4.1.1.

The above mentioned features promote the reusability, independent development and testing of class libraries. In OO paradigm in fact two systems; one the actual system itself and other sub-system or library of reusable modules; are built simultaneously. OOT supports bottom up strategy for software development in contrast to structured methods' top down approach. So all the developed modules can be thoroughly tested and made bug free at time of initial development for their later reuse. This reuse leads to overall increased productivity.

4.6.4 Implementation or Migration to OOT :

From the gains reported in recent literature and possible application of OOT to every field including scientific domain confirms that it would be the future paradigm for successful development of software. It is evolving speedily though not yet mature.

Like any other technology it will take some time until tools and techniques for all phases of Software Development Life Cycle (SDLC) become established. Structured programming started in late 1960s, whereas structured design and structured analysis methodologies became established in 1970s and early 1980s respectively. OO programming the first phase of OOT started in 1980s and it is now well established but OO Design (OOD) and Analysis (OOA) are still in the infancy. Lags similar to structured paradigm are expected for OOT also (Yourdon 1990). Coad and Yourdon 1991 has proposed an OOD methodology, but in practice most of the OO developers use structured methods design techniques. For example Moses and Jackson 1991 has emphasized the use of MASCOT 3 (Modular Approach to Software Construction Operations and Test - which is a method for development of real time systems) for the development OO software.

OO is a complete philosophy, a different way or view of looking at software. It needs significant formal training. It could be easier for novice programmers to understand the OO semantics. *Nevertheless it is not impossible for experienced programmers as well.* Due to its sound theoretical base and results achieved to date, the gains are sure, but these are neither free nor immediate. The promised productivity gains for an organization, for shift to OOT may take as long as 3 years (Due 1993), till all the involved personnel get properly trained and investment in reuse resource start paying off. The understanding of basic concepts of OO is essential and it should be the primary goal, whereas learning of an OOPL is secondary (Filho and Devloo 1991). Otherwise without proper training some powerful features like inheritance could be misused and could prove to be disastrous. Lilly 1993 has quoted some examples of extracting a general class from a special class i.e. showing misuse of inheritance to override encapsulation, from the codes of some hacking OO programmers. Such semantic mistakes are impossible to overcome until and unless some higher level OO design and analysis techniques are used, which could relate and verify the relationships between different classes. A programmer aware of the OO concepts is less likely to make such mistakes. It should be realized that formal training of staff is a long-term investment.

Gibson 1990 has proposed an analysis methodology called Object Behaviour Analysis OBA for OO software development. OBA provides a conceptual model of the system and concentrates on the behaviour of involved objects. The objects are grouped into classes with respect to their behaviour. OBA supports rapid prototyping at initial analysis and design stages to elicit more knowledge from the user. The purpose of OBA is to work out the system requirements and it feeds to design stage which further feeds to coding or implementation stage (as seen in section 4.3 following conventional approach). The focus on behaviour can lead to reduction in code as the behaviour common among different objects/classes can be shared which can lead to hierarchical class structure and promote inheritance.

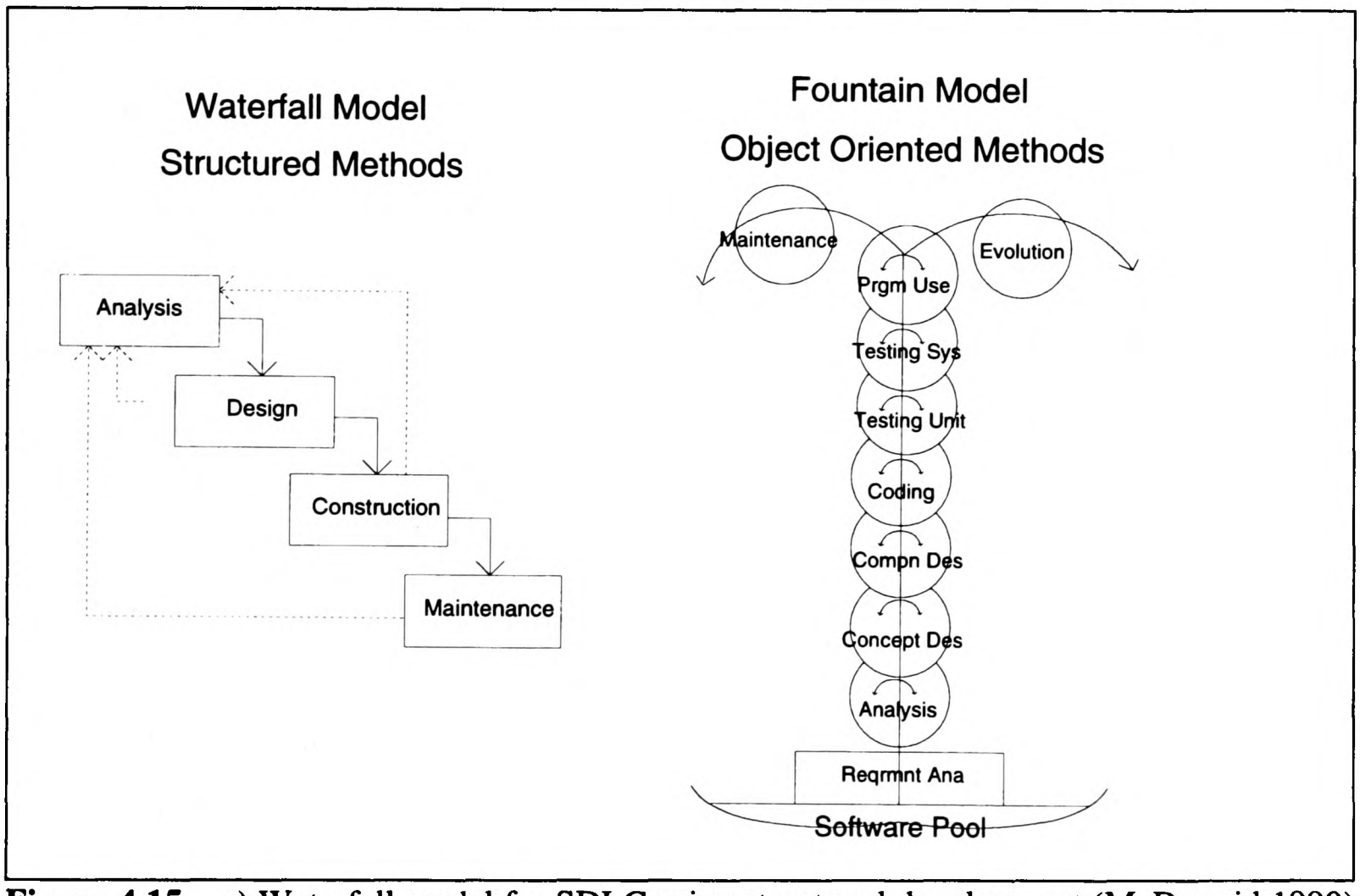

Figure 4.15 a) Waterfall model for SDLC using structured development (McDermid 1990) b) Fountain model using OO development (Henderson-sellers & Edwards 1993)

In contrast to the commonly used waterfall model (with different variants) for structured methodology, Henderson-Sellers and Edwards 1993 has proposed a *fountain model* to cover all stages of SDLC for OO environment. Fountain model reflects the iterative and recursive nature of OO software and supports generalisation and reuse of the code. Figure 4.15 compares the development process for the two models. The rectangles in waterfall and bubbles in fountain models present different stages of SDLC. In OO environment the classes are developed as an independent system, so they also follow the similar fountain model with exception of few bubbles like 'Conceptual Design' and 'System Testing' which are in fact specifically carried out at initial design and after completion. Finally evaluated classes are placed in library, from where these could be reused for future systems.

OO development is incremental by nature in contrast to structured (or traditional) development under which every new system is started from scratch without benefiting from the reuse of previously developed code.

The other important factors for migration to OOT are readiness to:

- Organizational change in working environment, as the existing hierarchy would be effected. The usual *flat* (one without rigid calling hierarchy of modules unlike structure charts) structure of OO code will also be reflected in working environment;
- Change in individuals working attitude - they will have to co-operate with each other i.e. to write for others and use others code;
- Invest for reuse resource - like any other code to develop this resource takes time and effort, or alternatively it is to be purchased from other vendors but that would be only possible for general purpose libraries;
- Adopt OO either using a new language or simulate OO in already used programming language. This is a critical decision and would require to consider the used programming language and dependence of organization on already developed code. The working life of software is in decades. If the organization has significantly invested in developed code, then some way must be sorted out for its re-use;

- Invest for OO tools and OOPLs - on which future of the organization will depend.

As mentioned earlier OO is not dependent on OOPL, and all existing procedural languages can simulate OO somehow, so one might argue that the best possible route is to use same language and simulate OO initially. It will provide good, inexpensive introduction of OO to the staff to grasp the basic concepts of OO and appreciate its gains. It will enhance their confidence in implementation of OO, encourage them to utilize the already developed code and there would be less overheads for investment in OO tools. This route has been suggested by many authors e.g. Lilly 1990, Duff and Howard 1990; and it has been adopted (for Fortran simulation) by Butler and Corbin 1989, Corbin and Butler 1989 & 1990, and Colbrook and Smythe 1990; and for Ada simulation by Corbin and Butler 1990 and Corbin et al 1993 and they have reported the above stated benefits.

From recent literature it is clear that OOT has been successfully (as only successful results are published!) applied in many fields and its benefits over the traditional methods have been mentioned. Some fields are more natural for OOT, where OOT can really excel. The mature and stable fields like the scientific domain (where methods and algorithms are stable and can form class libraries) are better candidates for OOT to exploit reuse, than the immature fields where all process and procedures are still changing like information systems (Rine 1993). Secondly, especially the fields where the structured methods failed, have also responded well to OOT, one such field is graphical user interfaces. Nowadays for the software running in WIMP environment, about 75% of its code is related to GUI whereas previously in 1980s the hardware allowed only textual display so the user interface of programs used to be a minimal fraction. OOT also suites well to CAD packages and Graphics, as they have a well defined hierarchical structure.

For best results the development team could be divided into two groups; *builders* to write the class libraries, and *users* to use these class libraries in their application programs. The two products have different goals and orientations, class libraries are written for reuse so must be general whereas the applications are specific to the clients requirements. After

some time these goals become the second nature to the programmers of respective groups. Tools for class browsing are important to both groups to inform what is already available and save them from its re-invention and hence more productive.

It should be very clear that OO techniques are separate and different then OO tools, although tools make the OO easy but still it is OO techniques which should be concentrated on and first grasped.

4.6.5 OO Programming Languages (OOPLs) :

The programming languages is a constantly evolving and improving subject. The spaghetti code problems caused by 'GOTO' were resolved by the design of new languages like Pascal, similarly better constructs like 'struct' of C appeared to handle complex data structure. To promote modularization and realize information hiding, in complex programs explicit constructs (e.g. 'interface', 'Private', and 'Public') are provided in modern languages like Modula and Ada.

Now for mega programming problems, the OO solution requires further facilities for inheritance and polymorphism. These are provided in OO programming languages. The OOPLs can be divided into two groups:

Real OOPLs

Extended OOPLs

Real OOPLs are the full fledged object oriented programming languages like SmallTalk and Eiffel. These are in fact object oriented programming environments. They provide the basic needed constructs as well as OO programming tools. For example a tool for browsing class hierarchies and methods is vital from OO point of view, it can save users as well as builders time.

These languages are designed for OO programming, and assume that the user knows the basic OO concepts. These have sharp learning curve and slightly strange syntax for conventionally experienced programmer (e.g. the message passing

convention). These languages assist programmers to write code general enough for reuse, enable them to write object oriented programs only thus helping them to make a complete shift to OO paradigm. For the same reasons their use is strongly recommended by many authors (e.g. Duff and Howard 1990, Due 1993).

Extended OOPLs are extensions of standard third generation languages like Objective-C, Object Pascal and C++. These language provide an easier and smooth route to migrate to OOT. Since the programmer is familiar with the basic language constructs of the language, so he can easily adapt to new constructs.

C++ is the most popular among this class of OOPLs. It is powerful, conforms well with the attitudes of 3GL programmers, have all the facilities for OO programming, and dominates the recent applications in various domains including scientific computation (Filho and Devloo 1991 and Rapheal and Krishnamoorthy 1993). It has been implemented on all platforms from PCs to Workstations and mainframes. Now, it is widely taught to future programmers at university level and claimed to be a step forward from C, while teaching in classroom, but in practice it is very complex language. Lilly 1993, quoted Bjarne Stroustrup, the inventor of C++ that 'growth of C++ outpaces programmers understanding how to use it'. It is still evolving and growing. The one man design and cropping features like Templates and exception handling the addition in recent version are even problematic for C++ compiler writers.

Due to its widespread use, complaints about its various weaknesses are coming to public. For example C++ is not backward compatible to C, i.e. C++ is not pure superset of C; which in 5/10 years time, after the acceptance of C++ as a standard and next revision of ANSI C standard, would result in two C type languages both different and incompatible to each other. Similarly Barber and Hay 1993 have quoted few examples of ambiguity of C++ constructs/fragments, their different interpretations at different implementations, they being compiler writers themselves, have mentioned that 'C++ is neither context free nor unambiguous'.

Now mathematics libraries are available for C++, but basically it is not a numeric programming language, the efficiencies compared to Fortran are not enough and this would be one major hurdle for programmers of CFD type compute intensive fields to shift to this language.

Overall OOPL is a fast growing subject but it is still evolving (Wilkes 1993). According to Fertuck 1992, yet there does not exist any OOP language on which 'industrial-strength' information systems could be built.

4.6.6 GASFLOw and Variants of OO :

The properties of GASFLOw described in section 4.5, show that these contains most of the properties intended from an OO code. To grade whether it is OO or not, we look for the four basic features (as mentioned in section 4.6.3).

In GASFLOw, the first two features 'data abstraction' and 'encapsulation' are fully implemented, using SAVE-ENTRY constructs.

The third feature 'inheritance' in reality was not needed in the modelled system so its implementation was not attempted. Nevertheless it can be simulated in F77 as mentioned by Corbin and Butler 1989, 1990, Colbrook and Smythe 1990, and by Collins and Miller 1991.

The last feature 'polymorphism' exists but syntactically it is not as succinct as it would be, as if some OOPL is used. The two reasons for not improving the syntax are; F77 constraint of not allowing duplicate subprogram names and computational efficiency. This constraint could easily be overcome; by defining another subroutine say CMPOBJ (OBJNME) to compute an object with generic object name OBJNME; and placing the block of the call statements PIPCOM, FANCOM, and BEDCOM to compute pipe, fan and bed respectively in that subroutine. The block shown Figure 4.14 could be replaced by a single call to CMPOBJ which in turn calls other modules PIPCOM, FANCOM, and BEDCOM according to the nature of the object. This introduction of the CMPOBJ module would though

achieve the clarity in coding but would increase another level in calling hierarchy of HIPO chart and thus would unnecessarily increase the computational load and communication. For efficiency reasons this improvement was dropped.

With the present state of the presence of these features in GASFLOw, it is left open that whether it is graded as *object oriented* or not. Lipworth et al 1991 have tagged their code as *object centred*, although they used Object Pascal (an extended OOPL) for its development but the final code contained some files which didn't exhibit object like properties and were more like traditional main programs calling other subprograms. In GASFLOw, we have the Network module which does the coordination level computation of state variables, calls other stream type object modules but itself does not conform to object like properties. Corbin et al 1993 have called their code 'Multi-Sim' as *object based.* They used Ada which being a modern language still lacks explicit constructs for 'polymorphism' and 'inheritance'. They simulated the former feature but for complexity reasons left out the latter, and hence called the code as object based rather than object oriented. According to these tagging standards GASFLOw satisfies the criteria for both of these terms whereas it satisfies the most of the properties of object oriented code. The capabilities of GASFLO are also covered in sections 4.5 and 5.4 from software developer and end-user prospective respectively.

Present day system development and maintenance tools, designed for structured programming; cross referencer, browsers, ripple effect analyzers and static analyzers; are no more helpful in OO environment. As most of these use the function name (which is meant to be unique in the whole system for procedural languages) as primary key to trace the modules, which in OO is not true, here same function can have multiple implementations. Wilde and Huitt 1992 have mentioned that the inheritance and dynamic binding, being the strengths of OOT, are problematic from maintenance point of view, they have illustrated showing examples from SmallTalk code. Lejter et al 1992 have described similar concern with reference to C++ code, which worsens the problem further by distributing codes for classes/subclasses to many files. They have also described their proposed cross referencing tool, XREFDB to deal with this problem.

It is mere coincidence that the features not implemented in GASFLOw have been reported to have some weaknesses. It increases confidence in software engineering techniques that their application assisted us in implementing only what was really needed and refrained us from simulating the not needed features artificially, just to conform to OOT.

4.7 Summary

In this chapter the application of software engineering concepts and techniques in context of scientific domain problems have been discussed. These techniques have been widely and successfully applied in non-scientific domains and significant gains reported to resolve the complexity of systems. Due the inherent natural differences of the two domains, the methodologies proposed for the nonscientific domain can not be applied. Instead, as the first attempt ever made to apply SE techniques to cover all stages of software development life cycle in scientific domain, the attention is focused on the software techniques and these are applied in order suited to the studied domain.

Abstraction is carried out by using entity relationship modelling, attribute analysis and data flow diagrams. The role of data dictionary is discussed. Initial design is carried out from the information available from previously applied techniques. Then the specifications of each of the software module could be written in structured English. This completed the analysis and logical design stages of software life cycle. The implementation related design decisions which were consciously postponed are then considered.

The implementation of encapsulation of data structures and functions using FORTRAN 77 reflected major changes in initial design or HIPO chart i.e. hierarchical structure of software module. The need to encapsulate data structures and functions was dictated by the analysis and design stages. The revised HIPO chart are drawn and modules and their specifications are accordingly amended. SAVE-ENTRY constructs of standard FORTRAN 77 enabled to encapsulate the related functions and data of objects together in their respective files, which could exist independently and behave like objects. The properties like highly

cohesive and loosely coupled modules are reflected in the resulting software, called GASFLOw.

In section 4.6 object oriented paradigm is discussed and the qualities of GASFLOw are compared with OO features. The application of software engineering enabled us to achieve the properties desired for a quality software. GASFLOw exhibits all the useful properties of object oriented technology.

Chapter 5
Model Calibration, Validation and Applications

5.1 Introduction

After development the network models are first calibrated i.e. refined or fine-tuned for the respective practical network, which is to be modelled. In this phase the available measured field data is used and the adjustable parameters are modified so that the model reproduces the scenario being analyzed. Later the calibrated model is checked by carrying out different parametric studies on the actual network and observing that the predicted results are valid. Finally, this validated model is used for the analysis and study of the actual system or process.

The use of models in the simulation of hydraulic, electricity and natural gas networks is widespread, and a significant amount of data, both measured and computed by the models, is available in public domain, which makes the validation of new models straight forward. For pellet induration systems this data is scarce, wherever available it is proprietary and being an 'enterprise resource' is hard to obtain. Also (to the best of author's knowledge) there does not exist any such model which could evaluate the airflow distribution in induration system networks and with which, the results of new models could be compared.

In this chapter the calibration, validation and application of the developed code, GASFLO, to a real life pellet induration system will be discussed. Section 5.2 will cover;

calibration, the problems on the availability of measured data, and the strategy adopted to calibrate the model. In Section 5.3 the model will be validated and parametric studies will be carried out for some components of the network and seen that it gives physically valid response. In Section 5.4 the capabilities of GASFLO i.e. what it can do, will be discussed. In Section 5.5 the interaction of GASFLO with another software tool INDSYS, which computes the heat transfer in the packed bed at microscopic level, will be explained and a combined simulation of the two tools will be carried out. In Section 5.6 a case study will be described to show a practical what-if scenario, for which the adjustable parameters will be determined, to increase the production of a pellet induration system plant by say 10%. Such a case study would simply be not possible without GASFLO or would have required a significant investment in terms of effort and time. Section 5.7 will conclude the chapter.

5.2 Model Calibration

The model calibration is in fact the refinement or fine-tuning of the model to match a specific network. It involves the adjustment of different model parameters which require estimation with the measured data and loading conditions, so that the results predicted by the model are as close as possible to the observed or measured data for the respective network.

Calibration of models simulating fluid flow networks has been around in different guises for a good time. Modelling of hydraulic networks as the oldest and mature field now has well defined and explicit calibration algorithms; whereas other fields like natural gas and mine ventilation networks use their own domain specific approaches.

In hydraulic network calibration; either pipe head loss coefficients or nodal loads, or both can be adjusted to match the predicted results by the model to the measured values of corresponding parameters or variables. Bhave 1991 has categorised these algorithms as *explicit* - which use the measured values explicitly, or *implicit* - where the measured or known

values are used implicitly by defining as many extra equations as are the known values and solving the complete set of equations simultaneously.

Ormsbee and Wood 1986 described an implicit algorithm which adjusts the pipe head loss coefficients for known values of node heads or pipe flows. This algorithm calibrates the network in one iteration and can be applied for different operating conditions. However it can consider only one loading condition at a time. Boulos and Ormsbee 1991 have extended this algorithm further to cater for multiple loading conditions. Whereas Boulos and Wood 1990 generalised the algorithm to determine any of the design (pipe diameters etc), operation (pump speeds or pressure regulating valve settings etc) or calibration parameters (friction factors or head loss coefficients for groups of pipes etc) of the network.

Bhave 1986 gives an explicit algorithm, which adjusts node loads and pipe head loss coefficients simultaneously. Herein, the adjustment factors for the node loads and head loss coefficients are evaluated using predicted and observed values. For the next iteration the node loads and head loss coefficients are multiplied by these factors. The process is carried out iteratively until the convergence of these adjustment factors (i.e. the one relating to head loss coefficients to unity and other relating to node loads to zero) is achieved.

Bhave 1991 has devoted a complete chapter to calibration of water networks and has explained the above and several other algorithms by solving a simple example using each of them.

Goldwater and Fincham 1981 included a pipe efficiency factor in the corresponding pipe (i.e. Panehandle) equation for natural gas networks, and mentioned its manipulation for network calibration. Osiadacz 1987 has discussed in detail the role of pipe efficiency factor in natural gas networks. He has explained that the predicted flows in medium and high pressure networks for non-laminar regimes, are higher than the actual observed flows, so the inclusion of an efficiency factor (≤ 1) can remedy this deviation. In fact the efficiency factor

takes into account the extra unaccounted pipe friction e.g. due to the pipe ageing and corrosion.

Among all fluid flow networks, the Mine Ventilation Networks (MVN) are closer to those of the pellet induration system's, because they also use air as process gas, consist of large area ducts and use large fans to transport the process gas in the system. D'albrand et al 1988, has described the calibration of their software tool called VENDIS (which computes flow and pressure distributions). Complications, like the presence of obstructions and variation of geometrical data e.g. lengths and cross sectional areas for some of the channels etc, make it impossible to have exact resistance values for these MVN components. So they established a database of the resistances offered by these components for various working conditions, by running VENDIS for different operating conditions. From available measured data they assign a reliability factor to each pipe flow or node pressure depending on its degree of correctness, and defined an objective function based on these reliability factors, predicted and measured values. Using another software tool called RESFIT, they select the appropriate instances of the components from the database, which provide the optimal value of the objective function and hence give the best fit to measured values.

For GASFLO calibration, none of the above algorithms is directly applicable. Instead, benefiting from the notion of a pipe efficiency factor, we will adjust the pipes' conductances or head loss coefficients such that the predicted values match to the observed or measured values. This will be done explicitly and may require several runs of GASFLO. Along with the pipe efficiency factors, the other adjustable parameters are: isothermal efficiencies for fans; and discharge coefficients for leaks and valves. The evaluation of all of these parameters would require a significant amount of measured data, which is unfortunately not initially available. So we will mainly concentrate on the available data. In Section 5.3 the calibrated model will be applied to simulate the original network and to illustrate that it produces physically valid and quantitatively correct results.

5.2.1 Problems in Data Availability :

The pellet induration systems have their own specific constraints for the measurement and provision of this needed data. To name few of them :

Hostile Environment; The temperature in some parts of the system is as high as 1300°C which inhibits the proper functionality of the involved instrumentation, and thus it is not possible to measure state variables in all parts of the system.

Lack of instrumentation due to very large size of respective components. The flows through the leaks cannot be measured due to the lack of proper flowmeters. Similarly other components are also of immense size, e.g. pipes are big ducts of few meters diameters, so special instrumentation would be needed if flow is to be measured;

Lack of mathematical models to determine the airflow and pressure distributions, which GASFLO is targeting to evaluate. If such models would have existed then they could be used for the validation of GASFLO. Secondly, their validation would have initiated the induration industry to measure this required data and encouraged the practitioners to provide this data readily;

Information is a resource for any enterprise and the pellet induration industry is no exception. For competition and commercial reasons no one would expose and provide his data to validate an evolving immature model, until some real gains are associated with it;

Provision of measured data is expensive as it costs effort and time, so unless there is some real incentive, the practitioners would not carry out this exercise. Further, the modeller needs to specify what system variables and at what location in the plant they are to be measured. These are also dependent on the model being validated/refined.

So in view of all these facts, we have to calibrate GASFLO with an incomplete data set, most of which is in the form of heuristics and some is practically measured.

5.2.2 Measurable and Available Data :

In pellet induration systems, the fan wattage can be adjusted to vary the pressures up or down the stream, which consequently changes its throughput; valves openings can be changed to control the airflow in different paths. Leak areas are almost fixed, their widths (being associated with beds) are known but their heights are not exactly known. The pipe dimensions (i.e. lengths and diameters) and bed related data are known. The volumes of junctions or regions are not known. However, GASFLO being a steady state model does not compute the overall process gas hold-up in the system or delays in its travel, so the node volumes are in fact not required. For node (junction and region) computation the stream node connectivity is needed - which is known.

The data relating to the system variables; flow, pressure and temperature in the network is rarely known. GASFLO needs only the down-stream end boundary (sink nodes) flows and up-stream boundary (source nodes) pressures for its computation. The rest of the data mentioned below is used for comparison of computed and practically measured results. The measurable data and heuristics available for pellet induration systems are as follows:

- The flows for suction and exhaust fans and also through the stack may be measured.
- The pressures in the zone regions can be measured.
- The temperatures of process gas in the packed bed are known (in fact for GASFLO these come from the other package INDSYS, which in return needs the airflow distribution from GASFLO).

Some useful heuristics about the state variable data are:

- The pipes/ducts being very large diameter and smooth, offer very small resistance to the flow and hence the pressure drop across the pipes/ducts is negligible.
- The resistance offered by leaks is also very small, so minute pressure differences in the neighbouring regions may lead to significantly large flows.

- The parallel paths having same capacity fans push nearly same flow (e.g. the pressure and flow variation in paths containing fans 1A and 1B in Figure 5.1 are similar).
- The off-gas pumped out of the system by the exhaust fans (2A and 2B in Figure 5.1) is 30% - 45% more than the on-gas being fed by the cooling zones through kiln and recuperation duct (i.e. the ambient air sucked in by suction fans 3A and 3B less the flow going out through the stack).
- Most of the pressure gain provided by the fans is utilized in overcoming the resistance offered by the packed beds.
- Although the ducts are well insulated there is still some heat lost from duct surface which may result in loss of few degrees of temperature in long ducts.

A set of known values for these state variables and above stated heuristics are used for model calibration and as a result a reference output is obtained, which will be used as base for the parametric studies later. The reference output is obtained by updating the values of unknown and adjustable parameters of the model systematically. The calibration procedure (in other words the process to obtain reference output) is described in the next subsection.

5.2.3 Strategy for GASFLO Calibration :

The model calibration is like fine-tuning the software tool to simulate a specific network. In other words it is to restrict the values of adjustable parameters to narrow ranges so that model should produce results which agree with the measured data for the respective network. In Section 2.4.2, we have seen that the computation of the respective network components requires the values of all used parameters, which must be known. In practice this is not possible; indeed, some of them cannot even be measured and will never be exactly known, thus for computation these have to be somehow assumed. In the following it is explained how sensible values can be assumed for such parameters.

We use the set of known values of the system variables (mentioned in Section 5.2.2) for a specific network as a guide and evaluate the unknown and adjustable parameters iteratively. The use of evaluated parameter values as input to GASFLO will produce an output

called *Base or Reference Output,* which will agree to the set of measured values. In subsequent parametric studies this base output will be treated as a reference. The procedure for the realization of the reference output or calibration involves the following steps:

Step-1 Initially, assume reasonable default values for the unknown parameters, e.g. all pipe calibration factors as unity, fan efficiencies less than one, and discharge coefficients for leaks and valves as 0.6 (Douglas et al 1985 have quoted this value for sharp edged orifice).

Step-2 Adjust the discharge coefficients and heights of external leaks by comparing the predicted and measured flows through suction fans. If the predicted flows are larger then to reduce them, more flow is allowed to come in from the external leaks so some of these leaks are opened or their discharge coefficients are increased. This will take a few runs of GASFLO. After obtaining a reasonable agreement (not exact) between the predicted and measured flows through fans 3A and 3B, the values of the adjusted parameters are fixed and next step is carried out.

Step-3 Evaluate the fan efficiencies and pipe calibration factors by treating measured region pressures as guide values. Start from the suction fans, set their efficiencies so that they provide the pressure at the exit ends slightly higher than the pressures at the inlet regions of the cooling zones. Then adjust the calibration factors of the ducts between the suction fans and input regions, so as to get the predicted region pressures close to their corresponding measured values. Continue the same procedure for the remaining duct and fan components, following the sequence of their occurrence in the network.

Step-4 Adjust the discharge coefficients and cross sectional areas of internal leaks to match the predicted region pressures to their measured values.

The Steps 2 to 4 could be repeated until an output agreeing closely to the known set of values is found. The updated values for unknown and adjustable parameters will remain fixed for subsequent analysis and parametric studies performed on the network.

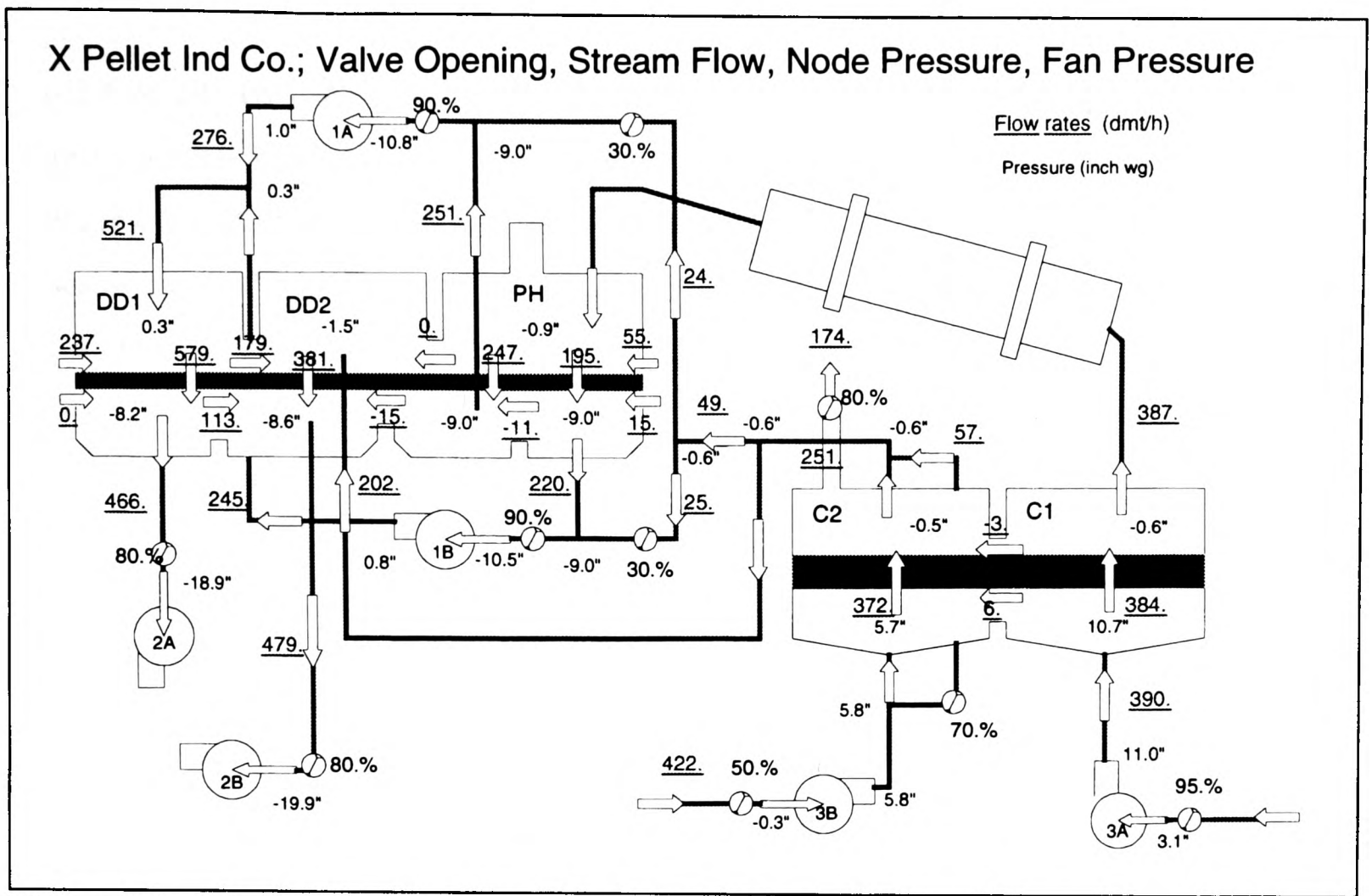

Figure 5.1 Reference output for simulated pellet induration system, showing flow and pressure distributions and valve openings

Figure 5.1 shows the reference output for a real-life pellet induration network, which was obtained using the above stated procedure. The network consist of 23 pipes, 6 beds, 10 valves, 11 leaks, 6 fans, 8 junctions, 11 regions and 9 atmospheric nodes where system either sucks in or exhausts out the process gas. When resolved into graph theoretic form (Section 2.2.3) the system reduces into a total of 37 streams and 28 nodes. The composition of the streams, and component related data for the network is given in Appendix A. The values of flows (in metric tonnes per hour of dry air) used for calibration were: 376.0, 429.0, 479.0 and 466.0 through the fans 3A, 3B, 2B and 2A respectively, and 174.0 through the stack (i.e. going out of zone C2 to the atmosphere). The measured and computed region pressures (in inches of water gauge) are given in Table 5-1.

Results given in the reference output and Table 5-1 show that these are physical, satisfy the practitioners heuristics and are in fair agreement with the measured results. The system sucks in 812.0 tph of air through suction fans, out of which 638.0 tph is recuperated

to preheat and drying stages, whereas 945.0 tph is pumped out to atmosphere through exhaust fans 2A & 2B. This shows that the air being pumped out by the fans 2A & 2B is about 33% more than what was recuperated from the cooling zone. All region pressures except for the outlet regions of drying zones (i.e. regions R09 and R11) are within the tolerance of ∓½ inch of water gauge. We will see in the later runs this is a well integrated system; the effect of variation of a parameter is not restricted to its immediate neighbourhood of respective component but is global.

Table 5-1 Comparison of Measured and Computed Region Pressures

	NODE PRESSURES in Inches of Water Gauge		
Location	Node Name	Measured	Computed
Inlet Reg - C1	R01	10.6	10.7
Outlet Reg - C1	R02	-0.7	-0.6
Inlet Reg - C2	R03	5.8	5.7
Outlet Reg - C2	R04	-0.3	-0.5
Inlet Reg - PH	R05	-0.5	-0.9
Outlet Reg_1 - PH	R06	-9.1	-9.0
Outlet Reg_2 - PH	R07	-9.1	-9.0
Inlet Reg - DD2	R08	-1.7	-1.5
Outlet Reg - DD2	R09	-10.5	-8.6
Inlet Reg - DD1	R10	-0.5	0.3
Outlet Reg - DD1	R11	-10.5	-8.2

The predicted pressure values (Table 5-1) for regions *R09* and *R11* deviate noticeably from their measured values by about 2.0 inches of water gauge. Further adjustment to match these pressures was postponed mainly due to two reasons:

(1) Model behaved smoothly and simulated all network components realistically. Both of these are outlet regions, whereas for corresponding inlet regions the predicted and measured pressure values matched well. The inlet and outlet regions are separated only by the packed bed whose mathematical model i.e. Erguns equation (2.19), is well tested and widely used in industry (see Section 2.4.2), it does not contain any

adjustable parameter. In fact there doesn't exist any adjustable parameter available to match them to the corresponding values. Secondly the same bed model has worked well for other zones, so its modification is not logical.

(2) Following the advice given by Bhave 1991, in the context of hydraulic network calibration, the resolution of such discrepancies needs the verification of measured data and examination of the modelled and actual network connectivities. It is possible there could be some phenomena or component influencing the system but not included in the modelled network.

Hence we will regard the predicted values as exact, and the outputs of all runs of the next section for GASFLO validation will be compared to this output, unless it is mentioned otherwise.

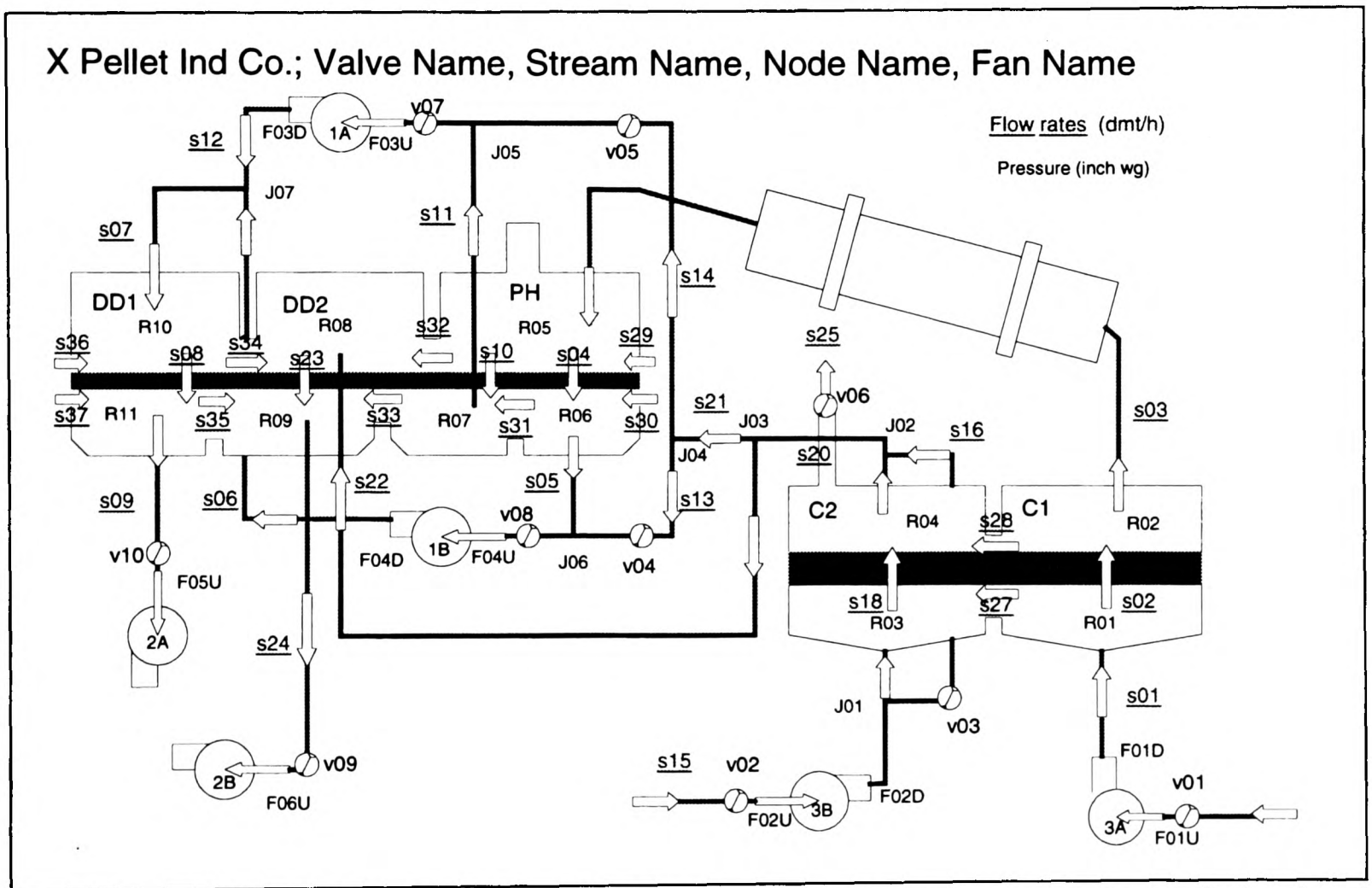

Figure 5.2 Stream, node, valve and fan names for the reference output of simulated network

In further discussion of results and runs, the names of concerned stream, node, valve and fan would be mentioned, which are shown in Figure 5.2. To show that the model produces physically valid and reliable results, variation of parameters of some network components are described in the next section.

5.3 GASFLO Validation

Model validation is to demonstrate that the output produced by the model is physically realistic and quantitatively exact or within an acceptable range of the observed or measured values. We validate the GASFLO model by carrying out parametric studies for all entities of the components i.e. pipe, fan, valve and packed bed; and verifying that the produced output conforms to the physics of the process.

For space limitation reasons, results from the eight of the performed parametric study runs are discussed. The referred node and stream names are shown in the Figure 5.2. The key node pressures and stream flows are compared in Tables 5-2 and 5-3 respectively. The remaining stream flows can be computed easily from these given values, e.g. the difference of flows in *s01* and *s02* is the flow through the internal leak i.e. stream *s27* connecting the regions *R01* and *R03*. The flow is always from the higher to lower pressures except the streams containing fans. Similarly for node pressures, the junction pressures have mostly been skipped from the Table 5-2, while pressures at the nodes above and below them are mentioned e.g. pressure at *J06* has been skipped and it can be evaluated by the pressures mentioned for *R06* and at the fan *1B-Entry* in the table.

The inputs and resulted outputs are briefly described. In these runs the input to one of the parameters of a component is changed to see the effect clearly without any interference. This is also to reduce the complexity of produced effect which in case of multiple simultaneous changes would be hard to analyze.

(1) The Reference output: As mentioned in the last section it was achieved by adjusting the fan efficiencies, leak areas and discharge coefficients and pipe efficiency factors. For further runs values of all of these adjusted parameters will remain fixed. The leak no 5 or *s32* was closed in the actual set-up for which the set of measured values was provided, and it will remain so for all subsequent runs. The discharge coefficient of leak no 11 i.e. *s37* was set to zero, to achieve the minimum possible region pressure for *R11*, otherwise its opening will increase the respective region pressure (see run 5). This output will be used as base output for the comparison to other runs.

(2) Increase of Local Friction Factor for a Pipe: The friction factor for pipe number 19, which links *J03* to region *R08*, and represented by *s22* was doubled. This increased the pressure drop across this pipe from 0.9" to 1.7", decreased the flow through the stream by 11 tph, and because of this 'restriction effect' the pressures upstream to this pipe in regions *R04* and *R03* increased, whereas at downstream nodes *R08* and *R09* they decreased. These changed pressures accordingly effected the internal leaks among these regions. The effect was more prominent in the zones C2 and DD2 of the system. The results for the decrease of the friction factor for this pipe, though not presented in the tables, had entirely opposite affect.

(3) Closing of a Valve: For this, the selection of valves like *v01* or *v02* would effect the whole system down stream and complicate the analysis, instead *v03* was chosen and changed its opening from 70% to 35%. This decreased the flow in bye-pass stream *s16* from 57.0 tph to 29.0 tph, and added it to the input of the zone C2. It also increased *J01* and *R03* pressures, but unexpectedly it decreased the pressures at onwards nodes (i.e. *R04, J02, J03, R08* and *R09*). This was due to increase of flow in *s18*, i.e. flow through the packed bed, which resulted in a higher pressure drop across the bed ($|R03-R04|$) of 6.8" instead of 6.2". The region pressures also effected the neighbouring regions through leaks.

(4) Increase of Leak Area: The cross sectional area of leak no 4, i.e. *s30* was doubled by increasing its height. Obviously this nearly doubled the stream flow, it increased the pressures for adjacent regions *R06* and *R05,* which increased the pressures upstream as well as downstream to these regions. However, the effect on upstream nodes is not as pronounced as on the downstream nodes. The increase of 14.0 tph in *s30* was compensated by decreasing the flow of other two external leaks *s29* and *s36* by 12.0 tph. The zones C2 and DD2 remained nearly unaffected.

(5) Opening of the Closed External Leak: The leak 11 i.e. *s37* which was initially closed in the reference run was opened, by just a small amount i.e to 2.0 cm of height (as leaks area is width of the bed multiplied by leaks height). It sucked in 26.0 tph, and increased the regions *R11*, *R09* pressures from -8.2", -8.6" previously to -7.3", -7.8" respectively. This increase in outlet region pressures decreased the flow through the beds i.e. in streams *s08, s23* by 29.0 and 14.0 tph respectively. The other external leak *s36* remains unaffected (because the *R10* pressure was not affected), but the flow entering to zone DD1, i.e. in stream *s07* is decreased by 43.0 tph and PH zone pressures are increased (see Section 5.4.2) thus sucking less air from atmosphere i.e. flows in streams *s29* and *s30* are decreased.

(6) Use of Fixed Pressure Gain for Fans 1A and 1B: As described in Chapter 2 and will be further discussed in Section 5.4, the entity fan can use an alternate mathematical model. In reference and all previous runs all fans used the fan equation (see Section 2.4), in which pressure gain introduced by the fan is dependent on the pressure on the entry and on its throughput. This equation has a realistic and self compensating effect, whereas the alternate fan model introduces the fixed pressure gain into the stream. In reference run the fans 1A and 1B introduced 11.8" and 11.3" of pressure gains and their throughput were 276 tph and 245 tph respectively. In this run the fixed pressure gain of 10" for each of these fans was introduced. This resulted in less suction (i.e. increase in upstream region pressures *R06, R07, R05* and *R02*), less push (i.e. decrease

Table 5-2 Node pressures for parametric study runs

	NODE PRESSURES in Inches of Water Gauge							
Run ->	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Node Name	Rfrnce							
3A-Exit	11.0	11.1	11.0	11.1	11.2	11.2	11.0	11.1
R01	10.7	10.8	10.7	10.8	11.0	10.9	10.7	10.8
R02	-0.6	-0.6	-0.7	-0.5	-0.2	-0.3	-0.7	-0.5
R05	-0.9	-0.8	-1.0	-0.8	-0.5	-0.6	-1.0	-0.8
R06	-9.0	-9.1	-9.1	-8.7	-7.8	-8.1	-9.4	-9.4
1B-Entry	-10.5	-10.5	-10.5	-10.2	-9.2	-9.4	-11.0	-10.8
R07	-9.0	-9.0	-9.1	-8.7	-7.8	-8.0	-9.2	-9.4
1A-Entry	-10.8	-10.9	-10.9	-10.6	-9.1	-9.4	-11.0	-11.1
R10	0.3	0.1	0.2	0.4	0.3	0.0	0.3	0.2
R11	-8.2	-8.3	-8.3	-8.0	-7.3	-7.9	-8.2	-8.9
2A-Entry	-18.9	-19.0	-19.0	-18.7	-18.0	-18.6	-18.9	-19.6
3B-Exit	5.8	6.2	6.3	5.9	6.1	5.9	5.8	5.9
J01	5.8	6.1	6.3	5.9	6.1	5.9	5.8	5.8
R03	5.7	6.0	6.1	5.7	5.9	5.7	5.6	5.7
R04	-0.5	-0.1	-0.7	-0.4	-0.2	-0.4	-0.6	-0.5
J03	-0.6	-0.1	-0.8	-0.5	-0.3	-0.5	-0.6	-0.6
R08	-1.5	-1.8	-1.7	-1.4	-1.2	-1.5	-1.5	-1.6
R09	-8.6	-8.8	-8.7	-8.4	-7.8	-8.2	-8.6	-9.3
2B-Entry	-19.9	-20.0	-20.0	-19.7	-19.0	-19.5	-19.9	-20.6

in downstream region pressure *R10*) and a decrease of overall flow through these two fans from 521 tph to 474 tph. The internal and external leaks were also affected correspondingly.

(7) **Increase of Fixed Pressure Gain to Fans 1A and 1B:** Here instead of the 10" pressure gain it is increased to 12". This caused an effect opposite to the previous run by increasing the suction and push. It also increased the overall throughput of the two fans from 521.0 to 531.0 tph. The effect of this increase is more visible on suction

Table 5-3 Stream flows for parametric study runs

	STREAM FLOWS in Tonnes/hour							
Run ->	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Strm Name	Rfrnce							
s01	390	390	391	389	387	388	391	389
s02	384	384	385	383	381	382	385	383
s03	387	391	384	386	381	379	389	383
s29	55	53	58	51	39	45	58	52
s04	195	196	195	193	186	187	198	192
s30	15	15	15	29	14	14	15	15
s10	247	248	247	244	234	237	249	243
s07	521	523	520	527	478	474	531	509
s36	237	244	242	229	235	252	233	241
s08	579	580	579	577	550	562	581	573
s15	422	417	414	421	418	421	423	422
s16	57	56	29	56	56	56	57	57
s18	372	367	390	370	368	370	372	372
s20	251	241	247	250	250	256	250	255
s22	202	191	198	202	204	209	200	204
s23	381	378	380	380	367	373	382	381

side, i.e. the regions *R06, R07* pressures are decreased by about 1.2" as compared to the run (6).

(8) Increase of Bed Height in Drying and Preheat stages: The bed height for all of the four beds, corresponding to streams *s04, s10, s23* and *s08* are increased by 10% i.e. from 14.5 cm to 15.95 cm. This is in fact equivalent to the increase of production of the plant by 10%, which is always one of the goals for practitioners. This increased the pressure drop across the packed bed in drying and preheat zones and decreased pressure in all of the regions from *R06* to *R11* noticeably. However the flow distribution is not much affected. This case will be considered further in Section 5.6.2,

where it is shown how the flow distribution can be increased throughout the system by a similar amount (i.e. 10%) to optimise the induration process.

All the above runs show that the behaviour of GASFLO model is very physical and provides detailed information about the system variables which is otherwise not possible. Its use provides insight to the induration process and can assist in control of the system. As seen it can identify flows entering into the system through leaks, their location and magnitudes, which are neither practically measurable nor known.

5.4 GASFLO Capabilities

Although GASFLO is still evolving, it is a complete stand alone software tool. It can compute flow, pressure and temperature distributions (see Sections 3.7-8 and Section 5.4.6) in pellet induration systems. The algorithm used for computation transforms the original system into a connected graph and partitions it further into forest (combination of trees or acyclic graphs) and coforest (the streams of the connected graph which are not contained in the forest) structures. All this partitioning is done by the GASFLO itself. The partitioning and solution algorithms are discussed in detail in Chapter 3, the development of GASFLO model is discussed in Chapter 4; how to use GASFLO is explained in Appendix A.

The qualities of GASFLO code from software engineering perspective are described in Sections 4.5 and 4.6.6. GASFLO is comprised of two main programs; PREPNET to prepare the network and CMPNET to compute the prepared network; 88 subroutines (including all function subprograms, subroutines and entry points); and 6100+ lines of code (including 48% comments). The working and pre-requisites of main programs are explained in detail in Appendix-A. The present (or developer's) version of GASFLO includes the needed debugging facilities e.g. writing internal per iteration or per step values to debugging files, and sends output to terminal for guidance. This version uses DBOS, the SALFORD F77/386 compiler's run-time library and extended memory manager which exploits the 80386 and later

processors' hardware. For a typical run, like mentioned in the last or here in this section GASFLO took about 27 seconds on a 50 MHz 486 machine using DBOS version 386, and this time reduced to 17 seconds when the proper DBOS 486 version was used. The used tolerance for error checking was $5.0x10^{-03}$, where error is the maximum of the relative error in mass balance at any of the internal nodes. Since the flows are computed in Kg/sec and this tolerance works out to be 0.018 tph whereas the accuracy of flowmeters is ±1.0 tph. The removal of unnecessary output to slower devices and to debugging files would improve these computational times even further.

In this section the GASFLO will be looked at from the functionality perspective, as a software tool to evaluate airflow distribution in pellet induration systems, that is from the practitioner's or operators point of view what it can do. In practice the real capabilities of any industrial software tool are dependent on its use, most of the time these are demanded by the practitioners and then added by the developer. However as a pre-requisite to this stage, first the software tool should at least be in useable form and give physically reliable results. The capabilities stated in the following are to justify that context.

5.4.1 Leaks Inclusion and Exclusion :

The inclusion and exclusion of leaks is one of the facilities which was embedded in the very early stages of the development of GASFLO. The user can simulate the system in an ideal state by opting to exclude all the leaks (internal; between the neighbouring regions, as well as external; between the system and atmosphere) of the system. In practice, the exclusion of leaks is impossible, although it has always been the wish of the system operators for efficiency and ease of operation reasons. However, from computational or analysis point of view it does provide a good insight to the system.

Figure 5.3 shows the output from GASFLO for the example system and with same boundary conditions as were for the reference output in Figure 5.1. Figure 5.3 demonstrates that if there are no leaks all the flow going out of the system through fans 2A, 2B and Stack,

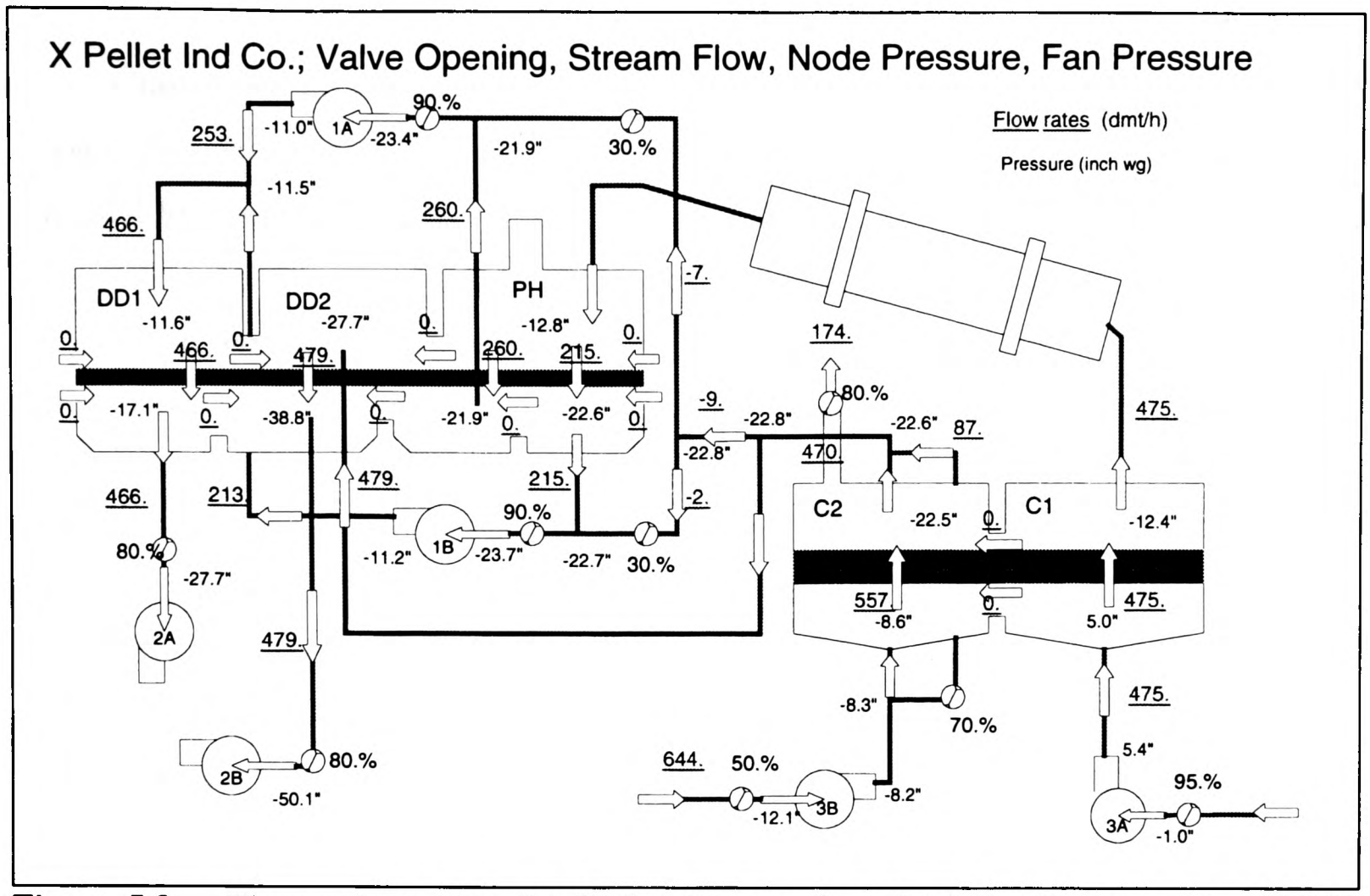

Figure 5.3 Flow and pressure distribution in example pellet induration network with NO LEAKS option

is to be pumped in by the suction fans 3A and 3B. The flows through the suction fans, in the absence are noticeably large and can give maximum pressure drop across cooling zone beds. Also in outlet regions these pressures can reach as low as -39". The network did not reduce to two independent subnetworks and exchange of flow between the two is visible. Incidently the flow direction in streams *s13, s14* and *s21* is reversed because the pressure at *J03* is less than the pressures at *J04, J05* and *J06*. For NO LEAKS option GASFLO assigns zero flow to all leak streams and excludes them from computation, as in this case it treated 26 streams rather than 37 (leaving 11 streams comprised of leaks) which reduces the computational load by a significant amount.

5.4.2 Fans Selectable Alternate Mathematical Models :

This is a versatile facility, from the software, computational and practical usage points of view. As seen in Section 5.3.4, runs (6) and (7) the alternate model of fixed pressure gain was used for fans 1A and 1B.

From modellers perspective, in GASFLO it is possible to add component's models incrementally; initially one can start with a coarse model of a component based on its available *shallow or heuristic* knowledge, which can be replaced by a refined model, later, on the availability of *deep or process based* knowledge. The provision of alternate run-time selectable models for fan entity illustrates a step further, that the coarse model is not only replaced by the refined model but these both co-exist in the same code.

The fixed pressure gain model for fans was included in the later stages of the development, after feed-back from the users that it is possible to adjust the fans such that they provide fixed pressure gain. Indeed, in the original fan model (Section 2.4) the pressure gain provided by a fan is dependent on - pressure of air at fan's entry, its throughput, efficiency and wattage; the analysis using this fan model for some cases was too complex to resolve, especially to isolate the effect of variation of any of these parameters for a specific fan on the system was difficult. For example in run (5) of the previous section, opening of an external leak *s37* decreased the flow in *s01* i.e. throughput of fan 3A. The use of the original fan model provided a higher pressure gain since the pressure gain is inversely proportional to the throughput which decreased. This further reduced the flow in *s02*, the flow through packed bed in zone C1, whose model i.e. Ergun's equation also resulted in lower pressure drop, so both of these resulted a net increase in pressure on all down stream nodes *R05, R06* and *R07* etc. The overall effect of fan model was realistic, but its dependence on multiple parameters complicated the 'cause and effect' analysis.

5.4.3 Fixed Pressure Region Nodes :

GASFLO can cater the situations when the pressures at the inlet regions are known and these are kept to be fixed. This is analogous to *fixed grade nodes* in hydraulic networks.

From the practitioner's view it is a very useful facility. The leaks into the system are always dependent on the region pressures, so if they could be controlled, then it is possible to control leaks to some extent. The simulation of fixed pressure regions is possible only for inlet regions of the zones. When GASFLO is asked to simulate for a known region pressure and keep it fixed for the run, then it first verifies that it is an inlet region, looks for the pipe upstream to the respective region, computes the calibration factor for that pipe, and instead of computing its downstream pressure when computing pressure distribution in the corresponding tree (see Algorithm in Figure 3.3 for network computation) it simply initializes it with the fixed known value.

Figure 5.4, shows a scenario, when pressures for *R03, R05, R08* and *R10* are respectively fixed to 4.7, -1.5, -2.5 and -0.3 inches of w.g. The effect on the outlet region pressures and on flows specially on external leaks *s29* and *s36* are noticeable.

5.4.4 Insertion of Wind Box to a Zone :

Practically, the insertion of an extra wind box, means extension of packed bed by a certain length say 10 meters, which is a full-fledged engineering project and would involve a significant amount of work. Such extension may help the induration process, depending on the zone or stage being extended. It will result in complete wastage (both in time and finance) if the expected positive results are not achieved.

Without a software tool like GASFLO such design problems and their after effects cannot be studied inexpensively. The addition of a wind box will have different effect on the overall performance of the system depending on the zone to which it is added. GASFLO can

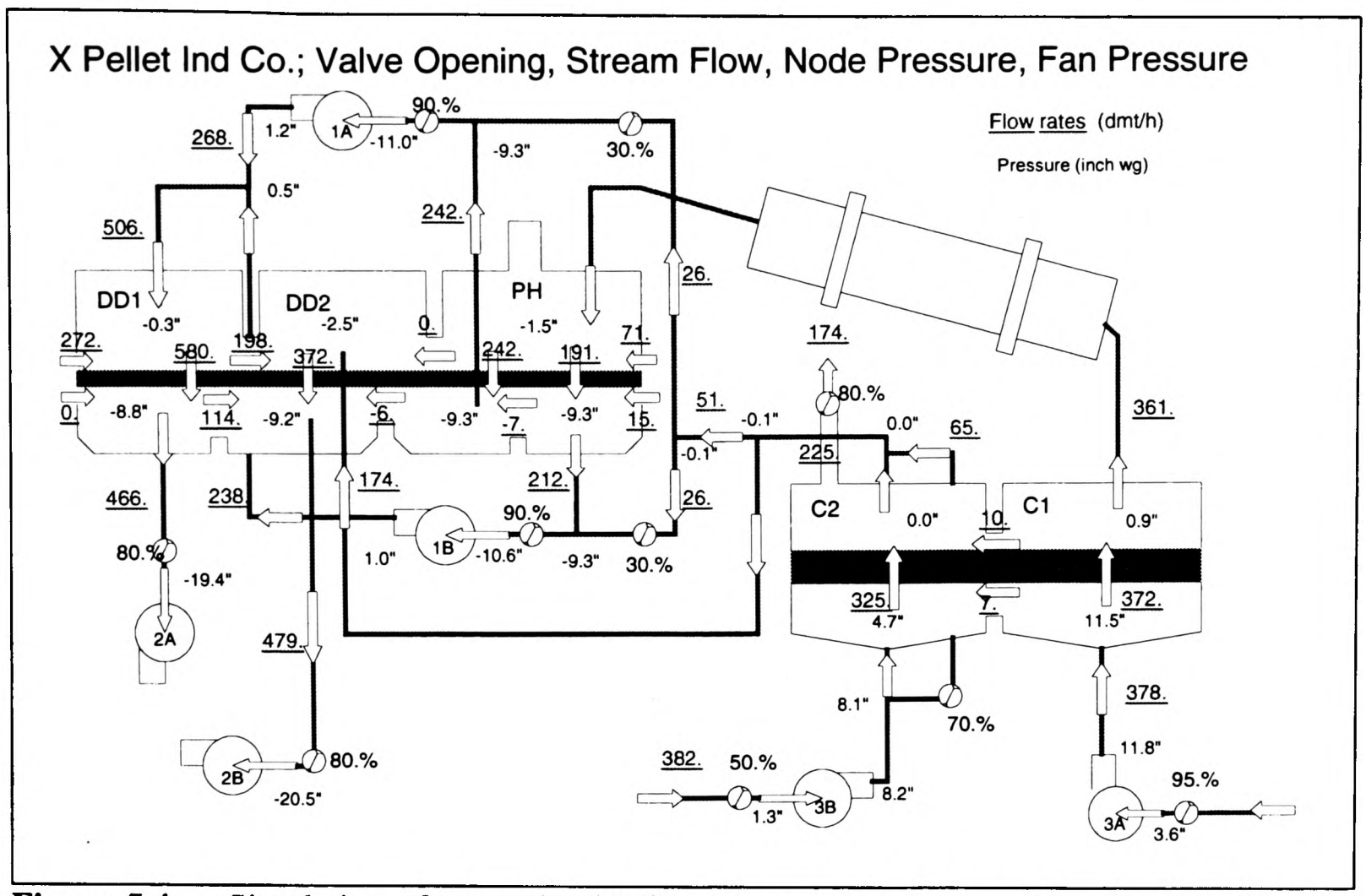

Figure 5.4 Simulation of network with fixed pressures at inlet regions to zones C2, PH, DD2 and DD1

easily simulate such an addition by varying the area of respective packed bed, and can show how the system will be effected.

The addition of a wind box to zone PH, specifically to the bed linking regions *R05* and *R06* was simulated by increasing the area of bed corresponding to *s04* (Figure 5.2) by 25% i.e. from 43.05 m^2 in reference run (Figure 5.1) to 53.81 m^2. The results of addition of wind box to the bed linking the regions *R05* and *R06,* are compared to the reference output in Table 5-4. This increased the flow in stream *s04*, since the bed area was increased, which was compensated by increasing the external leak *s29* and decreasing the flow through other packed bed no 5, i.e. *s10*. The flows and pressures, upstream to region *R05* and downstream to *R06* were affected and their values are shown in the Table. Quite surprisingly the flows and pressures, except leak linking *R07* and *R09* i.e. *s33* which decreased from 15 tph to 9 tph, in zones C2 and DD2 mostly remained unaffected by this change.

Table 5-4 Effect of addition of a wind box to zone PH, next to leaks streams *s29* and *s30*

	Region Pressures and Flows for Added Wind box Case				
	Reference	Added W-Box		Reference	Added W-Box
Region Pressures (" of w.g.)			Stream Flows (tph)		
R01	10.7	10.5	s01	390.0	393.0
R02	-0.6	-1.0	s29	55.0	66.0
R05	-0.9	-1.3	s04	195.0	229.0
R06	-9.0	-8.4	s10	247.0	233.0
R07	-9.0	-8.5	s07	521.0	533.0
R10	0.3	0.4	s36	237.0	224.0
R11	-8.2	-7.9	s08	579.0	576.0

5.4.5 Introduction of Cross Flow :

This is another practically expensive scenario, in which some part of flow from the inlet region of one zone is redirected to the outlet region of another neighbouring zone.

Provision of this capability in GASFLO required a noticeable effort. Previously the computation of packed bed streams could only be executed as tree streams, but simulation of the cross flow required the option for their computation as 'torn' streams also (Section 3.3). Because, for tree streams the flows are computed by solving the continuity equation at the downstream node, whereas for torn (or cotree/coforest) streams' flows are computed using the values of pressures at the two nodes connected by the respective stream. This required the improvement of partition algorithms (to include a bed stream as torn stream) and extension of the software module corresponding to bed entity (to compute bed as a torn stream).

Figure 5.5 shows the results of cross flow from the inlet region (*R08*) of DD2 zone to the outlet region (*R07*) of PH zone. This was simulated by introducing another bed which linked these two regions. In fact for this case we just reduced the bed area of bed 4 (corresponding to *s23*) by 25% and assigned it to new bed. The results can be compared with the reference output. This output also highlights the role of internal leaks, one can see to what extent the redirected flow of 91 tph from *R08* to *R07* is compensated by the internal leaks from the neighbouring regions. This redirection increased pressures in *R07* and *R06* by 0.8"

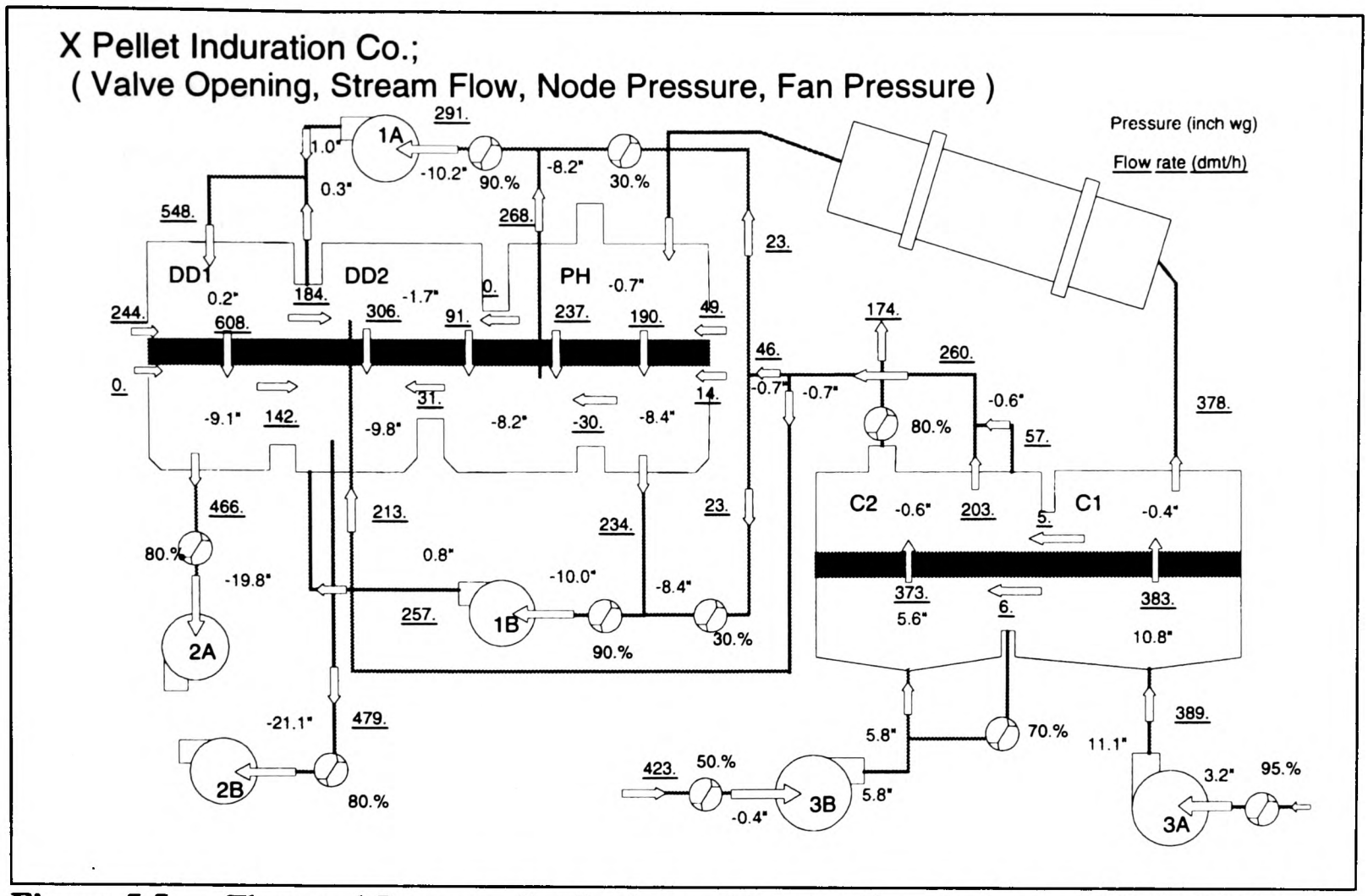

Figure 5.5 Flow and Pressure distributions with cross flow from zone DD2 (*R08*) to the zone PH (*R07*)

and 0.6" respectively, whereas it decreased pressures in regions *R08, R09* and *R11*. The pressure at *R10* remained almost un-effected due to its exposure to atmosphere through leak *s36*.

5.4.6 Evaluation of Temperature Distribution :

GASFLO initially computes the flow and pressure distributions in the network and using converged values of flows, it evaluates the stream and node temperatures. For this computation the basic principles are:

- All streams leaving a node should have same temperature as the node temperature (Kohler et al 1990).
- Overall heat entering a node is same as leaving the node. Node temperature is computed by this conservation equation assuming that there is complete mixing and the node is at the same temperature.

- The temperatures related to packed bed are computed by another software tool INDSYS, in GASFLO these are used as average temperature for the computation of Ergun's equation.
- No temperature loss is assumed for flow through fans, leaks and valves whereas convective and conductive heat loss through pipe surface is included in the model. The pipe material's thermal conductivity (which is not yet known) has been used as adjustable parameter for temperature computation calibration.

The temperature equations do depend on flow distributions, whereas the latter does not depend on temperature distribution i.e. the respective two equation sets are loosely coupled and are computed independently (see sections 3.1, 3.7 and 3.8). In the same run, GASFLO first evaluates flow and pressure distributions, then using the converged values of flows it evaluates temperatures. The node and stream temperatures for the flow distribution shown in Figure 5.5 are presented Figure 5.6.

Although from the induration process perspective 'kiln' is a significant component, from the view of airflow evaluation (i.e. from GASFLO perspective) it is simply a pipe or duct. All process gas entering the kiln is being passed on to the PH zone. So, the pipe upstream to the kiln, kiln and the pipe downstream being connected serially are replaced by an equivalent pipe (pipe no 3 in the case of the example network), which is further simulated by stream *s03*. INDSYS provides the temperature of gas flowing out of the kiln, which GASFLO simply equates it to the respective stream (i.e. *s03*) temperature. The leak stream temperature are the temperatures at their upstream ends.

5.5 INDSYS - GASFLO Interaction

INDSYS (INDuration SYstem Simulator) was developed in mid 1980's and has been used by induration industry since then. It computes the heat concentration in the key components of the induration process, namely the packed beds and kiln, and takes into account the heat transfer, involved chemical reactions and efficiencies of the heat sources -

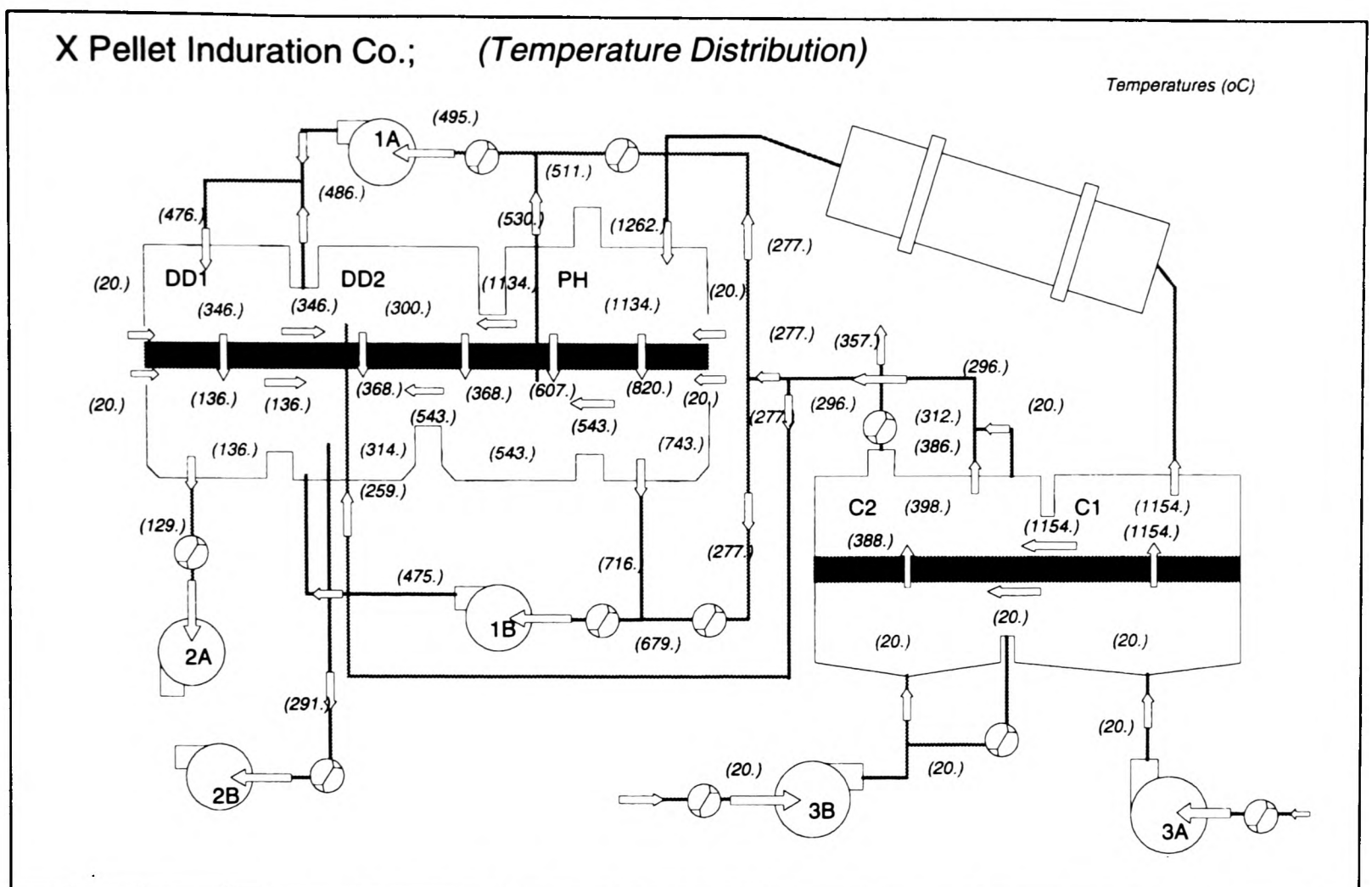

Figure 5.6 Temperature distribution for cross flow - bed height 14.5 cm

burners etc. The computation is based on a complex mathematical model of the process and done at microscopic level. For heat transfer from one zone to another, air is used as process gas whose flow distribution is needed by INDSYS to compute exactly the heat concentration in the system and amount of heat transfer from solids (the pellets) to the process gas (the air) and vice versa in different zones. Details about INDSYS computation, working and usage are described elsewhere (e.g. Cross and Englund 1987, Cross 1988).

Here in the following subsections, we will briefly discuss the data required by INDSYS and GASFLO from each other and how a combined simulation using the two software tools can be run to get more realistic, detailed and exact results for an induration system simulation.

INDSYS and GASFLO have been developed independently in different spans of time, they are written in different computer languages and have even assumed different network configuration schemes, each using the one which is more suited to its needs. INDSYS treats

induration network as combination of different zones, confines itself to the simulation of respective zones, and for airflow distribution it assumes how the input or on-gas of one zone is formed by the output or off-gas from the other zones. In terms of INDSYS airflow distribution, on-gas of DD2 consists of 80% of off-gas of C2 and 20% of off-gas of PH and these fractions remain fixed throughout the simulation. Whereas GASFLO considers the induration network as combination of nodes and stream, and simulates all those components which contribute to airflow distribution. It computes exact airflow for each of the streams; e.g. the on-gas of DD2 is represented by stream *s23* which is explicitly sum of streams *s22* (off-gas from C2) and *s34* (off-gas of PH with some mix of air through *s36*). At the time of GASFLO's development although the existence of the INDSYS was known it was developed in isolation without any influence of INDSYS.

5.5.1 INDSYS required data (to come from GASFLO) :

The degree of difficulty to determine flow distribution in pellet induration system is evident from chapters 1-4. The developers of INDSYS and other such packages (e.g CASCADE Patel et al 1993) realized the involved complexity and so used guessed flow distributions instead of indulging into the computation of exact flow distributions. As a result these packages require several runs to reach to good guess of airflow distribution by hit-and-trial, nevertheless this approach gave them a good start. However, the validity of such guessed flow distributions is not guaranteed.

Now GASFLO can provide an exact airflow distribution which was needed by INDSYS (and its counterparts) and in return the temperatures of gas in packed bed which are assumed in GASFLO computation can be extracted from INDSYS results.

INDSYS requires:

- Exact values of flows entering into the inlet (the regions upstream to the packed bed) and outlet (the regions downstream to the packed bed) regions of all zones,

- Temperatures of these flows entering into the regions.

Since the internal leaks are recuperated from within the system, their inclusion in one region will be taken as negative flow from the other region, so these are ignored and attention is focused to the flows being sucked into the system from atmosphere whether through suction fans 3A and 3B, or through external leaks. Similarly the bye-pass stream (*s16*) of C2, having all attributes of ambient air would be treated as if it is added to the outlet region of zone C2.

5.5.2 GASFLO required data (to come from INDSYS) :

As mentioned in last section, these are:

- The average temperature of gas entering and leaving the packed bed for each zone,
- The temperature of gas coming out of kiln and entering PH zone.

INDSYS outputs the temperature distribution of gas and solids in matrix form, assuming symmetry from front to back, discretising it vertically in 11 layers and horizontally depending upon its length in number of intervals of equal length. Luckily it also output the average temperatures entering and leaving the bed, which can be used as input to GASFLO directly.

5.5.3 INDSYS - GASFLO combined simulation :

Ideally both of these software tools should be integrated to a single simulation tool, as they are simulating different interdependent subprocesses of the same induration process, but this integration will involve a significant amount of work. With their existing states, the following steps can enable the use to run a combined simulation:

Step-1: Run INDSYS with an assumed flow distribution for the simulated network,

Step-2: Extract from the output of INDSYS; the average gas temperatures entering and leaving the packed bed for each zone and temperature of gas leaving the kiln. Treating these temperatures as input parameters and run GASFLO,

Step-3: From the output of GASFLO, extract gas flows entering to cooling stages C1 and C2, and external leaks corresponding to streams *s29, s30* and *s36*. Consider these new flow distributions as input and run INDSYS.

The steps 2 and 3 should be repeated until converged flow and temperature distributions are achieved.

This combined simulation will require the understanding of the adopted network configuration strategies, and the knowledge of procedure to run each of these tools. A mapping of zones modelled by the INDSYS to the packed beds simulated by the GASFLO should be first worked out. For example in the modelled network INDSYS treats the pre-heat (PH) zone as a single zone whereas GASFLO according to its own configurational strategy, simulates it as combination of three regions *R05, R06* and *R07*, and two beds corresponding to streams *s04* and *s10*. So the temperatures output by INDSYS for PH zone will need some interpolation to conform to GASFLO input. Similarly stream *s16* which is bye-pass to cooling zone C2 is treated by INDSYS as an external leak entering into the system in outlet region of zone C2. Such intricacies does need a working knowledge of both software tools.

The network shown in Figure 5.5, the one with cross flow, was simulated using both of these packages. The combined simulation took three iterations to converge. The results, comprised of key stream flows and temperatures, are shown in Table 5-5. The '0' iteration corresponds to the INDSYS computation using a well guessed flow distribution, and the temperatures computed by INDSYS in '0' iteration column were used by GASFLO iteration '1'. The flows computed by GASFLO in iteration '1' were used for INDSYS iteration '1', and so on. The variation between these key flows and bed temperatures was negligible after 3rd iteration hence further iterations were stopped. It has been noticed that the combined simulation requires 3-5 iterations to converge.

Table 5-5 INDSYS-GASFLO interactive simulation results

	Key Stream Flows and Bed Temperatures			
Itr no.	0	1	2	3
Flows from GASFLO (tph)				
Through Fan 3A	434.0	386.0	384.0	384.0
Through Fan 3B	528.0	424.0	414.0	413.0
Stream s16	10.0	57.0	61.0	61.0
External Leak s29	20.0	35.0	41.0	42.0
External Leak s30	17.0	15.0	15.0	15.0
External Leak s36	189.0	259.0	265.0	266.0
Bed Tempr from INDSYS (^{o}C)				
Zone DD1 - IN	520.4	445.0	440.6	440.4
OUT	137.1	130.0	130.0	130.3
Zone DD2 - IN	605.9	675.1	678.1	678.9
OUT	366.6	296.8	292.4	292.9
Cross flow - IN	471.3	579.2	582.8	583.7
OUT	611.5	540.4	535.5	535.4
Zone PH - IN	1214.0	1155.7	1138.0	1136.0
OUT	831.8	832.7	830.0	830.0
Kiln - OUT	1261.7	1244.0	1240.6	1241.0
Zone C2 - OUT	400.3	611.5	631.2	632.6
Zone C1 - OUT	1152.0	1210.1	1213.1	1213.8

5.6 Case Study

In the case study we simulate one of the practical scenarios of how the system's production can be increased by a fixed amount say 10%. In other words we have to increase the pellets input by 10% which can be done by increasing the beds' height in zones DD1, DD2 and PH by 10%, and increasing the overall gas flow through the system, especially the recuperated flow, by the same amount. The increase of gas flow through the system is important to complete the induration process and for thermal efficiency reasons.

For this, first we see the behaviour of flow and pressure distributions when the flow through exhaust fans 2A and 2B is decreased or increased. Later after knowing the source for this extra gas which is being pumped out; we increase the beds height and flow through the exhaust fans by 10% and determine the necessary adjustments in the other fans parameters or valve openings, to achieve the desired increase of gas flow through the suction fans 3A and 3B.

5.6.1 Decrease and Increase of Exhaust Fans Flow :

The case study is carried out on the same induration system which is shown in Figure 5.5. We keep the bed height same as 14.5 cm and vary the flow through the exhaust fans 3A and 3B.

Since these exhaust fan flows are used as boundary condition by GASFLO, so their changed values are fed into the input file. We decrease and increase these flows by 5% and 10%, keeping all other parameters constant. The results for these four along with the standard case are presented in Table 5-6. The flows through exhaust and suction fans and leaks are the main influencing variables, so these are compared to see the effect on pressures, the pressure drops across the beds in zones are also given.

The effect of this increase or decrease is more prominent on the external leak *s36*, then on other external leaks *s30, s29*, and the flows through the suction fans 3A and 3B are least effected. Similar effect can be noticed from the pressure drops across the zones; DD1 and DD2 are most effected and the effect on cooling zones C1 and C2 is comparatively negligible.

Results given in Table 5-6 show that an increase (decrease) of 10% i.e. ± 94.5 tph in the overall exhaust flow; produces an effect of +78 tph (-82 tph) in external leaks and an effect of +17 tph (-13 tph) in suction flow through fans 3A and 3B. Thus, the increase or decrease of suction flow is mostly compensated by external leaks into the system rather than by the suction flow as expected and desired. Similar effects were noticed practically by the

Table 5-6 Effect of decrease/increase of exhaust flow on the system

	Stream Flows and Zones Pressure drops				
Change ->	-10%	-5%	Standard	+5%	+10%
Flows Through (tph)					
Exhaust Fan 2A	419.0	443.0	466.0	489.0	513.0
Exhaust Fan 2B	431.0	455.0	479.0	503.0	527.0
External Leak s36	182.0	213.0	244.0	275.0	306.0
External Leak s30	13.0	14.0	14.0	14.0	15.0
External Leak s29	30.0	40.0	49.0	57.0	64.0
Suction Fan 3B	414.0	419.0	423.0	429.0	436.0
Suction Fan 3A	385.0	387.0	389.0	391.0	393.0
Pressure Drop (" of W.G.)					
Zone DD1	7.9	8.5	9.3	10.2	11.0
Zone DD2	6.7	7.4	8.1	8.8	9.6
Zone PH	7.1	7.3	7.6	7.8	8.1
Zone C2	5.9	6.1	6.2	6.3	6.6
Zone C1	11.1	11.2	11.2	11.4	11.6

practitioners on the actual system and are reported elsewhere (Afzal and Cross 1992). This also reveals that if the overall flow through the system is to be increased than some mechanism to restrict these external leaks would be required.

5.6.2 Increase of Pellet Production rate by 10% :

Some runs of GASFLO will be required to simulate this situation. As seen in the last sections, the increase of production rate needs, the bed height as well as the gas throughput of the system to be increased by 10%. Since GASFLO uses the exhaust fan flows as boundary condition so they can be increased straightaway, but (as seen in Section 5.6.1) this will increase the external leaks rather than the recuperated flow through the cooling stage or through suction fans. Also the increase of bed height as seen in run (8) of Section 5.3 would increase the pressure drops across the beds, thus giving a lower pressure in respective outlet regions, which would increase the external leaks.

To resolve the problem, we first determine the approximate values of external leaks which would indirectly provide us the desired increased recuperated flows. The leaks are governed by the respective region pressures (*R05, R06* and *R10* in this case), so if we could know the values for these region pressures, then a calibration like procedure can adjust the parameters systematically to achieve these target region pressures or external leaks. The GASFLO run with the original exhaust flows and bed height (14.5 cm), whose results are shown in Figure 5.5, can provide these values. These leaks (through streams *s36, s30* and *s29*) should be in the range of 244.0, 14.0 and 49.0 tph respectively and the associated region pressures for regions *R10, R06* and *R05* should be 0.2, -8.4 and -0.7 inches of w.g.

The 10% increase in height and in exhaust flow are introduced simultaneously and the alternate (fixed pressure gain) fan model is opted for the booster and suction fans. Because

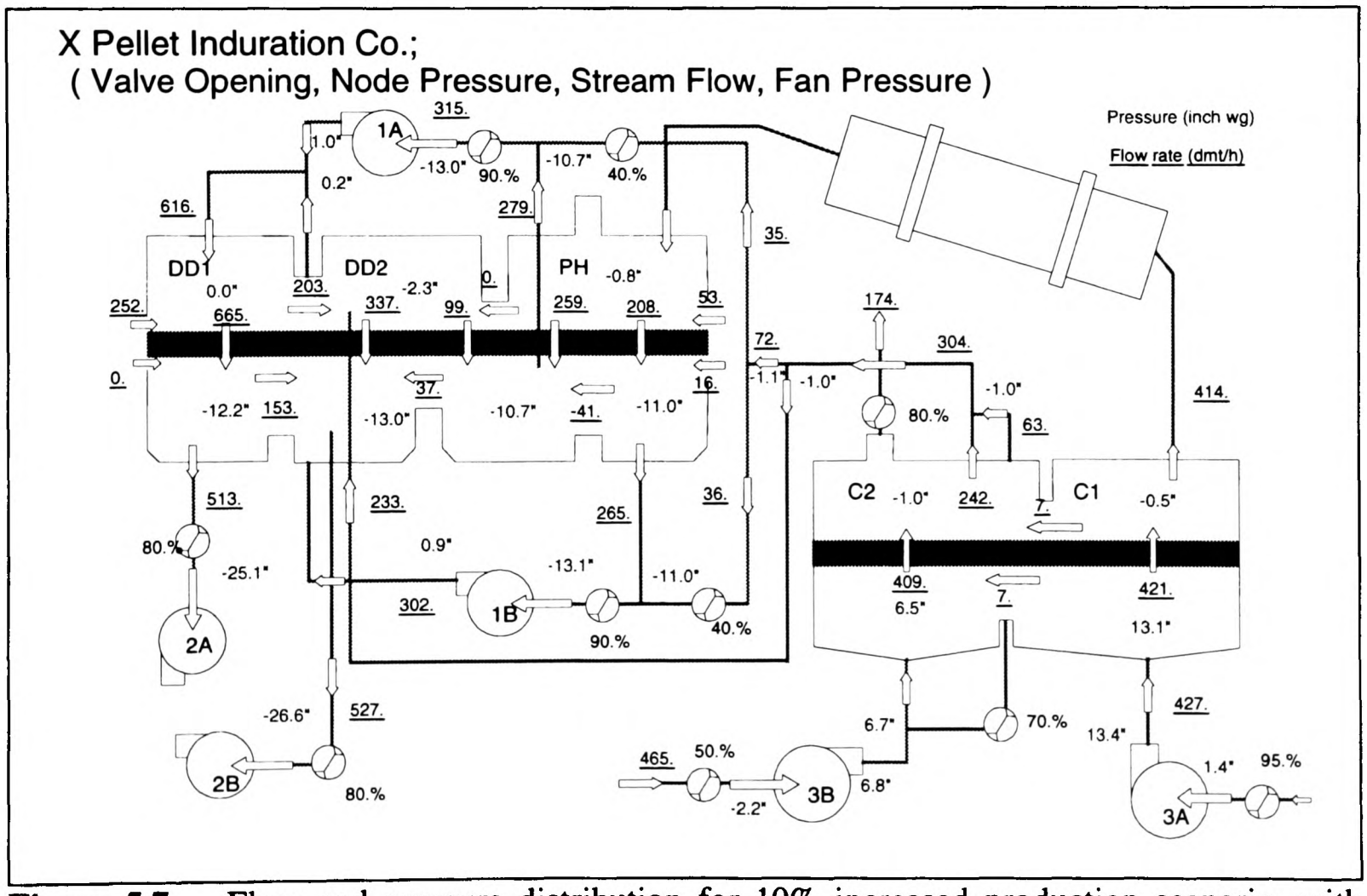

Figure 5.7 Flow and pressure distribution for 10% increased production scenario, with 10% more flow through suction and exhaust fans

as stated previously, the effect of original mathematical model for fan entity is quite complex, whereas the alternate model is easy to manipulate and observe the produced effect. It took few runs to achieve the desired output, which is shown in Figure 5.7, and it shows that 10% more flow has been sucked through the fans 3A and 3B. The aim throughout all these runs have been to increase the throughput of fans 3A and 3B by 10%, while using the above stated region pressures for *R05, R06* and *R10* as guide values so as to control the overall leak into the system. The resulted output (Figure 5.7) was achieved by using the pressure gains of 12.0, 9.0, 14.0 and 14.0 inches for 3A, 3B, 1A and 1B fans respectively, and the valves *v05* and *v06* were opened from initially 30% to 40%.

5.7 Summary

The calibration of model for the respective (to be modelled) network is an important and basic step. It needs to be performed before the analysis is undertaken. During calibration the adjustable and unknown parameters of the model are modified in the light of available observed (or field) data, such that the model gives results as close to those observed as possible. For hydraulic networks, using either Hardy Cross, Newton Raphson or linear theory methods, well defined algorithms exist for model calibration. But for GASFLO those algorithms could not be used as it based on a different computational approach so accordingly, an analogous but different strategy was devised and adopted for calibration, which caters specifically the pellet induration systems. This is explained in the first part of the chapter.

After calibrating the model for a real life pellet induration system, it is run for different parametric studies, to show that it gives physically valid and practical results. The capabilities of GASFLO are demonstrated by applying it to the commonly encountered situations of induration industry. This also shows that how this model (or more appropriately software tool) can assist the practitioners to resolve and simulate the real situations - like extension of a zone by introducing an extra wind box, and redirection of cross flow from inlet

region of one zone to the outlet region of another; which are otherwise impossible to analyze. The interaction of GASFLO with another software tool INDSYS, which computes gas temperatures in packed beds but used a guessed airflow distribution in the system, is described. The procedure and common data between these two software tools are given. In the last section a case study is described, which explains what fan and valve settings would be required if the production of the same system is to be increased by 10%.

All these applications briefly show that how GASFLO can support already existing software tools in the pellet induration field, and mainly how it can assist the induration systems' operators, designers and plant engineers in performing their routine jobs, as well as providing inexpensive, speedy, detailed and practically valid solutions to problematic situations, which would otherwise either be not possible or be very expensive in terms of time and finances.

Chapter 6
Conclusions

In this research, the development of a mathematical model and the associated software tool, GASFLO, have been described and the capabilities of the resulting code have been illustrated. GASFLO has been written in FORTRAN 77, for high-end PC compatibles especially for 486 machines, and evaluates steady state flow, pressure and temperature distributions of process gas in pellet induration systems networks.

The computation of GASFLO is based on *device-centred or unit based approach*, so that the network components or units are picked up and computed in the order of their connectivity. This approach lead to an elegant bi-level hierarchical algorithm for the network computation. The original pellet induration system network is transformed into a connected graph of streams (comprised of single or serially connected multiple network components having same flow) and nodes (where more than one stream meets or an external boundary of the system i.e. atmosphere). At the higher level the network is solved to satisfy the two Kirchhoff's laws, whereas at lower level the computation of respective streams is carried out to satisfy the mathematical models of their constituent components. The pressures and flows are interdependent and computed simultaneously, whereas the temperature and flow equations are loosely coupled, so the temperature distribution is computed using converged flow distribution. The computation algorithm requires the partitioning of the connected graph into *forest* (collection of trees i.e. acyclic graphs) and *coforest* (the streams of connected graph but not belonging to forest) structures, which has been automated by algorithms, based on heuristics specific to pellet induration systems.

The solution algorithms have been implemented in GASFLO. The results of GASFLO simulation application to a real pellet induration systems (whose data was available) show that they always converge and are versatile, robust and fast.

Realizing the involved complexity, and desired capabilities from the outset, the development of GASFLO was based on software engineering principles and techniques. The 'encapsulation' and 'information hiding principle' played a vital role in the resulting code. The resulting code is comprised of 2 main programs and 88 routines, and is 6.1+ KLOC (thousands of lines of code).

6.1 GASFLO Features

The present (developer's) version of GASFLO takes about 27 seconds to simulate a typical pellet induration system, on a 486 50 MHz machine with 386 version of DBOS, the Salford FTN77 run-time library and extended memory manager. This time reduces to 17 seconds if the more recent 486 version of DBOS is used. These timings can still be improved for 'users version' by avoiding the output to slow devices (i.e. terminal) and debugging related files (which was required for development). The output from GASFLO can be seen graphically on screen or printed on a PostScript printer.

From developer's view, the new network component entities can be added into the system by writing their respective software modules. The data and methods (including the mathematical model and its computation) related to the entity are encapsulated together into the corresponding software module, using SAVE and ENTRY constructs of standard FORTRAN 77. The modules are mutually well insulated and communicate with each other through a pre-defined generic interface. The developer can use any numerical scheme appropriate to the nature of equations of the mathematical model of respective entity. The existing modules' functionality and their data structures can be extended. This architecture

is flexible, powerful and the resulting code has very low maintenance costs (i.e. easy to change).

As a modeller's workbench, the mathematical model for any entity can be refined and its effect seen. The alternate mathematical models, coarse as well as refined, can be embedded into the same module and can be selected at run-time. The modelled process can be extended by adding corresponding code to the only effected entity modules. For example the computation of temperature distribution was added to GASFLO at a later stage of its development, its implementation required significant addition to node and pipe modules but was a simple initialization for leak module. This extensible and incremental nature of GASFLO makes it an ideal software tool which conforms to the modelling environment needs and character.

As a practitioner's assistant, GASFLO, can simulate pellet induration systems realistically, speedily and inexpensively. It can be readily used for operator training and analysis, and can communicate with another software tool, INDSYS, which computes heat concentration in pellet induration systems. It can simulate quite complicated but practical situations; like addition of extra wind box to a zone, cross flow from inlet region of one zone to the outlet region of another zone, or increase of pellet production rate etc; with least effort and can predict the disastrous situations whose implementation, otherwise, could lead to unrecoverable losses.

The solution algorithm uses sink node flows and source node pressures as initial condition for computation, which are known parameters for any pellet induration system. However, it seems to be a limitation from the generality aspect since presently GASFLO, cannot use other initial conditions to start its computation e.g. source node flows and sink node pressures as initial conditions.

6.2 Future Work

GASFLO has very flexible and powerful architecture, and especially: the fast algorithms; being founded on software engineering principles and hence independent *object* like resulting modules; the facilities to refine or replace the component mathematical models and extend the modelled process; assure a great potential. To harness this potential it is suggested that future work can be initiated in the following three main directions:

6.2.1 GASFLO - as a stand-alone package:

It can be readily used for operator training and analysis of pellet induration systems. It reads input from data files which are to be created by the user, who is presumed to have knowledge of the system and network configuration strategy used by GASFLO. The addition of following features would make it an ideal, user friendly and more productive tool.

- Graphical User Interface (GUI) - having all WIMP (window, icon, mouse and pull down menus) attributes, providing facilities to the user to; compose the network by dragging and placing the component icons of his choice on the board; enter required data through forms and its validation; connect and compute the network by clicking on icons or through pull down menus. Similarly display and hardcopy of input data and computed output in the user familiar graphical format. A platform like Microsoft Windows 3.1 or later could be an ideal environment for such GUI.

- Extension of modelled process (airflow) to include 2-phase flow computation. In fact, the drying and pre-heat zones have noticeable water vapour content, so for more realistic results and for thermal efficiency, so instead of considering it as single phase flow the model should treat it as two-phase flow.

- Generalisation of the component (mathematical) models to simulate: mine ventilation, hydraulic, natural gas or any other fluid flow networks' components. Since all these

use Kirchhoff's law at higher level, so only code relating to the individual components (say compressors in case of natural gas or pumps for water networks instead of presently modelled fans) i.e. at all lower level, would be required to be added.

- The facility to link and execute from user-written linkable modules, and keeping the higher level computation related modules as a dynamic link library (DLL), would excel its power as workbench.

- Addition of optimization module to enhance its usage as a tool for design, operation and planning purposes.

- Extension to transient simulation and provision for the computation of species concentration, could enable it to simulate more challenging projects like propagation of fires in mine ventilation networks. The interrogatable nature of system components can keep the computational loads to minimum e.g. only those paths or parts of the network where smoke has reached can compute the fire model, in addition to the airflow model.

6.2.2 For process control :

The present state and its computational speed instigates that it can be used for process control. As a supervisory control system it could receive the control variables from telemetry data, predict the results verify that they are within their valid ranges, if not then suggest the user of required settings of controllable components. Later it could be upgraded to fully automated process based control system, where to avoid human related input errors the system could actuate the controllable devices itself. Such state of plant operation will make the induration process very efficient, improving the pellet quality by precise control, more effective use of man-power, saving in fuel costs by less burning and hence more environment friendly.

6.2.3 Integration with other related software tools :

It will depend on the nature and extent to which this integration is sought. It will require that the other tools and GASFLO use the same network configuration strategy, and should communicate to each other implicitly. However, this will require re-writing of some portions of the codes of these packages to fulfil this homogeneity. Otherwise a communication layer could be written which could import and export data from GASFLO to the format required by these packages.

It is suggested that wherever possible the computational core of GASFLO i.e. the higher level computation should not be disturbed, and all other facilities like any of the above mentioned, should be bolted on the top of that. The GASFLO architecture permits that elegantly. It will keep the variation to minimum and hence lower costs for the maintenance of its further versions.

References

Afzal M (1991)
Evaluation of Gas Flow Distribution in Process Analysis
Dissertation for MSc in Scientific & Engineering software technology,
Thames Polytechnic, (presently Greenwich University), London

Afzal M and Cross M (1992)
"Mathematical modelling of the gas flow distribution in iron ore pellet induration systems"
Proc of 10th Process Technology Division Conf,
April 5-8, 1992, Tronoto, Published by AIME;
Vol 10, pp 367-374

Ammers E W and Kramer M R (1993)
"The CLiP style of literate programming"
ftp from ammers@rcl.wau.nl

Ahuja R K, Magnanti T K and Orlin J B (1993)
Network Flows - Theory, Algorithms and Applications
Printice Hall Englewood Cliffs, New Jersey

Angell I O and Griffith G (1989)
High-resolution Computer Graphics Using FORTRAN 77
Macmillan

ASCENT (1993)
ASCENT: Automated Strict Case Environment at Teesside
Documentation version 2
University of Teesside, UK

Ashworth C and Goodland M (1990)
SSADM: A Practical Approach
McGraw-Hill Book Co

Azbel D S and Cheremisinoff N P (1983)
Fluid Mechanics and Unit Operations
Ann Orbor Science

Babrow D G (1984)
"Qualitative reasoning about physical systems: An introduction"
Artificial Intelligence, V 14, pp 1-5

Barber G and Hay A (1993)
"EPC compilers on Unix"
Proceedings of UNICOM seminar on 'FORTRAN and C in Scientific Computation', Brunel University, UK, 9-10 June 1993, pp 88-112

Barker H A, Grant P W, Jobling C P and Townsend P (1993)
"The object oriented paradigm: A mean for revolutionizing software development"
Computing & Control Engineering Journal, V 4, no 1, pp 10-14, Feb 1993

Batey E H, Courts H R and Hannah K W (1961)
"Dynamic approach to gas pipeline analysis"
The Oil and Gas Journal, pp 65-78, Dec 18 1961 issue

Bentley J (1987)
"Programming pearls - the Furbelow memorandum"
Communications of the ACM, V 30, n 12, pp 998-999, Dec 1987

Bhave P R (1986)
"Unknown pipe characteristics in Hardy Cross method of network analysis"
J. Indian Water Works Assoc, V 18, n 2, pp 133-135

Bhave P R (1990)
"Rules for solvability of pipe networks"
J. Indian Water Works Assoc, V 22, n 1, pp 7-10

Bhave P R (1991)
Analysis of FLOW in WATER DISTRIBUTION NETWORKS
Technomic Publishing Co, Lancaster, Pennsylvania US

Bird R B, Stewart W E and Lightfoot E N (1960)
Transport Phenomena
International Edition, John Wiley & Sons

Boehm B (1981)
Software Engineering Economics
Printice-Hall, Englewood Cliffs, New Jersey

Boghosian B M (1990)
"Computational physics on the connection machine : Massive parallelism - a new paradigm"
Computers in Physics, pp 14-33, Jan-Feb 1990

Bogle I D L and Pantelides C C (1988)
"Sparse non-linear systems in chemical process simulation"
In SIMULATION AND OPTIMIZATION OF LARGE SYSTEMS;
Osiadacz A J (Ed), *Oxford University Press, Oxford, pp 245-261*

Boulos P F (1989)
Explicit Determination of System Parameters for Upgrading and Enhancing Water Distribution Systems
PhD Thesis, Civil Engineering Department,
University of Kentucky, Lexington, KY 40506

Boulos P and Altman T (1991)
"A graph-theoretic approach to explicit non-linear pipe network optimization"
Journal of Applied Mathematical Modelling, V 15, n 9, pp 459-466

Boulos P F and Altman T (1993)
"An explicit approach for modelling closed pipes in water networks"
Applied Mathematical Modelling, V 17, n 8, pp 437-443

Boulos P, Altman T and Sadhal K (1992)
"Computer modelling of water quality in large multiple source networks"
Applied Mathematical Modelling, V 16, n 8, pp 439-445

Boulos P F and Ormsbee L (1991)
"A comprehensive algorithm for network calibration"
18th Annual Water Resource Conference, New Orleans, pp 949-953

Boulos P F and Wood D J (1990)
"Explicit calculation of pipe network parameters"
J of Hydraulic Engineering, Proc of ASCE, v 116, n 11, pp 1329-1345

Boulos P F and Wood D J (1991)
"An explicit algorithm for calculating operating parameters for water networks"
Civil Engineering Systems, V 8, pp 115-122

Boxer G (1988)
Work Out Fluid Mechanics
Macmillan Education

Boyne G C (1970)
The Design and Analysis of Gas Distribution Networks
PhD Thesis, Dept of Civil Engineering, Heriot-Watt University, Edinburgh, UK

Boyson H F (1993)
"Renormalization group theory based turbulence models and their application to industrial problems"
Proc of European Conf on Engineering Applications of CFD; Sept 7-8, 1993, IMechE, London; Paper no C461/035/93; pp 43-47

Brereton R G (1993)
"Object oriented programming for personal computers"
Chemometerics Intelligent Laboratory Systems, V 19, pp 127-127

Bruce W E and Koenning T H (1987)
"Computer modelling of underground coal mine ventilation circuits: Selection and application of airway resistance valves"
Proc of 3rd Mine Ventilation Symposium; Oct 12-14 1987, Pennsylvania; Mutmanski J M (Ed); *Society of Mining Engineers; pp 519-525*

Burden R L and Faires J D (1989)
Numerical Analysis
4th Edition, PWS-KENT Publishing Company, Boston

Butler N C (1982)
"Pipeline leak detection techniques"
Pipes and Pipelines International, pp 24-29, April 1982

Butler G F and Corbin M J (1989)
"Object oriented simulation in Fortran 77"
REA working paper MM 38/89; 20 Pages, Nov 1989

Chandra S, Blockey D I and Woodman N J (1992)
"An interacting object physical process model"
Computing Systems in Engineering, V 3, no 6, pp 661-670, Dec 1992

Chansler J M and Rowe D R (1990)
"Microcomputer analysis of pipe networks"
Water/Engineering & Management, V 137, no 7, pp 36,38,39; July 1990

Chen PP (1976)
"The entity relationship model - towards a unified view of data"
ACM Transactions on Database Systems, V 1, no 1, pp 9-36, Mar 1976

Coad P and Yourdon E (1991)
Object Oriented Design
Printice-Hall, Englewood Cliffs, New Jersey

Colbrook A and Smyth C (1990)
"Formal specification of data abstraction in Fortran 77: Abstract arrays"
Software Engineering Journal, V 5, no 3, pp 151-159

Collins W R and Miller K W (1991)
"Defining and implementing FORTRAN generic abstract data types"
Information and Software Technology, V 33, no 4, pp 281-291, May 1991

Corbin M J, Birkett P R and Crush D F (1993)
"Multi-Sim: A distributed object-based simulation environment in Ada"
Proceedings of European Simulation Symposium ESS'93, Delft, pp 405-410

Corbin M J and Butler G F (1989)
"Object oriented simulation is FORTRAN"
Proceedings of Society of Computer Simulation Conference, Tempa, Florida, 28-31 Mar 1989

Corbin M J and Butler G F (1990)
"Object oriented simulation in Fortran and Ada"
Proceedings of 1990 UKSC Conference on Computer Simulation, Brighton, Uk, pp 63-68

Coulbeck B and Orr C H (1990)
"Computer-aided analysis, design and operation of water distribution systems versus power distribution systems"
Proceedings of 25th, University Power Engineering Conference, Aberdeen, 1990, pp 465-468

Cox B J (1986)
Object Oriented Programming: An Evolutionary Approach
Addison Wesley, Reading MA

Cox B J (1990)
"There is a Silver Bullet"
BYTE, V 15, no 10, pp 209,210,212,214,216 & 218, Oct 1990

Crosier R (1991)
"FORTRAN programming techniques"
Journal of Quality Technology, V 23, no 4, pp 348-354, Oct 1991

Cross M (1988)
'INDSYS - Iron Ore Pelletizing INDuration SYstem Simulation' User Guide for MS-DOS version
Computational Software Ltd, Surrey, UK

Cross M, Bogren E C, Wakmen J S and Frans R D (1982)
"Mathematical Models of Iron Ore Pellet Induration - Validation and Application"
Proc of 3rd Process Technology Conference, American Institute for Mining, Metallurgical and Protect Engineers, pp -

Cross M and Englund D (1987)
"Assessment of Iron Ore Induration System using Computer Simulation"
Mathematical Modelling of Material Processing Operations, Metallurgical Society Inc.

Cross M, Patel M K and Afzal M (1991)
"Computer simulation of pellet induration for process optimization and control"
Proc of 52nd Annual Mining Symposium, Jan 16-17, 1991; Duluth Minnisota; Published by SMME(AIME); pp 353-365

Cross M, Patel M K and Wade K C (1990)
"Analysis of the gas flow and heat distribution in iron ore pellet induration systems"
In *Control'90: Minerals and Metallurgical Processing;* Rajamain R K and Herbert J A (eds);*Published by SME-AIME, pp 99-108*

Cross M and Wade K C (1989)
"Computer simulation of iron ore pellet induration with additives"
ICHEME-5th International Symposium on Agglomeration; pp 291-298

Cross M and Young R W (1976)
"Mathematical model of rotary kilns used in the production of iron ore pellets"
Ironmaking and Steelmaking, no.3; pp 129-137

Dahl O J and Nygaard K (1966)
"SIMULA - An ALGOL based simulation language"
Communications of the ACM, V 9, no 9, pp 671-678

D'albrand N, Be'gis D, Chavant G and Gunther J (1988)
"Validation of measurements used to solve ventilation problems"
4th Int Mine Ventilation Congress, Brisbane, Queensland, July 1988, pp 133-139

Daniel P T (1966)
"The analysis of compressible and incompressible fluid networks"
Trans of Inst of Chem Engrs; V 44, pp T77-T84

Daugherty R L, Franzini J B and Finnemore E J (1985)
Fluid Mechanics with Engineering Applications
McGraw Hill Book Co pp ix-xi, 28-9, 253-6

DeKleer J (1084)
"How circuits work"
Artificial Intelligence, V 24, pp 205,280

DeMarco T (1978)
Structured Analysis and System Specification
Printice-Hall, New York

Denshyar H (1976)
One Dimensional Compressible Flows
Pergamon Press N.Y.

Deo N (1974)
Graph Theory with Applications to Engineering and Computer Science
Printice Hall

Douglas J F, Gasiorek J M and Swaffield J A (1985)
Fluid Mechanics
2nd Ed., Longman Scientific and Technical Series

Due R T (1993)
"Object oriented technology - The economics of a new paradigm"
Information Systems Management, V , no , pp 69-77, Summer 1993

Duff C and Howard B (1990)
"Migration patterns"
BYTE, V 15, no 10, pp 223,224,226-228,230 & 232, Oct 1990

Duff I S, Erisman A M and Reid J K (1990)
Direct Methods for Sparse Matrices
Oxford Science Publications

Ellis M and Stroustrup B (1990)
The Annotated C++ Reference Manual
Addison Wesley

Ellis D W, Worall K E and Miller S P (1987)
"The computer control of pressures in distribution networks - British gas/Wales"
The Institute of Gas Engineers, Communication 1354, 27 pages, Nov 1987

Ellison A (1993)
"Modelling, philosophy and limitations"
Computing & Control Engineering Journal, V 4, no 4, pp 190-192, Aug 1993

Elmasri R and Navethe S B (1989)
Fundamentals of Database Systems
World Student Series, Addison Wesley
The Benjamin/Cummings Publishing Co, California

Epp R and Fowler A G (1970)
"Efficient code for steady state flows in networks"
Journal of Hydraulics Division, Proc of ASCE, V 96, n HY1, pp 43-56

Ergun S (1952)
"Fluid flow through packed columns"
Chem Eng Prog, v 48, pp 89-94

Fairley R (1985)
Software Engineering Concepts
McGraw-Hill, New York

Fenech K, Cross M and Voller V R (1987)
"Numerical modelling of the cohesive zone formulation in the iron blast furnace"
PCH - PhysicoChemical Hydrodynamics, V 9, n 1/2, pp 71-83

Fertuck L (1992)
System Analysis and Design - with CASE Tools
Wm. C. Brown Publishers, Cubuque, IA 52001, USA

Filho J S R A and Devloo P R B (1991)
"Object oriented programming in scientific computation: The beginning of new ERA"
Engineering Computation, V 8, no , pp 81-87,

Fincham A E (1971)
"A review of computer programs for network analysis (developed at London Research Station)"
The Gas Council Research Communication no CG189, London

Fincham A E and Goldwater M H (1979)
"Simulation models for gas transmission networks"
Transactions of Institute of Measurement and Control, v 1, n 1, pp 3-13, Jan-Mar 1979

Fincham A E and Goodwin N H (1988)
"Methods for gas network simulation"
In: Osiadacz A J (Ed) *Simulation and Optimization of Large Systems*; *Oxford University Press, pp 209-227*

Fitzsimons C J and Greenough C (1993)
"A programming guide for the development of engineering application software in FORTRAN"
Report, Mathematical Software Group, RAL, 27 pages, Jan 1993

Francis J R D (1975)
FLUID MECHANICS for Engineering Students
Edward Arnold

Francis R F (1982)
"The efficient management of the bulk transmission of gas"
Gas Engineering and Management, V 22, pp 123-133

Gane C and Sarson T (1979)
Structured Systems Analysis; Tools and Techniques
Printice-Hall, New York

Gerald C F and Wheatley P O (1989)
APPLIED NUMERICAL ANALYSIS
4th Edition, *Addison-Wesley Pub Co (World Students Series)*

Gibson E (1990)
"Objects - born and bred"
BYTE, V 15, no 10, pp 245,246,248,250,252 & 254, Oct 1990

Goldwater M H and Fincham A E (1981)
"Modelling of gas supply systems"
In: Nicholson H (Ed) *Modelling of Dynamical Systems*; *Vol-2, Peter Pereginus Ltd, pp 150-177*

Goldwater M H, Rogers K and Turnbull D K (1976)
"The PAN Network Analysis Program - its development and use"
Institute of Gas Engineers, Communication 1009, London

Gomasta S K and Devi R (1989)
"Analysis and optimization of pipe networks"
Proceedings of Conference on Engineering Software, New Delhi, India, Norsa Publishing House, pp 383-389

Hall C J (1987)
"Work/lost work, Fan/system: Characteristic curves"
Proc of 3rd Mine Ventilation Symposium; Oct 12-14 1987, Pennsylvania; Mutmanski J M (Ed); *Society of Mining Engineers; pp 418-430*

Hamam Y M and Brameller A (1971)
"Hybrid method for the solution of piping networks"
Proceedings of IEE, V 118, n 11, pp 1607-1612

Hansen C T (1988)
Optimization of Large Networks for Natural Gas
PhD Thesis; Institute for Numerical Analysis, The Technical University of Denmark, Lyngby, DK 2800

Hansen C T, Madsen K and Nielsen H B (1991)
"Optimization of Pipe Networks"
Mathematical Programming, V 52, n 1, pp 45-58

Hardy C (1936) [Referred by Fincham 1971, Jeppson 1976 and others]
"Analysis of flow in networks of conduits and conductors"
Experimental Station Bulletin no 286, University of Illinois

Henderson-Sellers B and Edward J M (1993)
"The fountain model for object oriented system development"
Object Magazine, V , no , pp 71-74,76 & 79, July-Aug 1993

Holloway S (1991)
Choosing CASE Tools
DCF, Information Management Consultancy Ltd, Surrey, U K; 35+ pages

IDE (1992)
Software Through Pictures - Introduction to StP Integrated Structured Environment
Release 4.2D
Interactive Development Environments (IDE), Calafornia 94105

Ingham D B, Heggs P J and Hildyard M L (1988)
"The evaluation of pressure drop across a filter using the boundary element method"
Mathematical Engineering in Industry, vol 2, no 1, pp 1-18

Intersolve (1992)
Refernce Guide - Excelerator Windows Ver 1.0
Intersolve Inc; Rockville, Maryland 20852

Isner J F (1982)
"A Fortran programming methodology based on data abstraction"
Communications of the ACM, V 25, no 10, pp 686-697

Jackson M A (1975)
Principles of Program Design
Academic Press, London

Jackson M (1983)
System Development
Printice-Hall International

Jacky J P and Kalet I J (1987)
"An object oriented programming discipline for standard Pascal"
Communications of the ACM, V 30, no 9, pp 772-776

Jeffrey A (1971)
Mathematics for Engineers and Scientists
Thomas Nelson Ltd (London)

Jeppson R W (1976)
Analysis of Flow in Pipe Networks
Ann Arbor Science, MI 48106

Jones D (1993)
"The C standard and its continuing evolution"
Proceedings of UNICOM seminar on 'FORTRAN and C in Scientific Computation', Brunel University, UK, 9-10 June 1993, pp 214-220

Jones W P (1993a)
"Turbulence Modelling"
Keynote Address to European Conference on Engineering Applications of CFD, IMechE, London, 7-8 Sept 1993

Kiuchi T (1991)
"Calculation of steady state flows in pipeline networks by means of the node admittance matrix" -- in Japanese
Nippon Kikai Gakkai Ronbunshu B hen, V 57, n 540, pp 2784-2790

Knight B (1983)
"A mathematical basis for entity analysis"
In Entity-Relationship Approach to Software Engineering;
Davis C G, Jojodia S, Ng P A and Yeh R T (Eds.);
Elsevier Science Publishers B. V. (North-Holland), pp 81-90

Knight B and Petridis M (1992)
"FLOWES: An intelligent computational fluid dynamics system"
Engineering Application of Artificial Intelligence, V 5, n 1, pp 51-58

Knuth D E (1973a)
The Art of Computer Programming: Vol - 1 Fundamental Algorithms
2nd Edition, Addison Wesley, Reading

Knuth D E (1973b)
The Art of Computer Programming: Vol - 3 Sorting and Searching
Addison Wesley, Reading

Knuth D E (1984)
"Literate Programming"
Computer Journal, V 27, no 2, pp 97-111

Knuth D E (1989)
"The errors of T_EX"
Software Practice & Experience, V 19, no 7, pp 607-685, July 1989

Kóhler W, Walcher M and Kastner W (1990)
"SIMULATION of two-phase flow in pipe networks"
Proc of 1990 European Simulation Multi-Conference Modelling & Simulation, pp 503-507

Lee W, Chris-Tewen J H and Rudd D F (1966)
"Design variable selection to simplify process calculations"
AIChE Jr, V 12, n 6, pp 1105-1110

Lejter M, Meyers S and Reiss S (1992)
"Support for maintaining object oriented programs"
IEEE Transactions of Software Engineering, V 18, no 12, pp 1045-1052, Dec 1992

Levy S (1993)
"Literate programming and CWEB"
Computer Languages, pp 67-68,70, Jan 1993

Lilly S (1993)
"Is object programming harmful?"
Object Magazine, V , no , pp 68-70, July-Aug 1993

Lipworth A D, Walker A J and Annegarn H J (1991)
"FORTRAN package renewal using object-centred design techniques"
The Transactions of the SA institute of Electrical Engineering, V 82, no 1, pp 43-51, Mar 1991

Livny M and Melman M (1982)
"A package for network simulation"
Proc of 5th Biennial Conference of Simulation Society of Australia; University of New England, Armidale, 10-11th May 1982, pp 10-15

Lougher R and Rodden T (1993)
"Group support for the recording and sharing of maintenance rational"
Software Engineering Journal, V 8, no 6, pp 295-306, Nov 1993

Lowndes I S and Weimin H (1988)
"The application of optimization methods to mine ventilation planning"
University of Nottingham, Mining Department Magzin, XL, pp 39-47

Lugt H J (1983)
Vortex Flow in Nature and Technology
John Wiley & Sons, pp 119-126

Mannings R (1889) [Cited by Yen B C 1992]
"On the flow of water in open channels and pipes"
Trans of Inst of Civil Engineers of Ireland, v 20, pp 161-207

Mannings R (1895) [Cited by Yen B C 1992]
"On the flow of water in open channels; Supplement"
Trans of Inst of Civil Engineers of Ireland, v 24, pp 179-207

Marquardt W, Holl P and Gilles E D (1987)
"Dynamic process flowsheet simulation - an important tool in process control"
Proc of International Federation of Automatic Control, IFAC'87, V 2, pp 374-379

Martin J and McClure C (1985) [ref by Fertuck for HIPO charts notation]
Diagramming Techniques for Analysts and Programmers
Printice-Hall

Massey B S (1972)
Mechanics of Fluids
Van Nostrand Reinhold

McDermid D C (1990)
Software Engineering for Information Systems
Blackwell Scientific Publications

McGee W C (1976)
"On user criteria for data model evaluation"
ACM Transactions on Data Base Systems, V 1, no 4, pp 370-387, Dec 1976

McRae G J (1990)
"Chemical process modelling and simulation using advanced computational architectures"
Proc of 3rd International Conf on Foundations of Computer-Aided Process Design; Siirola J J, Grossmann I E and Stephanopoulus G (Eds); held at Snowmass, Colorado, 10-14 July 1989; Published by CACHE and Elsevier (1990)

Metcalf M (1985)
Effective Fortran 77
Clarendon Press Oxford

Metcalf M and Reid J (1990)
Fortran 90 Explained
Oxford University Press

Meyer B (1988)
Object Oriented Software Construction
Printice Hall, New York

Moll A T J and Lowndes I S (1992)
"Graph theory applied to mine ventilation analysis"
IMA Bulletin, V 28, n 6/7/8, pp 103-106

Montagna J M and Iribarren O A (1988a)
"Optimal resolution sequence of problems modeled by directed graphs"
Mathematical Computer Modelling, V 10, n 7, pp 515-521

Montagna J M and Iribarren O A (1988b)
"Optimal computation sequence in the simulation of chemical plants"
Computer & Chemical Engineering, V 12, n 1, pp 71-79

Moses J and Jackson K (1991)
"Ensuring robustness and reliability of object oriented software using MASCOT 3"
In *Reliability and Robustness of Engineering Software II;*
Eds Brebbia C R and Ferrante A J; *Elsevier, pp 19-34*

Motard R L and Westerberg A W (1981)
"Exclusive tear sets for flowsheets"
AIChE Journal, V 27, n 5, pp 725-732

Mucharam L and Adewumi M A (1990)
"A compositional two-phase flow model for analysing and designing complex pipeline network systems"
Proc of CIM/SPE (Society of Petroleum Engineers) Technical Meeting, Calgary, Canada, pp (18)1-(18)16

Nielsen H B (1989)
"Methods for analyzing pipe networks"
Journal of Hydraulic Engineering, v 115, n 2, pp 139-157, (Feb 1989)

Ormsbee L E and Wood D J (1986)
"Explicit pipe network calibration"
J Water Resources, Planning and Management, Proc of ASCE, v 112, n 2, pp 166-182

Orr and Ken (1987) [ref by Fertuck 1992 for Warnier-Orr SCs notation]
Structured Requirements Definitions
Ken Orr and Associates Inc

Osiadacz A J (1987)
Simulation and Analysis of GAS Networks
E & F N Spon, London

Osiadacz A J (1988)
"Method of steady state simulation of a gas network"
Int Journal of Systems Science, V 19, n 11, pp 2395-2405

Osiadacz A J and Pienkosz K (1988)
"Methods of steady state simulation for gas networks"
Int Journal of Systems Science, V 19, n 7, pp 1311-1321

Osiadacz A J and Salimi M A (1988a)
"Comparison between sequential and hierarchical simulation of gas networks:
Part I: Dynamic simulation of gas flow in single pipe"
Information and Decision Technologies, V 14, pp 77-98

Osiadacz A J and Salimi M A (1988b)
"Comparison between sequential and hierarchical simulation of gas networks: Part II: Dynamic simulation of gas flow in networks"
Information and Decision Technologies, V 14, pp 99-123

Osinski E J, Barr P V and Brimacombe J K (1989)
"Mathematical model for gas flow through a packed bed in the presence of sources and sinks"
The Canadian Journal of Chemical Engineering, V 67, pp 722-730

Parnas D L (1972)
"A technique for software module specification with examples"
Communications of the ACM, V 15, no 5, pp 330-336, May 1972

Parnas D L (1972a)
"On the criteria to be used in decomposing system into modules"
Communications of the ACM, V 15, no 12, pp 1053-1058, Dec 1972

Patel M K and Cross M (1989)
"The modelling of fluidized beds for ore reduction"
Proc of 6th Conf on Numerical Methods in Lamiar and Turbulent flows; Taylor C, Gresho P, Sani R L and Hauser J (Eds);*held at Swansea U.K., June 11-15 1989; Peneridge Press; pp 2051-2068*

Patel M K, Pericleous K and Cross M (1993)
"Numerical Modelling of Circulating fluidized beds"
Computational Fluid Dynamics, V 1, pp 161-176

Perkins J D, Barton G W, Chan W-K and Howell J M
"The use of SPEEDUP simulation package for process operability analysis"
Proc of Int Federation of Automatic Control; IFAC'87, V 2, pp 380-385

Petridis M, Knight B and Edward D (1991)
"A design for reliable CFD software"
In *Reliability and Robustness of Engineering Software II;* Eds Brebbia C A and Ferrante A J; *Elsevier, pp 3-17*

Petley B W (1991)
"Fine tuning the SI units and fundamental physical constants"
Proc of Royal Society of London, Series A, vol 433, pp 219-233

Pho T K and Lapidus L (1973)
"Topics in computer-aided design: Part I - A optimum tearing algorithm for recycle systems"
AIChE Jr, V 19, n 6, pp 1170-1181

Pressman R S (1988)
Software Engineering - A Practitioners Approach
McGraw-Hill Book Co

Raphael B and Krishnamoorthy C S (1993)
"Automating finite element development using object oriented technology"
Engineering Computation; V 10, no , pp 267-278

Reid J K (1988)
"Using FORTRAN 8x to solve large problems"
In *Simulation and Optimization of Large Systems;*
Osiadacz A J (Ed), Clarendon Press Oxford, pp 161-173

Reid J (1992)
"The advantages of FORTRAN 90"
Computing, V 48, no 3-4, pp 219-238

Rine D (1993)
"Object oriented technology and software reuse"
Computer (IEEE), Vol , no , pp 6-6, July 1993

Rizman K and Rozman I (1993)
"Facilitating composition and increasing object reusability by means of an event-driven object oriented development"
Microprocessing and Microprogramming, no 37, pp 111-114

Rose E (1981)
"Ironmaking and Steelmaking - I"
In: Nicholson H (Ed) *Modelling of Dynamical Systems vol-2*
Peter Peregrinus Ltd

Sargent R W H (1978)
"The decomposition of systems of procedures and algebraic equations"
In: Watson G A (Ed), *Proc of Biennial Conference, Dundee 1977,*
(Lecture notes in Mathematics # 630 - Numerical Analysis)
Published by *Springer - Verlag*

Smith J P (1988)
Digital Records - Utility Applications
The Institute of Gas Engineers; Communication 1362, pages 18

Smith W A (1979)
ELEMENTARY NUMERICAL ANALYSIS
Harper and Row Publishers

Sommerville I (1989)
Software Engineering
3rd Edition, Addison Wesley

Shum S and Cook C (1993)
"AOPS: an abstraction oriented programming system for literate programming"
Software Engineering Journal, V 8, no 3, pp 113-120, May 1993

Syslo M, Deo N and Kowalik J S (1983)
Discrete Optimization Algorithms with PASCAL Programs
Printice Hall

Sissom L E and Pitts D R (1972)
Elements of Transport Phenomena
McGraw Hill Book Co pp 12-13

Tarjan R E (1983)
Data Structures and Network Algorithms
CBMS-NSF, Regional Conf Series in Applied Maths; SIAM

Tennent R M (Ed) (1971)
Science Data Book
Olier & Boyde Publishers

Theodor L (1971)
Transport Phenomena for Engineers
International Text Book Co

Thimbleby H (1993)
"A personal view: software mechanics"
Software Engineering Journal, V 8, no 3, pp 110-111, May 1993

Travers K (1967)
"The mesh method in gas network analysis"
Gas Journal, V 332, pp 167-174

Turner W J, Bakker N A and Severs M (1982)
"Simulation of natural gas pipeline networks"
Proc of 5th Biennial Conf of Simulation Society of Australia, University of New England, Armidale, 10-11 May 1982, pp 154-158

Turner W J, Kwon P S-J and Maguire P A (1991)
"Evaluation of a gas pipeline simulation program"
Mathematical and Computer Modelling, v 15, n 7, pp 1-14

Turner W J and Mudford N R (1988)
"Leak detection, timing, location and sizing in gas pipeline"
Mathematical Computer Modelling, V 10, n 8, pp 609-627

Turner W J and Rainbow M J (1983)
"NAIAD - A package for modelling flow networks and heat transport systems"
Proc of Conf on Computers and Engineering, Sydney, 31st Aug - 2 Sept 1983, pp 127 - 131

Turner W J and Simonson M J (1985)
"Compressor station transient flow modelled"
Oil and Gas Journal, V 83, n 20, pp 79-83

Usman A, Powell R S and Sterling M J H (1987)
"Comparison of Colebrook-White & Hazen-Williams flow models in real-time water network simulation"
Computer Applications in Water Supply: Vol-1, System Analysis & Simulation; Coulbeck B and Orr C H (Eds); pp , John Wiley & Sons

Voyles C F and Wilke H R (1962)
"Selection of circuit arrangements for distribution network analysis by the Hardy Cross method"
Journal of American Water Works Association, V 54, n 3, pp 285-290

Wang Y J (1982)
"Critical path approach to mine ventilation networks with controlled flow"
Transactions of Society of Mining Engineers of AIME, V 272, pp 1862-1872

Wang Y J (1990)
"Solving mine ventilation networks with fixed and non-fixed branches"
Mining Engineer, vol 42, no 9, pp 1091-1095; Sept 1990

Wang Y J, Mutmanski J M and Harthan H L (1988)
"Characterizing multiple operating points in mine ventilation systems"
4th International Mine Ventilaton Congress, Brisban, Queensland, July 1988, pp 93-100

Ward Smith A J (1971)
Pressure Losses in Ducted Flows
Butterworth London

Ward T and Bromhead E (1989)
FORTRAN and the Art of PC Programming
John Wiley and Sons

Welch J T (1966)
"A mechanical analysis of the cycle structure of undirected linear graphs"
Jr of Assoc of Computing Machinery, V 13, n 2, pp 205-210

Wiegers K E (1993)
"Implementing software engineering in a small software group"
Computer Languages, V 10, no 6, pp 55-58,60,62,64; June 1993

Wilde N and Huitt (1992)
"Maintenance support for object oriented programs"
IEEE Transactions on Software Engineering, V 18, no 12, pp 1038-1044, Dec 1992

Wilkes M V (1993)
"From FORTRAN and ALGOL to object oriented languages"
Communications of the ACM, V 36, no 7, pp 21-23, July 1993

Wilkinson M K and Byers P J (1993)
"The engineering of complex software systems"
Computing & Control Engineering Journal, V 4, no 4, pp 187-189, Aug 1993

Wilson J G, Mallinson J R and Cheney J E (1986)
"Simulation and optimization of gas transmission systems"
Proc of 1986 International Gas Research Conference, Toronto, pp 373-385

Wilson J G, Wallace J and Fur B P (1988)
"Steady-state optimization of large gas transmission systems"
In: Osiadacz A J (Ed) *Simulation and Optimization of Large Systems*
Oxford University Press, pp 193-207

Wilson R J and Watkins J J (1990)
GRAPHS - An Introductory Approach
John Wiley & Sons

Wimblad A L, Edwards S D and King D R (1990)
Object Oriented Software
Addison Wesley, Reading MA

Wood D J and Charles C O A (1972)
"Hydraulic network analysis using linear theory"
Journal of Hydraulics Division, Proc of ASCE, V 98, n HY7, pp 1157-1170

Wood D J and Rayes A G (1981)
"Reliability of Algorithms for pipe networks"
Journal of Hydraulic Division, ASCE, V 107, n HY10, pp 1145-1161, (Oct 1981)

Yen B C (1992)
"Dimensionally homogeneous Manning formula"
Journal of Hydraulic Engineering, vol 118, no 9, pp 1326-1332

Yevdokimov A G (1969)
"A theory of the solution of steady state network problems with special reference to mine ventilation networks"
Int Journal of Numerical Methods in Engineering, V 1, pp 279-299

Young R W, Cross M and Gibson R D (1979)
"Mathematical model of grate-kiln-cooler process used for induration of iron ore pellets"
Ironmaking and Steelmaking; no 1, pp 1-14
The Metals Society, London Publication

Yourdon E (1990)
"Auld lang syne"
BYTE, V 15, no 10, pp 257,258,260,262 & 264, Oct 1990

Zografos A J, Martin W A and Sunderland J E (1987)
"Equations of properties as a function of temperature of seven fluids"
Computer Methods in Applied Mechanics and Engineering; Vol 61, pp 177-187

Appendix - A
GASFLO Users' Guide

A.1 Introduction

GASFLO is a software tool to determine the airflow distribution in pellet induration system pipe networks. The airflow distribution includes the computation of the flow of process gas in all paths, its pressures at all nodes and its temperature at all nodes and paths of the network.

The manufacture of iron ore pellets is a well established industry and induration process is an important component of this industry. The indurated or processed pellets are used as raw input for the blast furnace process in ironmaking and steelmaking. Air is used as process gas for the induration process and it transfers heat among different stages of the process. The air at ambient temperature is pumped into the system in the cooling stage and passed through the hot burnt pellets to cool them, where the gas extracts heat from the pellets, which is transferred to the drying and heating stages of the system. The working of pellet induration systems is explained in Chapter 2 and by Rose 1981.

The hostile environment restricts the ability to measure all the variables required for the optimization of the induration process. Without a tool such as GASFLO, the concerned staff have to rely on the approximate guessed data which is usually inaccurate and erroneous. Consequently, the optimization strategies lack confidence due to the inherent inaccuracy in

the airflow distributions. Some packages like INDSYS (Cross and Englund 1987) have appeared for the study of heat concentration in the system but they also need to have the airflow distribution defined. Hence, the quality of their results is dependent on the exactness of the fed in airflow distribution.

In the following; section A.2 covers the requirements of GASFLO software tool and installation of the package; section A.3 describes the structure of the tool and input/output data files; section A.4 illustrates the creation of main input data files, the running procedure and the graphical display of the computed results; section A.5 shows how the results from GASFLO are used by INDSYS and vice versa; in the last section A.6 some possible future extensions are discussed.

A.2 GASFLO Requirements

GASFLO is written in standard FORTRAN 77. Presently it works on high end PCs (i.e. 100% IBM compatibles), but later if required it can be ported to other platforms. The hardware and software requirements for GASFLO are :

A.2.1 Hardware Requirements :

The computation of GASFLO requires:

- IBM PC 100% compatible machine with a 386SX or higher processor;
- High resolution colour monitor with VGA or SVGA graphics card;
- 4 MByte of RAM;
- 3 Mbytes of hard disk storage; and
- PostScript printer for printing graphical output of computed results.

A.2.2 Software Requirements :

GASFLO has been developed using FTN77/386 Salford compiler, which enables to exploit the 32 bit capability of these high end machines, but the compiler uses its own run-time library and extended memory manager DBOS to overcome the DOS 640K limitation. The software related requirements are:

- Microsoft DOS 5.0 or later;
- DBOS - the extended memory manager and run-time library. This needs to be pre-loaded to run the programs PRPNET and CMPNET;
- Graphical display of result requires the UNIX like 'awk' and 'sed' utilities to automate the editing of the 'static' PostScript file;
- GhostScript - a public domain PostScript viewer to display of the edited PostScript file on the screen.

A.2.3 Installation of GASFLO :

GASFLO comes on a 3.5" high density floppy and can be installed on hard disk by typing (from A:)

```
A:\> INSTALL2 d:
```

Where d: is the target drive where GASFLO is to be installed.

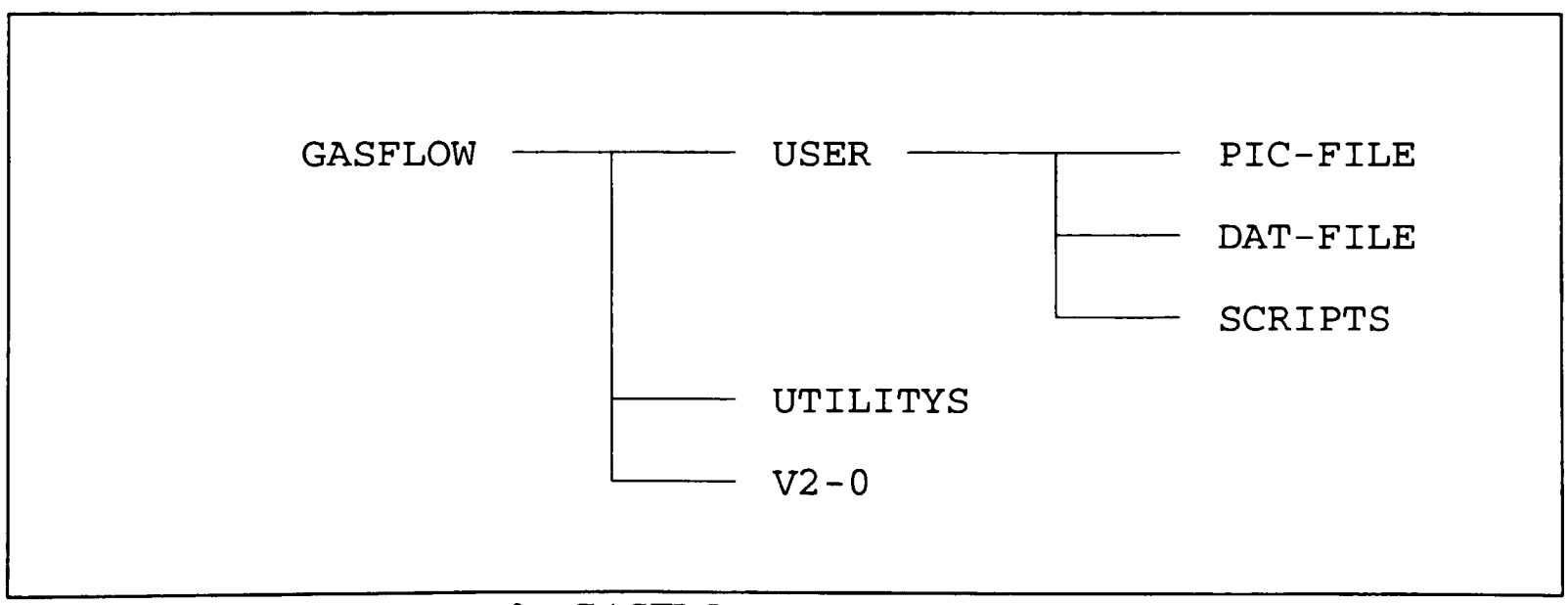

Figure A.1 Directory tree for GASFLO

INSTALL2 unpacks the files into the directory structure shown in Figure A.1. and provides sufficient guidance for changes required to be made in the batch files for successful running of the program. In directory UTILITYS, the files README, README.V20 and GASFLOW.LST respectively give information about the DBOS, gasflow version 2.0 and a list of all files included in the package with brief description of their functions.

INSTALL2 generates the GASFLOW.BAT file taking into account the selected target drive, which when run modifies the PATH to include UTILITYS and V2-0 directories. In some cases when the original PATH is significantly long, the failure of this modification has been noticed. This is due to the DOS limit of 127 characters for PATH string and would require manual adjustment by typing in command like

```
PATH=C:\;C:\DOS;...;d:\GASFLOW\UTILITYS;d:\GASFLOW\V2-0;
```

Where ... represents the other directories of ones choice and d: is the target drive where GASFLOW is installed. The presence of these directories in path is required for efficiency reasons. For example, GhostScript is not compatible with DBOS, so display of results via GhostScript requires the DBOS to be down loaded, and loaded again for the running of PRPNET or CMPNET. The batch files used for viewing results can do this loading and unloading efficiently and invisibly.

A.3 GASFLO Program Structure

GASFLO consists of the following three main parts:

1. Preparation of the Network (PRPNET.EXE),
2. Computation or Simulation of the Network (CMPNET.EXE), and
3. Output of computed results in graphical format (DISPLAY.BAT).

All these three are stand alone programs. Figure A.2 shows their inter-dependence through data files. It shows that the output produced by the program PRPNET i.e. *NETWORK.INF*, being used by CMPNET. Similarly CMPNET produces *DISPLAY.DAT* which is used by DISPLAY. Apart from these two files other input files are to be provided by the user to run the respective programs. Figures A.3a and A.3b show the lists of input and output files for these programs. The functionality of these programs is briefly discussed in the following subsections .

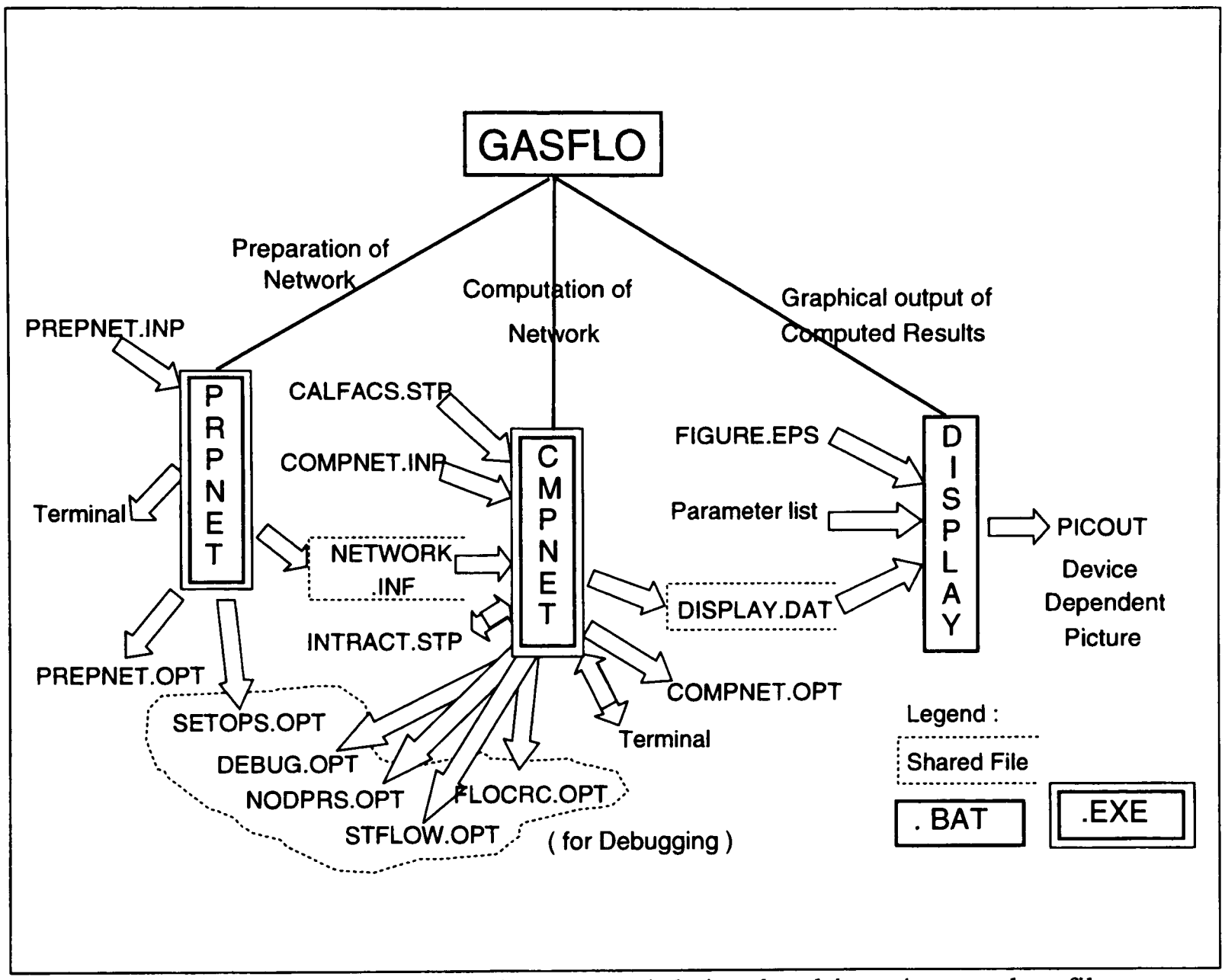

Figure A.2 GASFLO program main parts and their related input/output data files

A.3.1 PRPNET - PRePare NETwork :

Analysis of any induration system requires a significant number of runs of respective system, each varying from the other by only few input parameters. The network structure

remains constant for all these runs, so the information related to the connectivity of network should logically be compiled once and reused time and again until there is some physical change in the network. The program PRPNET is to do this compilation task.

Input and Output Files for PRPNET and CMPNET Components of GASFLO

PRPNET.EXE

Input Files:

PREPNET.INP	Main input file for PRPNET, includes total number of network components and information about node and stream connectivity information

Output Files:

PREPNET.OPT	Output of PREPNET gives information about the network components' connectivity in the linked lists form
NETWORK.INF	The network connectivity information written in the binary CMPNET readable form to avoid errors from direct editing
SETOPS.OPT	Output from the performed SET OPeration routines, while executing tree partitioning algorithms (for debugging)
Terminal	Output on terminal, includes node-node and node-stream incidence matrices, stream composition, status of streams whether belonging to tree or Cotree and overall counters controlling linked list structures. [This can be re-directed to some file using DOS re-direction facility]

CMPNET.EXE

Input Files:

COMPNET.INP	Main input file, containing program controls, process gas, geometrical and property data for each component of the induration system
NETWORK.INF	Network components' connectivity data in linked list format in binary mode, produced by PRPNET
CALFACS.STP	Calibration or efficiency factors for all pipes
INTRACT.STP	Stores the run-time options for the program from previous run of CMPNET, to relieve user from specifying same options and to improve execution time.

Output Files:

COMPNET.OPT	Detailed Output in text form; contains property, geometric and distribution related data for all components. The output on each iteration of the computation can be stored. This file also includes network connectivity information i.e. copy of PREPNET.OPT.
Terminal	The interactive output on screen. At various stages CMPNET also asks input from terminal, so it is input as well as output file.
DISPLAY.DAT	This includes the final computed (Airflow, Pressure and Temperature) distributions of the network for all streams and nodes. This file is being used as input for DISPLAY.BAT.
DEBUG.OPT	Contains information about the relaxation of leaks areas. This is mainly to aid debugging of the program.
NODPRS.OPT	Shows node pressures at selected nodes for each iteration.
STFLOW.OPT	Shows stream flows of selected stream for each iteration.
FLOCRC.OPT	Shows error or flow imbalance at selected nodes for each iteration.

(These three files output in spreadsheet format, so that these could be directly imported and plotted using some package like Lotus 1-2-3. These are to identify the misbehaving network components and track down algorithmic errors)

Figure A.3a The input and output files of programs PRPNET and CMPNET

PRPNET prepares the network information file, *NETWORK.INF*, by reading in the file *PREPNET.INP* which contains the connectivity of all network nodes and streams, and composition of each of these streams. It transforms the network information into the node-stream and node-node incidence matrices and shows them on screen or via *Terminal*. Further these incidence matrices are re-written in linked list structures and output in file *PREPNET.OPT*. For reuse the same information is also written in a FORTRAN direct access file *NETWORK.INF* in binary format, which cannot be manually edited and requires less storage, and it is efficiently read in by CMPNET program. There is an other output file called *SETOPS.OPT* which contains the outputs of different set operations performed by tree partitioning algorithm.

Input and output files for DISPLAY Component of GASFLO

DISPLAY.BAT

Input Files:

FIGURE.EPS	The main Encapsulated PostScript graphic file. Which contains stubs for network components whose computed values are to be displayed
Parameter list	These are the parameters fed in by the user at command line stating what distributions are to be displayed and what is the destination device for generated output
DISPLAY.DAT	This file is output by CMPNET and contains the computed variables and their corresponding stubs. It is written in specific format so that it can be easily manipulated by the 'awk' utility

Output Files:

PICOUT	This is the edited FIGURE.EPS file, with stubs replaced by their corresponding computed values picked-up from DISPLAY.DAT. It is scaled and rotated according to the options specified in parameter list, for specific output device i.e. either for screen usable via GhostScript or for PostScript printer

Figure A.3b Input and output files for DISPLAY component of GASFLO

For debugging and confirmation that correct network is being modelled, the *Terminal* file can be redirected using standard DOS redirection command '>' and printed or viewed on the screen. All these three output files, *Terminal, PREPNET.OPT* and *SETOPS.OPT* provide sufficient information to track down the problems introduced during the preparation of input file or some malfunction of the algorithms.

A.3.2 CMPNET - CoMPute NETwork :

CMPNET is the most dominant part of GASFLO. It reads in *NETWORK.INF*, *INTRACT.STP*, *CALFACS.STP* and *COMPNET.INP* files and computes the airflow, pressure and temperature distributions for the induration system (mostly referred as 'network'). *COMPNET.INP* includes all data related to program controls, process gas and component material and geometric properties, which is required for the computation of respective mathematical models. *CALFACS.STP* contains the calibration factor data for all pipes of the network, these are same as pipe efficiency factor, which should ideally be unity but since the pipe friction factor and other data is not completely known so their values are adjusted to compensate that data. This needs some fine tuning to get region pressures close to physical values. *INTRACT.STP* is a file generated by CMPNET itself, which restores the chosen options for different control parameters at run-time and in subsequent runs this file is read by the CMPNET instead of asking user for these inputs from terminal.

The computed results are output in tabular form in *COMPNET.OPT* and *DISPLAY.DAT*. The former file contains the detailed output relating to all instances of each entity i.e. for every pipe and every fan etc. It can have the variable for each iteration or after any selected step. Whereas *DISPLAY.DAT* file contains the final output relating to fans' end pressures, valve openings, stream flows and temperatures (at exit end), and pressures and temperatures for all nodes. The other difference is that since the *COMPNET.OPT* is mostly used for debugging so it contains the values of system variables in computational units whereas in *DISPLAY.DAT* these are in experimental units.

CMPNET also outputs to *DEBUG.OPT* file which contains the data related to execution and to the *Terminal*. The contents of some other auxiliary output files; *NODPRS.OPT* for node pressures, *STFLOW.OPT* for stream flows and *FLOCRC.OPT* for error in flows at internal nodes; can be selected by the user while running the program. These *NODPRS.OPT, STFLOW.OPT* and *FLOCRC.OPT* files are in spreadsheet compatible format and can be easily imported into some LOTUS 1-2-3 like spreadsheet package to view the values graphically and trace the problematic component. In fact these were the aids for the

development stage and for algorithmic refinements, but still these can be used for continuous monitoring of any node or stream variable.

A.3.3 DISPLAY - Graphical Output of System Variables :

DISPLAY is post-processor of GASFLO. This was developed as an ad hoc facility to display the computed output values on the actual figure. This will be used until a complete Graphical User Interface is developed for GASFLO.

DISPLAY is a batch file which makes use of different public domain utilities to achieve the desired goal. It reads in *DISPLAY.DAT* produced by CMPNET and *FIGURE.EPS* input by the user. *FIGURE.EPS* is usually a schematic of simulated network drawn by some drawing package and exported as Encapsulated PostScript file. It is important that *.EPS* file should contain the same component names (referred as 'stubs') whose values are to be output on the figure, as these are used in *DISPLAY.DAT* along with their computed values.

DISPLAY replaces the stubs in *FIGURE.EPS* by their corresponding values given in *DISPLAY.DAT* and scales the edited drawing i.e. *PICOUT* for the chosen device. Figure A.4 shows the DISPLAY related files and its working will be discussed in Section A.4.3.

A.4 Program Running and Creation of Input Files

The components of the GASFLO program should be run in the specified order. PRPNET produces *NETWORK.INF* file, which is being used by CMPNET, and further, the output of CMPNET i.e. *DISPLAY.DAT* is used by DISPLAY to output the computed results graphically. The sample listings of source of the input and output files for these programs will be given later in Blocks A.1-A.5. In this section we mainly discuss the creation of input files and procedures to run these programs.

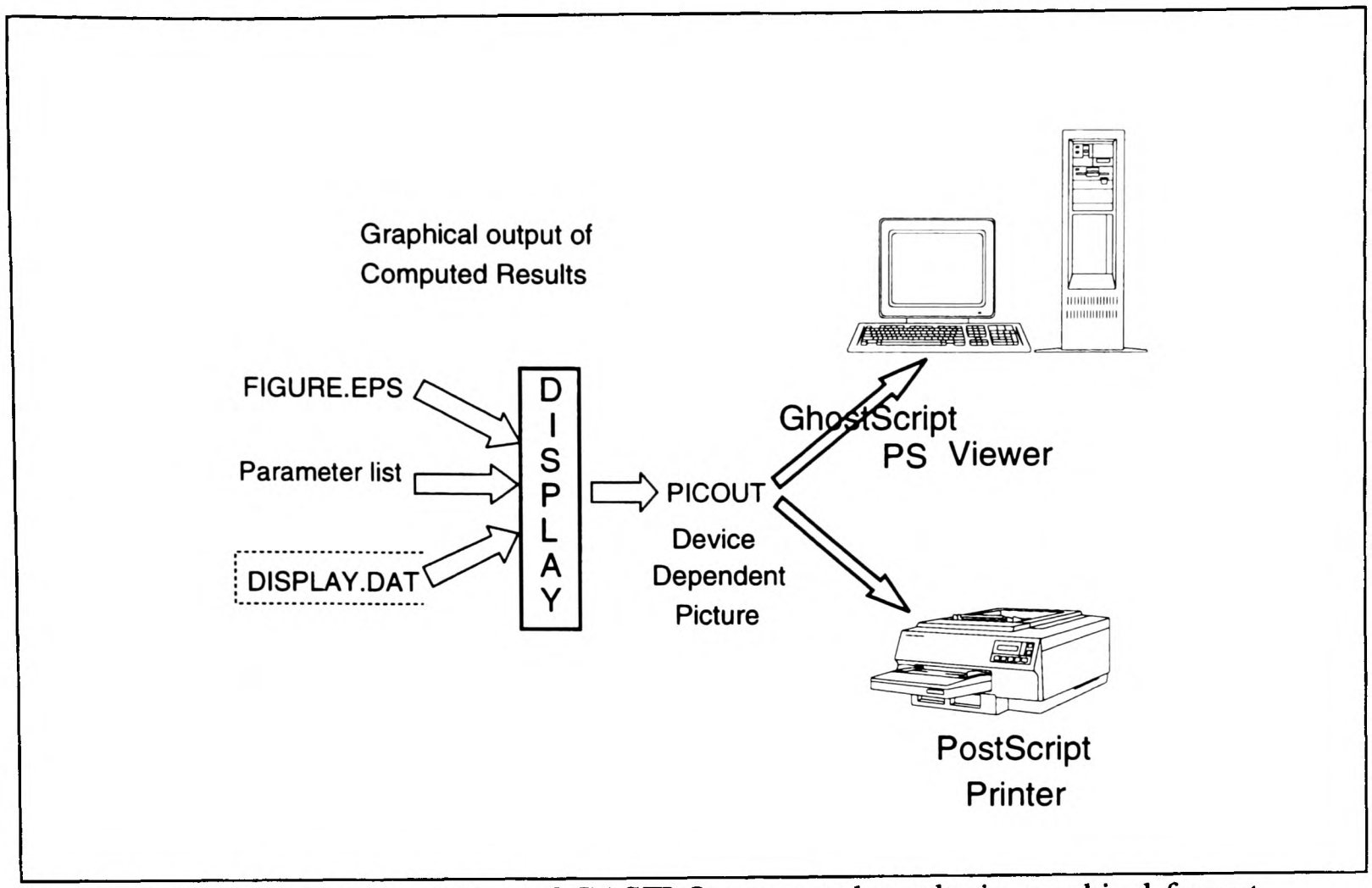

Figure A.4 Display or printing of GASFLO computed results in graphical format

A.4.1 PRPNET - Prepare Network

In general the information provided by the practitioners about the systems to be analyzed is not in the state as it is required by the software tool. PRPNET deals with the connectivity of the network. The transformation of available information to the concrete input data for computer programs needs some basic procedure. In practice, the information about the pellet induration system is usually given as 'plant schematics' like the one shown in Figure 2.3. To create the input data file for PRPNET program (i.e. *PREPNET.INI* file whose partial listing is shown in Block A.1) needs the following steps:

Step-1: Mark all the nodes and components on the schematic,

Step-2: Reduce all non-circular pipes or ducts into their equivalent circular pipes, since the mathematical model caters only circular pipes. The pipes connected to other pipes either in series or in parallel, should be replaced by an equivalent pipe offering the

same resistance to the flow. This reduces the overall size of the network and lessens the computational load. This also promotes connections between different components rather than between pipes,

Step-3: Assign integral identification numbers to each of the components tagged with their entity names. The resulting six character name is treated as name of each of the network components,

Step-4: Identify all streams i.e. the components connected serially and having constant flow through them and note down the composition of each of the streams,

Step-5: Assign a default flow direction to each stream,

Step-6: Allocate an integer identity numbers to each node and each stream. First all nodes be numbered, starting from a source to its sinks following the assigned flow direction, and then repeating the same procedure for subsequent source nodes. The nodes once numbered are skipped on further encounters. A similar approach is adopted for the numbering of all the streams. The separate integer sequences from 1 to **NTNDS** for nodes and 1 to **NTSTR** for streams would result, which are used while referring to the connectivity of the network components.

Now after following these steps, the nodes and streams have been identified and the composition of each of the streams is known. The network can be drawn in graph theoretic form as a connected graph of 'nodes' and 'edges', this will be helpful to visualise the resulting tree and cotree structures which are computed by PRPNET. Now the input data file *PREPNET.INP*, for PRPNET can be created. *PREPNET.INP* consists of the following four types of data:

a. Node and stream related statistics;

NTNDS - no of total nodes, **NTSTR** - no of total streams, **NTSRCS** - no of total sources, and **NTSNKS** - no of total sinks.

b. Data for each of the **NTSTR** streams;

ISTR - stream number, **NDUP** - Upstream end node number, **NDDN** - downstream end node number, **NCMPS** - number of components comprising the stream, and **CMPNAM(1 .. NCMPS)** - component name for each of the components.

c. Data for each of the **NTNDS** nodes;

NODID - node (identification) number, **NDEG** - node degree i.e. number of incident streams on the node, **LSTRM(1..NDEG)** - list of stream numbers incident on node {streams leaving the node must have -ve sign}, and **NODNAM** - a six character string as the name of the node.

d. Data related to each of the **NTSRCS** source nodes;

ISRC - source node Id, **NSNKS** - no of sink nodes it is feeding to, and **LSNKS(1 .. NSNKS)** - list of sink nodes associated to the respective source.

All this data is related to network connectivity. The component (as well as node) names are six character strings and are enclosed in quotes according to FORTRAN data conventions. The first three characters correspond to the entity name i.e. fan, region, bed etc and last two integers determine its instance number in the entity, and these two parts are separated by a '-'. For example 'FAN-01' is the first fan, 'REG-11' is the 11th region and 'BED-06' is the 6th packed bed.

Block A.1 shows the partial listing of the file *PREPNET.INP* for a typical pellet induration system. The first line is read as a comment line and is not read by the program, whereas in subsequent lines the characters following '!' are also comments enabling user to refer what the respective data is referring to, these are also not read by the program.

Block A.1 Listing of *PREPNET.INP* file

```
!This is a PREPNET.INP file; for NETwork CONfiguration - 17th July 93 (Sat)
28,38,2,3  !ntnds,ntstr,ntsrcs, ntsnks
  1,  1,  2,  3,'PIP-01','FAN-01','PIP-02'     !St#,Upnd,Dnnd,Ncmps,(Cmp-names)
  2,  2,  3,  1,'BED-01'
  3,  3,  4,  1,'PIP-03'
  4,  4,  5,  1,'BED-03'
  5,  5,  7,  1,'PIP-07'
  6,  7, 10,  3,'PIP-08','FAN-04','PIP-09'
  7, 10, 11,  1,'PIP-10'
  8, 11, 12,  1,'BED-06'
  9, 12, 13,  2,'PIP-11','FAN-05'
 10,  4,  6,  1,'BED-04'
 11,  6,  8,  1,'PIP-04'
 12,  8,  9,  2,'PIP-05','FAN03'
 13, 24,  7,  1,'PIP-21'
 14, 24,  8,  1,'PIP-20'
 15, 14, 15,  3,'PIP-12','FAN-02','PIP-13'
 16, 15, 18,  1,'PIP-15'
 17, 15, 16,  1,'PIP-14'
 18, 16, 17,  1,'BED-02'
 19, 17, 18,  1,'PIP-16'
 20, 18, 19,  1,'PIP-17'
 21, 19, 24,  1,'PIP-18'
 22, 19, 20,  1,'PIP-19'
 23, 20, 21,  1,'BED-05'
 24, 21, 22,  2,'PIP-22','FAN-06'
 25, 17, 23,  1,'PIP-23'
 26,  9, 10,  1,'PIP-06'
 27, 20,  6,  1,'BED-07'
 28,  2, 16,  1,'LEK-01'
 29,  3, 17,  1,'LEK-02'
 30, 25,  4,  1,'LEK-03'
 31, 26,  5,  1,'LEK-04'
 32,  5,  6,  1,'LEK-05'
 33,  4, 20,  1,'LEK-06'
 34,  6, 21,  1,'LEK-07'
 35, 11, 20,  1,'LEK-08'
 36, 12, 21,  1,'LEK-09'
 37, 27, 11,  1,'LEK-10'
 38, 28, 12,  1,'LEK-11'
  1,  1, -1,'BDY-01'         !Nd#,Ndeg,(Strm#s){-ve GOING OUT},Ndname
  2,  3,  1, -28,  -2,'REG-01'
  3,  3,  2, -29,  -3,'REG-02'
  4,  5,  3,  30, -33,  -4, -10,'REG-05'
  5,  4,  4,  31,  -5,  -32,'REG-06'
  6,  5, 10,  27,  32,  -34, -11,'REG-07'
  7,  3,  5,  13,  -6,'JUN-06'
  8,  3, 11,  14, -12,'JUN-05'
  9,  2, 12, -26,'JUN-08'
 10,  3,  6,  26,  -7,'JUN-07'
 11,  4,  7,  37,  -8, -35,'REG-10'
 12,  4,  8,  38, -36,  -9,'REG-11'
 13,  1,  9,'BDY-05'
 14,  1,-15,'BDY-02'
 15,  3, 15, -17, -16,'JUN-01'
 16,  3, 17,  28, -18,'REG-03'
 17,  4, 18,  29, -19, -25,'REG-04'
 18,  3, 19,  16, -20,'JUN-02'
 19,  3, 20, -22, -21,'JUN-03'
 20,  5, 22,  33,  35, -23, -27,'REG-08'
 21,  4, 23,  34,  36, -24,'REG-09'
 22,  1, 24,'BDY-04'
 23,  1, 25,'BDY-03'
 24,  3, 21, -13, -14,'JUN-04'
 25,  1,-30,'ATM-06'
 26,  1,-31,'ATM-07'
 27,  1,-37,'ATM-08'
 28,  1,-38,'ATM-09'
  1,  1, 13          !isrc,no_of_fed_snks, snk_#s
 14,  3, 23, 22, 13
```

PRPNET can be run by typing in **PRPNET** from command line and it will ask for input filename, which contains node and stream related connectivity information. In response '*PREPNET.INP*' could be specified. This is to enable the user to use the name of his own choice and simulate multiple networks simultaneously. The output produced on the terminal can be redirected to a file say Terminal.out, simply by typing in **PRPNET > Terminal.out** on command line. The output files produced by PRPNET has been briefly described in Figure A.3a. *PREPNET.OPT* the output file containing linked lists for node and stream related connectivity information is included in *COMPNET.OPT* which will be discussed in next Section.

A.4.2 CMPNET - Compute Network :

CMPNET is the main simulation part of the GASFLO tool. It simulates the network, computes the airflow, pressure and temperature distributions. The computed results along with the input data are output to *COMPNET.OPT* file. Other output files are; *DISPLAY.DAT* which is used for graphical display of results and manipulated by DISPLAY program; *DEBUG.OPT* contains data relating to program debugging mainly for fan and leak area step-wise relaxation; *NODEPRS.OPT, STRMFLO.OPT* and *FLOCRC.OPT* are files in which the data for selected nodes or streams can be sent for successive iterations. These files are in spreadsheet compatible format and selection of nodes, streams and start step and iterations is possible at run time through these data files.

CMPNET uses four input files; namely *NETWORK.INI*, which contains all network connectivity information and is being generated by PRPNET, it is in binary form; *COMPNET.INP*, which is the main input file containing data about program controls, process gas, system components, and boundary conditions; *INTRACT.STP* contains the run-time program control parameters, first time these parameters are read interactively and saved to this file (in ASCII form), later for subsequent runs this file is read by the program without any user interaction and thus enabling faster computation; *CALFACS.STP* contains the calibration or pipe efficiency factors for all pipes and it is created in batch mode i.e. using editor.

The input files for CMPNET can be generated either interactively or in batch modes. In interactive or dialogue mode, the required inputs for all instances of each simulated entity are asked and input in turn. This is useful mode for novice users, however it is time consuming and it becomes boring because of the bulk of input required. CMPNET saves the read input to a specified file which can be edited and reused. Another shortcoming of this mode is that the whole process of input should be completed in one session, in case of crash or some data error, the created file is lost and the process is to be repeated from start. In batch mode these input files are created using some standard editor observing FORTRAN 77 data read conventions e.g. comma delimitation, enclosure of strings in single quotes etc. Since FORTRAN 77 looks only for the variables specified in READ statement so the comments can be entered in the remaining space on the lines in data file. These comments could (be their variable names or any other explanatory note) serve as guide for later changes in the file.

Creation of *COMPNET.INP* can be started from Step-2 of last section. It would be quite useful if one draws out the network in graph theoretic form also. All the instances of an entity should be grouped together and their common and specific data collected before hand. The pipe related data needs special attention. First all the non-circular pipes are converted into circular pipes offering same resistance to flow. This uses the concept that friction offered to flow is proportional to mean hydraulic depth which is cross sectional area divided by the wet perimeter (Francis 1975). Then the pipes' interconnections, either in parallel, series or both, are resolved by reducing them to a single equivalent pipe. This conversion is done manually and should be done carefully as data for these equivalent pipes is used for simulations by CMPNET.

The input file *COMPNET.INP* contains the following types of data:

a. Program controls,

b. Process gas related data,

c. Data about all the entities; pipe, valve, fan, bed and leak. The data for each entity follows a comment line, which is skipped by the program. First the number of instances of each entity are read, then the data common for all instances, and after that data specific to each instance is read.

d. Data related to leak areas relaxation is read. It is also in the form similar to type (c) above. Fist number of leaks **N2CHNG** whose areas are to be changed and stepsize for relaxation **STPSIZ** are read. Then the final values of their respective heights **HYTFIN** are read for all of these **N2CHNG** in turn.

e. Boundary conditions data in experimental units. The boundaries are all those nodes (source, sink and atmosphere) from where the system can either suck in or exhaust out the process gas. First total number of boundaries **NTBCNS** is read then for each of these boundaries, a boundary name, a 20 character comment, boundary temperature **BCTEMP**, pressure **BCPRSR** and flow **BCFLOW** are read. Although only source and atmospheric node pressures and temperatures, and sink node flows are used in computation but these all are read for the sake of format consistency.

f. Counters to dump data to spreadsheet compatible file are read. These are total number of nodes and respective node numbers and similarly for streams are specified. This information is used when the flag **DEBUG** in program controls is .TRUE.

The detailed discussion of variables etc is not possible, the sample input data file is given in Block A.2 with variable names specified, which provides sufficient information to understand and enable the running of CMPNET program.

Block A.2 Listing of sample *COMPNET.INP* file

```
!This is set-up data file COMPNET.INP; for Cross Flow in R08 to R07;11-01-93
.FALSE.,.TRUE.,1000                          !DEBUG, VLVINC, IPRINT
5.0E-03                                  !ACCURC Global
! The following is read-in by INPUT subroutine
```

```
23,6,7,7,11,11,9                    !NPIPS,NFANS,NJUNS,NBEDS,NREGS,NLEKS,NBCNS
2.000000E-05,1.400000               !DYNVSC,RATSPH
286.68,102.0,1.205                  !GASCNS,SPHTCP,DENSTY
1.01325E+05,293.0                   !PRSATM,TMPATM
50,1.000000E-05                     !ITRTNS,ACCURC
! The following is the Pipes' related data
23                                 !NPIPS
100.0,2.500000E-02,500.0,86126.25      !PIPCON,PIPTHK,FLOMAX,PRSMIN(85% of PRSATM)
'BDY-01','F3A-01',1.707,10.058,1.0     !UPUNIT,DNUNIT,PIPDIA,PIPLEN,LOCFF pipe#1
'F3A-01','REG-01',2.271,12.192,1.0                                  !       #2
'REG-02' 'REG-05',2.802,29.566,0.10
'REG-07' 'JUN-05',2.695,4.3434,1.0
'JUN-05' 'F1A-03',3.052,6.7056,1.0       ! for pipe#5
'F1A-03' 'JUN-07',2.568,30.48,1.0           ! for pipe#6 LOCFF=2.5
'REG-06' 'JUN-06',2.695,4.3434,1.0
'JUN-06' 'F1B-04',3.052,6.7056,1.0
'F1B-04' 'JUN-07',2.766,39.929,1.0          ! for pipe#9 LOCFF=2.5
'JUN-07' 'REG-10',2.568,4.572,0.1     ! for pipe#10
'REG-11' 'F2A-05',2.677,22.860,1.0
'BDY-02' 'F3B-02',2.351,9.4488,1.0
'F3B-02' 'JUN-01',3.005,3.048,1.0
'JUN-01' 'REG-03',3.005,4.572,1.0
'JUN-01' 'JUN-02',1.168,16.1544,1.0     ! for pipe#15
'REG-04' 'JUN-02',3.046,7.3152,1.0
'JUN-02' 'JUN-03',3.046,15.24,1.0
'JUN-03' 'JUN-04',1.759,9.144,1.0
'JUN-03' 'REG-08',3.098,38.7096,0.4
'JUN-04' 'JUN-05',1.106,14.9352,1.0     ! for pipe#20
'JUN-04' 'JUN-06',1.106,5.1816,1.0
'REG-09' 'F2B-06',2.677,22.860,1.0
'REG-04' 'BDY-03',3.9624,30.48,1.0      ! Stack approximated values; Last pipe#23
! The following is the Valves' related data - read_in by PIPNEW.FOR
10                                 !NVLVS
'DAMPER','GATE','GLOBE'            !3 Supported Valve Types
500.0,325.0,250.0                  !Supported Valves' Discharge COeFficients
1,'Val-01',95.0,1,'GLOBE'           !IV,VLVNAM,VLVOPN(%age),IVPIP,VLVTYP
2,'Val-02',60.0,12,'GLOBE'
3,'Val-03',70.0,15,'DAMPER'
4,'Val-04',30.0,21,'DAMPER'
5,'Val-05',30.0,20,'DAMPER'
6,'Val-06',80.0,23,'DAMPER'
7,'Val-07',90.0,5,'DAMPER'
8,'Val-08',90.0,8,'DAMPER'
9,'Val-09',80.0,22,'DAMPER'
10,'Val-10',80.0,11,'DAMPER'
! The following is Fans related data; Orignal Designe Fan data
6                                   !NFANS
0.65,0.90,179292.41,273.0,20.0,10288.45,9826.2,10819.25,2       !DISCOF,EFFCNCY,WATTAGE,IFSTAT
0.90,0.90,153788.75,273.0,44.0,10290.99,9917.6,10829.4,2         !fan#2-3B 1318720.0
0.65,0.90,352292.17,588.6,22.0,10009.08,9854.1,10468.8,2         !fan#3-1A 1580800.0
0.65,0.90,299189.31,588.6,38.0,9991.30,9902.4,10476.4,2   !fan#4-1B 1580800.0
0.65,0.90,387840.60,369.12,54.0,10245.27,9995.9,10329.1,2       !fan#5-2A
0.65,0.90,432504.92,399.7,56.0,10103.05,10052.3,10324.0,2       !fan#6-2B
1,104.4,0.2,20.0,9.0,.FALSE.  !f01-3A,ID1,AVRGFL(Kg/s),RNGFRC(%age of =/-),...
2,119.2,0.2,20.0,7.0,.FALSE.  !f02-3B,...TMPFAN(degC),PRGAIN",FXDPRS
3,67.5,0.3,500.0,10.0,.FALSE.   !f03-1A
4,60.0,0.3,500.0,10.0,.FALSE.  !f04-1B
5,129.4,0.2,100.0,12.0,.FALSE. !f05-2A
6,133.0,0.2,100.0,12.0,.FALSE. !f06-2B
! The following is Beds related data; Orignal BEDHYT & BEDWDTs;23-4-93 BED data chngd
7                                  !NBEDS; data fed in on 11th Jan 1993(Mon)
0.011,86126.25                      !PARDIA,PRSMIN (85% of PRSATM)
84.5,20.0,1132.7,0.76,3.0,0.46     !Bed#1BDAREA,BDTENT,BEDOUT,BEDHYT,BEDWDT,VOIDAG (Tent=1300 K
before 11/2/92)
82.5,20.0,726.0,0.76,3.0,0.46       !Bed#2
43.05,1198.0,806.2,0.1595,5.5,0.35  !Bed#3 incrsd BEDHYT .145 to .1595 by 10%
43.05,774.4,391.5,0.1595,5.5,0.35   !Bed#4
51.67,727.9,240.3,0.1595,5.5,0.35    !Bed#5,75% of 68.9, rest lumped to B-7,11/1/93
86.1,372.0,116.12,0.1595,5.5,0.35     !Bed#6 TBDENT chngd from 293. on 8-12-92
17.23,727.9,391.5,0.1595,5.5,0.35    !Bed#7 introdusced for X-flow on 11-01-93
! The following is Leaks related data; Orignal leaks Areas
11                                 !NLEKS; data fed in on 17th Sept 1992(Thu)
1,300.0,1.0,1.0                   !Leak#1;ID,WIDTH(cm),HEIGHT(cm),DISCOF (from 4.0)
2,300.0,4.0,1.0                    !Leak#2
3,550.0,2.0,1.0                    !Leak#3 lastly 0.825,0.7
4,550.0,1.0,1.0                   !Leak#4
5,550.0,5.0,1.0                   !Leak#5 as if there is no obstruction lastly 4.125,
6,550.0,4.0,0.0                    !Leak#6
7,550.0,5.0,1.0                    !Leak#7  XAREA, DISCOF
8,550.0,5.0,1.0                   !Leak#8  lastly 1.925,0.95
```

```
9,550.0,5.0,1.0                !Leak#9 as if there is no obstruction .9m lastly 4.950,0.95
10,550.0,2.0,1.0               !Leak#10 lastly 4.675,0.95
11,550.0,2.0,1.0               !Leak#11
! The following lines used for LEaKs' height ReLaXation, if N2CHNG=0 no RLXtn
6,4.0                           !N2CHNG,STPSIZ
10,37.0                        !for N2CHNG leaks input Leak#,HYTFIN (in Cms)
8,27.5
9,35.0
11,0.0
3,12.0
5,15.0
! The following is Bdys related data
9                             ! NTBCNS; data fed in on 28th July 1992(Tue)
BC-1; Up-end to F3A           !Bc#1; BCNAME
40.0,17.0,13811.61             !      BCTEMP,BCPRSR,BCFLOW(lbs/min) in EXP units
BC-2; Up-end to F3B           !Bc#2
40.0,16.0,15758.46
BC-3;Dn-end to STACK          !Bc#3
608.0,0.0,174.0                !7487.963,before was 6391.54 lbs/min, now in tph
BC-4; Dn-end to F2B           !Bc#4
240.0,0.0,527.0               !|orignal 17595.11,before was 19354.621 lbs/min, now in tph
BC-5; Dn-end to F2A           !Bc#5
245.0,0.0,512.0               !|orignal 17117.58,before was 18829.338 lbs/min, now in tph
ATM-06;lnkd->REG-05           !Bc#6
40.0,0.0,0.0
ATM-07;lnkd->REG-06           !Bc#7
40.0,0.0,0.0
ATM-08;lnkd->REG-10           !Bc#8
40.0,1.0,0.0
ATM-09;lnkd->REG-11           !Bc#9
40.0,0.0,0.0
! The following is read by NEWALG-main for output dumping to Worksheet format
4                             !NUMNOD
9,10,11,21
6                             !NUMSTR
12,26,6,7,36,34
```

The connectivity data for components come from *NETWORK.INF* file, the up-end and down-end components mentioned in context of pipe entity have just been used for output purpose only. These are to keep track of equivalent pipes that what they represent and to what other network components they are connected to. The upend and down-end components mentioned here are not used in the actual computation.

The program is run by typing **CMPNET** from command line. If the file *INTRACT.STP* does not exist in the current directory, then it generates one and asks all required inputs from the user interactively. However, if the file exists then it compares the date when it was previously read with today's date, in case these two are same then program proceeds without any further request for data, but if these dates are different then the user is asked whether it should modify the date to today's date, enter 'yes' to proceed. The objective of this file is to automate and facilitate multiple runs to simulate a system having same values for most of the parameters.

Block A.3 shows the contents of a sample *INTRACT.STP* file.

Block A.3 Listing of a sample *INTRACT.STP* file

```
          100     !MAXITR
Y     !LEAKS
N     !CALBRT
Y     !DMPWKS
           20     !STPSLC
            1     !SLCITI
          200     !SLCITF
26-03-94     !TODAY
            3     !NUMRUN
```

Any of these parameters can be changed by editing this data file. ***MAXITR*** is maximum number of iterations to be performed at a step; ***LEAKS*** is a flag to include (Yes) or exclude (No) leaks from the simulation; similarly ***CALBRT, DMPWKS*** are also flags to calibrate the network and to dump out in worksheet format to *.OPT* files for debugging respectively. Next three integer variables are to control this selection for dumping ***STPSLC*** selects the step, and ***SLCITI, SLCITF*** are initial and final iteration numbers. ***TODAY*** is a string for today's date, it must be input in the shown format (or be prepared to re-run the program). ***NUMRUN*** is run number of current simulation which is incremented by one each time CMPNET is run.

On successful completion of the CMPNET run, the user is asked to enter a comment (usually identifying the objective of present run) which along with other key information about the run, like run number, date of execution and CPU time used etc, is embedded in *DISPLAY.DAT* file and serves as reference in graphical output.

The main CMPNET output file is *COMPNET.OPT*, which contains all input data, connectivity information, geometrical and property data of all instances, initial fed in boundary conditions, and flow, pressure and temperature distributions with respect to each of the network components. These values are output in computational as well as in experimental units. The connectivity information read through *NETWORK.INF* file is output in linked lists form, this includes the information about stream composition as well. The computed results

can be output on each iteration to see how these are improving, but obviously it will require good amount of space.

Block A.4 shows the excerpts from the *COMPNET.OPT* file. The repetitive parts have been deleted for space reasons.

Block A.4 Partial listing of a sample *COMPNET.INF* file

```
  **** Output of CMPNET ****          Program Executed on 26-03-94
                                             Started at      16:17:09

Flowing medium properties :

    GASCNS =   286.6800 N-m/Kg-K          DYNVSC =    0.00002 N-s/m**2
    SPHTCP =  102.00000 N-m/Kg-K          RATSPH = 1.400 #
    DENSTY =  1.205 Kg/m**3

NETWORK consists of :

    Pipes =  23        Fans =   6         Beds  =   7
    Regions =  11      Leaks = 11         Junctions =   7

Computational constants :

   (Global)            Max.Itrns = 100    Accurac =    0.500E-02
   (Local)             Max.Itrns =  50    Accurac =   0.100E-04

Printing Options :

    DEBUG =   F        Print Intrvl =1000

    Following are the READ-in values for  9 Bndy Cndtns
              { in EXPerimental units }

    Sr.  Name or                        Temp       Stat.Pres    Flow
     #   Description                     o F       " of Water   Lbs/Min

     1   BC-1; Up-end to F3A           40.0          17.0       0.138E+05

     2   BC-2; Up-end to F3B           40.0          16.0       0.158E+05

     3   BC-3;Dn-end to STACK          608.        0.000E+00      174.
      .
      .
      .
     9   ATM-09;lnkd->REG-11           40.0        0.000E+00    0.000E+00

   Boundary condition variables converted :
              { in COMputational/SI units }

    Sr.  Name or                        Temp       Pres         Flow
     #   Description                     o K       Pascal       Kg/Sec

     1   BC-1; Up-end to F3A           313.         0.106E+06    0.384E+04

     2   BC-2; Up-end to F3B           313.         0.105E+06    0.438E+04

     3   BC-3;Dn-end to STACK          881.         0.101E+06     48.3
      .
      .
```

```
      9   ATM-09;lnkd->REG-11          313.          0.101E+06     0.000E+00


 INITIAL input for  23 pipes of network

      Material Conductivity : 100.000 Pipe thickness:  0.0250 m

        (Pipes followed by '*' contain valves)
          Pipe       Diameter       Length     Up-end     Dn-end
            #         meter          meter       unit       unit

            1         1.707          10.058     BDY-01     F3A-01
+                                               *
            2         2.271          12.192     F3A-01     REG-01
            3         2.802          29.566     REG-02     REG-05
            4         2.695           4.343     REG-07     JUN-05
            5         3.052           6.706     JUN-05     F1A-03
+                                               *
            6         2.568          30.480     F1A-03     JUN-07
            7         2.695           4.343     REG-06     JUN-06
            8         3.052           6.706     JUN-06     F1B-04
+                                               *
            9         2.766          39.929     F1B-04     JUN-07
           10         2.568           4.572     JUN-07     REG-10
                 .
                 .
                 .
           22         2.677          22.860     REG-09     F2B-06
+                                               *
           23         3.962          30.480     REG-04     BDY-03
+                                               *


 INITIAL inputs for    6 fans in the network

    Fan D.Coef  Efcncy  Wattag          Tcrtcl          Pcold          IFSTAT
     #     #       #      Watt            o K           Pascal           #
    1   0.650    0.900  0.17929E+06      273.00         10288.          2
    2   0.900    0.900  0.15379E+06      273.00         10291.          2
    3   0.650    0.900  0.35229E+06      588.60         10009.          2
    4   0.650    0.900  0.29919E+06      588.60         9991.3          2
    5   0.650    0.900  0.38784E+06      369.12         10245.          2
    6   0.650    0.900  0.43250E+06      399.70         10103.          2

Parameteric input of  11 Leaks in the NETWORK

              Leak #     Cross-AREA          Dis-Coeff
                           m**2                #

                L01      0.30000E-01          1.00
                L02      0.12000              1.00
                L03      0.11000              1.00
                L04      0.55000E-01          1.00
                .
                .
                .
                L11      0.11000              1.00

INITIAL DATA FOR BEDS

Number of Beds in network =     7

    Bed related parameters:
```

PARDIA = 0.01100 m

Bed no.	Bed Area	Bed Temp	Bed Voidag	Height
1	84.500	849.350	0.760	0.460
2	82.500	646.000	0.760	0.460
3	43.050	1275.100	0.160	0.350
4	43.050	855.950	0.160	0.350
5	51.670	757.100	0.160	0.350
6	86.100	517.060	0.160	0.350
7	17.230	832.700	0.160	0.350

STREAMs AND LINKED-LIST STRUCTURE

No. of Total Streams : 38

INDEX=	1	2	3	4	5	6	7	8	9	10	11	12
SCMPs=	p01	f01	p02	b01	p03	b03	p07	p08	f04	p09	p10	b06
NXTCMP=	2	3	0	0	0	0	0	9	10	0	0	0
SPREV=	0	1	2	0	0	0	0	0	8	9	0	0
SNAME=	s01	s02	s03	s04	s05	s06	s07	s08	s09	s10	s11	s12
SFIRST=	1	4	5	6	7	8	11	12	13	15	16	17
SLAST=	3	4	5	6	7	10	11	12	14	15	16	18
TORN_st=	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.
UPNODE=	1	2	3	4	5	7	10	11	12	4	6	8
DNNODE=	2	3	4	5	7	10	11	12	13	6	8	9
InclFN=	1	0	0	0	0	3	0	0	0	0	0	2
INDEX=	13	14	15	16	17	18	19	20	21	22	23	24
SCMPs=	p11	f05	b04	p04	p05	f3	p21	p20	p12	f02	p13	p15
NXTCMP=	14	0	0	0	18	0	0	0	22	23	0	0
SPREV=	0	13	0	0	0	17	0	0	0	21	22	0
SNAME=	s13	s14	s15	s16	s17	s18	s19	s20	s21	s22	s23	s24
SFIRST=	19	20	21	24	25	26	27	28	29	30	31	32
SLAST=	19	20	23	24	25	26	27	28	29	30	31	33
TORN_st=	.T.	.T.	.F.	.T.	.F.	.F.	.F.	.F.	.F.	.F.	.F.	.F.
UPNODE=	24	24	14	15	15	16	17	18	19	19	20	21
DNNODE=	7	8	15	18	16	17	18	19	24	20	21	22
InclFN=	0	0	4	0	0	0	0	0	0	0	0	0
INDEX=	25	26	27	28	29	30	31	32	33	34	35	36
SCMPs=	p14	b02	p16	p17	p18	p19	b05	p22	f06	p23	p06	b07
NXTCMP=	0	0	0	0	0	0	0	33	0	0	0	0

SPREV=	0	0	0	0	0	0	0	0	32	0	0	0
SNAME=	s25	s26	s27	s28	s29	s30	s31	s32	s33	s34	s35	s36
SFIRST=	34	35	36	37	38	39	40	41	42	43	44	45
SLAST=	34	35	36	37	38	39	40	41	42	43	44	45
TORN_st=	.F.	.T.	.T.	.T.	.T.	.T.	.T.	.T.	.T.	.T.	.T.	.T.
UPNODE=	17	9	20	2	3	25	26	5	4	6	11	12
DNNODE=	23	10	6	16	17	4	5	6	20	21	20	21
InclFN=	0	0	0	0	0	0	0	0	0	0	0	0
INDEX=	37	38	39	40	41	42	43	44	45	46	47	
SCMPs=	101	102	103	104	105	106	107	108	109	110	111	
NXTCMP=	0	0	0	0	0	0	0	0	0	0	0	
SPREV=	0	0	0	0	0	0	0	0	0	0	0	
SNAME=	s37	s38										
SFIRST=	46	47										
SLAST=	46	47										
TORN_st=	.T.	.T.										
UPNODE=	27	28										
DNNODE=	11	12										
InclFN=	0	0										

NODES and LINKED-LIST
STRUCTURE

No. of Total NODEs : 28

NINDEX=	1	2	3	4	5	6	7	8	9	10	11	12
In/Out=	-1	1	-1	-1	1	-1	-1	1	1	-1	-1	-1
STRNUM=	1	1	28	2	2	29	3	3	30	33	4	10
NXTSTR=	-1	3	4	-2	6	7	-3	9	10	11	12	-4
NINDEX=	1	2	3	4	5	6	7	8	9	10	11	12
NODNAME=	B01	R01	R02	R05	R06	R07	J06	J05	J08	J07	R10	R11
NDPNTR=	1	2	5	8	13	17	22	25	28	30	33	37
InDEGree=	1	3	3	5	4	5	3	3	2	3	4	4
NINDEX=	13	14	15	16	17	18	19	20	21	22	23	24
In/Out=	1	1	-1	-1	1	1	1	-1	-1	1	1	-1
STRNUM=	4	31	5	32	10	27	32	34	11	5	13	6
NXTSTR=	14	15	16	-5	18	19	20	21	-6	23	24	-7

```
 NINDEX=     13   14   15   16   17   18   19   20   21   22   23   24
 NODNAME=    B05  B02  J01  R03  R04  J02  J03  R08  R09  B04  B03  J04
 NDPNTR=     41   42   43   46   49   53   56   59   64   68   69   70
 InDEGree=    1    1    3    3    4    3    3    5    4    1    1    3

 NINDEX=     25   26   27   28   29   30   31   32   33   34   35   36
 In/Out=      1    1   -1    1   -1    1    1   -1    1    1   -1   -1
 STRNUM=     11   14   12   12   26    6   26    7    7   37    8   35
 NXTSTR=     26   27   -8   29   -9   31   32  -10   34   35   36  -11

 NINDEX=     25   26   27   28
 NODNAME=    A06  A07  A08  A09
 NDPNTR=     73   74   75   76
 InDEGree=    1    1    1    1

 NINDEX=     37   38   39   40   41   42   43   44   45   46   47   48
 In/Out=      1    1   -1   -1    1   -1    1   -1   -1    1    1   -1
 STRNUM=      8   38   36    9    9   15   15   17   16   17   28   18
 NXTSTR=     38   39   40  -12  -13  -14   44   45  -15   47   48  -16

 NINDEX=     49   50   51   52   53   54   55   56   57   58   59   60
 In/Out=      1    1   -1   -1    1    1   -1    1   -1   -1    1    1
 STRNUM=     18   29   19   25   19   16   20   20   22   21   22   33
 NXTSTR=     50   51   52  -17   54   55  -18   57   58  -19   60   61

 NINDEX=     61   62   63   64   65   66   67   68   69   70   71   72
 In/Out=      1   -1   -1    1    1    1   -1    1    1    1   -1   -1
 STRNUM=     35   23   27   23   34   36   24   24   25   21   13   14
 NXTSTR=     62   63  -20   65   66   67  -21  -22  -23   71   72  -24

 NINDEX=     73   74   75   76
 In/Out=     -1   -1   -1   -1
 STRNUM=     30   31   37   38
 NXTSTR=    -25  -26  -27  -28
****************** Output of CMPNEET on    4 th Iteration ******************
FINAL variables of  23 pipes in NETWORK at    4 th Iteration :

   Pipe      Up-stream  end                        Dn-stream  end
     #         Temp       Pres          Flow         Temp       Pres
               K          Pascal        Kg/S         K          Pascal

      1      313.00     0.1056E+06     120.5       312.36     0.1028E+06

      2      312.36     0.1045E+06     120.5       311.76     0.1044E+06
```

3	1401.07	0.1012E+06	121.0	968.53	0.1011E+06
4	664.50	0.9831E+05	89.52	657.65	0.9830E+05
5	664.71	0.9830E+05	97.62	655.28	0.9759E+05
6	655.29	0.1014E+06	97.62	655.29	0.1013E+06
7	982.04	0.9824E+05	77.39	958.92	0.9823E+05
8	938.63	0.9823E+05	85.61	911.08	0.9768E+05
9	911.08	0.1014E+06	85.61	749.84	0.1013E+06
10	699.97	0.1013E+06	183.2	692.49	0.1013E+06
11	389.12	0.9851E+05	142.2	383.71	0.9532E+05
12	313.00	0.1053E+06	143.9	312.55	0.1029E+06
13	312.55	0.1041E+06	143.9	312.43	0.1041E+06
14	312.44	0.1041E+06	122.6	312.26	0.1040E+06
15	312.44	0.1041E+06	21.25	312.44	0.1012E+06
16	998.92	0.1012E+06	73.83	961.13	0.1012E+06
17	831.18	0.1012E+06	95.08	789.05	0.1012E+06
18	789.06	0.1012E+06	16.33	741.20	0.1012E+06
19	789.06	0.1012E+06	78.76	695.25	0.1009E+06
20	741.22	0.1012E+06	8.099	741.22	0.9830E+05
21	741.22	0.1012E+06	8.227	741.22	0.9823E+05
22	476.29	0.9824E+05	146.4	464.49	0.9486E+05
23	998.92	0.1012E+06	48.33	858.84	0.1012E+06

FINAL Values of variables for 6 Fans After 4 Iterations

Fan #	i n p u t e n d Temp o K	Pres Pascal	Flow Kg/s	o u t p u t e n d Temp o K	Pres Pascal	Flow Kg/s	T.Cntrlr Fcold Kg/s
1	312.36	0.10285E+06	120.55	312.36	0.10450E+06	120.55	0.00
2	312.55	0.10288E+06	143.88	312.55	0.10407E+06	143.88	0.00
3	655.28	97593.	97.62	655.28	0.10144E+06	97.62	0.00
4	911.08	97682.	85.61	911.08	0.10141E+06	85.61	0.00
5	383.71	95319.	142.22	383.71	98145.	142.22	0.00
6	464.49	94863.	146.39	464.49	97914.	146.39	0.00

Computed values for 12 Leaks after 4 Iterations

Leak no.	Up-end PRES Pascal	FLOW Kg/S	Dn-end PRES Pascal
L01	0.10442E+06	0.95259	0.10400E+06
L02	0.10119E+06	-1.4343	0.10125E+06
L03	0.10133E+06	15.308	0.10110E+06
L04	0.10133E+06	4.7427	98240.
L05	98240.	-10.817	98311.
L06	0.10110E+06	0.00000E+00	0.10094E+06
L07	98311.	3.5402	98243.

L08	0.10130E+06	44.190	0.10094E+06
L09	98515.	49.286	98243.
L10	0.10157E+06	52.463	0.10130E+06
L11	0.10133E+06	0.00000E+00	98515.

OUT PUT for 7Beds after 4 Iterations

Bed #	Temp in	Pres in	Flow in	Pres out
1	311.765	104415.000	119.592	101186.445
2	312.250	103996.641	123.582	101245.727
3	903.120	101102.203	61.826	98240.078
4	903.120	101102.203	74.511	98311.414
5	666.589	100944.516	93.563	98242.648
6	614.309	101298.703	191.507	98514.648
7	666.589	100944.516	29.369	98311.414

Nodes Pressure & Temperature Distribution

Nd#	Nd_Name	Nd_Pressure " of Water	Nd_Temperature (oC)
1	B01	17.000	40.000
2	R01	12.375	38.765
3	R02	-0.61216	1128.1
4	R05	-0.95004	630.12
5	R06	-12.504	709.04
6	R07	-12.220	391.50
7	J06	-12.535	665.63
8	J05	-12.260	391.71
9	J08	0.30447	382.29
10	J07	-0.12180	426.97
11	R10	-0.26484	341.31
12	R11	-11.426	116.12
13	B05	-12.913	245.00
14	B02	16.000	40.000
15	J01	10.911	39.436
16	R03	10.667	39.250
17	R04	-0.43058	725.92
18	J02	-0.45661	558.18
19	J03	-0.54469	516.06
20	R08	-1.6551	393.59
21	R09	-12.512	203.29
22	B04	-13.834	240.00
23	B03	-0.77680	608.00
24	J04	-0.57151	468.22
25	A06	-0.63243E-05	40.000
26	A07	-0.63243E-05	40.000
27	A08	0.99999	40.000
28	A09	-0.63243E-05	40.000

Streams Flow & Temperature Distribution

St#	St_Name	St_Flow Tonnes/Hr	St_Temperature (oC)
1	s01	433.96	38.765
2	s02	430.53	1132.7
3	s03	435.65	695.53
4	s04	222.57	806.20
5	s05	278.59	685.92
6	s06	308.21	476.84
7	s07	659.64	419.49

```
 8    s08          689.43                116.12
 9    s09          512.00                110.71
10    s10          268.24                391.50
11    s11          322.28                384.65
12    s12          351.43                382.28
13    s13          29.618                468.22
14    s14          29.155                468.22
15    s15          517.99                39.433
16    s16          76.513                39.436
17    s17          441.47                39.256
18    s18          444.90                726.00
19    s19          265.78                688.13
20    s20          342.30                516.05
21    s21          58.775                468.20
22    s22          283.52                422.25
23    s23          336.83                240.30
24    s24          527.00                191.49
25    s25          174.00                585.84
26    s26          351.42                382.29
27    s27          105.73                391.50
28    s28          3.4293                38.765
29    s29         -5.1636                725.92
30    s30          55.108                40.000
31    s31          17.074                40.000
32    s32         -38.942                391.50
33    s33         0.00000E+00            630.12
34    s34          12.745                391.50
35    s35          159.08                341.31
36    s36          177.43                116.12
37    s37          188.87                40.000
38    s38         0.00000E+00            40.000
                                  Finished at     16:17:39
                                  CPU time used   00:00:30
```

The node and stream names here are worth noticing, these names have been generated by PRPNET, the first alphabetic character for node names is capital letter B - for sink and source boundary, A - for atmosphere linked to leaks, J - for junction and R - for regions. Whereas the streams first character 's' is always in lower case. Similarly the new component names are also in lower case and all these are of length 3 rather than 6 as were input in *PREPNET.INP* file. The same node and stream names would be used in the other output file *DISPLAY.DAT*, whose listing is shown in Block A.5.

Block A.5 Listing of Sample *DISPLAY.DAT* file

```
( 3) 26-03;1617 hrs[ACCU= 5.000E-03 MAXITR=100,MXSTPs= 9,CPU Time=00:30] Run second time now
directories are in path AL
Node B01      17.0 "
Node R01      12.4 "
Node R02      -0.6 "
Node R05      -0.9 "
Node R06     -12.4 "
Node R07     -12.1 "
Node J06     -12.4 "
```

```
Node J05      -12.1 "
Node J08        0.5 "
Node J07        0.0 "
Node R10       -0.1 "
Node R11      -11.3 "
Node B05      -12.8 "
Node B02       16.0 "
Node J01       11.0 "
Node R03       10.7 "
Node R04       -0.3 "
Node J02       -0.3 "
Node J03       -0.4 "
Node R08       -1.5 "
Node R09      -12.4 "
Node B04      -13.7 "
Node B03       -0.7 "
Node J04       -0.5 "
Node A06        0.0 "
Node A07        0.0 "
Node A08        1.0 "
Node A09        0.0 "
Stream s01       434.
Stream s02       431.
Stream s03       436.
Stream s04       223.
Stream s05       279.
Stream s06       308.
Stream s07       660.
Stream s08       689.
Stream s09       512.
Stream s10       268.
Stream s11       322.
Stream s12       351.
Stream s13        30.
Stream s14        29.
Stream s15       518.
Stream s16        77.
Stream s17       441.
Stream s18       445.
Stream s19       266.
Stream s20       342.
Stream s21        59.
Stream s22       284.
Stream s23       337.
Stream s24       527.
Stream s25       174.
Stream s26       351.
Stream s27       106.
Stream s28         3.
Stream s29        -5.
Stream s30        55.
Stream s31        17.
Stream s32       -39.
Stream s33         0.
Stream s34        13.
Stream s35       159.
Stream s36       177.
Stream s37       189.
Stream s38         0.
Valve v01  95. %
Valve v02  60. %
Valve v03  70. %
Valve v04  30. %
Valve v05  30. %
Valve v06  80. %
Valve v07  90. %
Valve v08  90. %
Valve v09  80. %
Valve v10  80. %
Fan F01     6.1    12.7 "
Fan F02     6.3    11.0 "
Fan F03   -15.0     0.5 "
Fan F04   -14.6     0.3 "
Fan F05   -24.1   -12.8 "
Fan F06   -25.9   -13.7 "
Temperature B01     40.
Temperature R01     39.
Temperature R02   1128.
Temperature R05    630.
Temperature R06    709.
```

```
Temperature R07    392.
Temperature J06    666.
Temperature J05    392.
Temperature J08    382.
Temperature J07    427.
Temperature R10    341.
Temperature R11    116.
Temperature B05    245.
Temperature B02     40.
Temperature J01     39.
Temperature R03     39.
Temperature R04    726.
Temperature J02    558.
Temperature J03    516.
Temperature R08    394.
Temperature R09    203.
Temperature B04    240.
Temperature B03    608.
Temperature J04    468.
Temperature A06     40.
Temperature A07     40.
Temperature A08     40.
Temperature A09     40.
Temperature s01     39.
Temperature s02   1133.
Temperature s03    696.
Temperature s04    806.
Temperature s05    686.
Temperature s06    477.
Temperature s07    419.
Temperature s08    116.
Temperature s09    111.
Temperature s10    392.
Temperature s11    385.
Temperature s12    382.
Temperature s13    468.
Temperature s14    468.
Temperature s15     39.
Temperature s16     39.
Temperature s17     39.
Temperature s18    726.
Temperature s19    688.
Temperature s20    516.
Temperature s21    468.
Temperature s22    422.
Temperature s23    240.
Temperature s24    191.
Temperature s25    586.
Temperature s26    382.
Temperature s27    392.
Temperature s28     39.
Temperature s29    726.
Temperature s30     40.
Temperature s31     40.
Temperature s32    392.
Temperature s33    630.
Temperature s34    392.
Temperature s35    341.
Temperature s36    116.
Temperature s37     40.
Temperature s38     40.
```

The format shown for *DISPLAY.DAT* is specific to the needs of the program DISPLAY, which reads in this file. The first line includes the comment generated by the CMPNET. This file has two parts, the first relates to flow and pressure distribution whereas the later part corresponds to temperature distribution. Both of these parts have slightly

different formats, since for the first the flows are associated to streams and pressures to nodes, whereas in the second part each node and stream is having a temperature.

The general format for the flow and pressure distribution is:

```
Entity_Name, Instance_Name, Variable_Value, Units
```

Where `Entity_Name` can be exclusively a node, stream, valve or a fan; `Instance_Name` is the component name for respective component; `Variable_Value` is pressure value for node and fan entities, flow value for stream and percentage of valve opening for respective valve; `Units` are " representing pressure in Inches of water, % for valve opening and blank for flow. The upend and downend pressures are refered for fans so the variable value field has two values. All fields are terminated by white-space.

Since the temperature distribution is to be plotted only for node and stream entities so the format for the second part of output is simpler. It has fields

```
Temperature, Instance_Name, Variable_Value
```

Where the fist field `Temperature` remains fixed for all instances of Node and Stream entities; `Instance_Name` contains the node or stream names; and `Variable_Value` is corresponding value of temperature for respective instance. Manipulation of this data file by program DISPLAY will be discussed in next section.

A.4.3 DISPLAY - Display of computed results :

This is the post-processor part of GASFLO. It can either display the computed airflow distribution graphically on screen, or can print it on a PostScript printer. Basically it is a batch file which is run by typing

```
DISPLAY -P Picture.fil -D DISPLAY.DAT [-Q|-V|-H] Parameter-list
```

Where the switches and parameters in **`Parameter-list`** are case sensitive, so must be input in the shown case. The switches have following meanings:

`-P` indicates the next argument is the static PostSctipt picture file,

`-D` indicates the next argument is a data file *DISPLAY.DAT* generated by program CMPNET,

`-Q` to send or queue the produced graphical output to PostScript printer,

`-V` to view the produced graphical output on screen using PostScript viewer like GhostScript (this is default), and

`-H` shows a brief help about the syntax of the expected command.

The **`Picture.fil`** is a static PostScript file (procedure for whose creation will be discussed shortly), and **`Parameter-list`** may contain any or all of the following pairs:

`[Node Pressure], [Stream Flow], [Fan Pressure], [Valve Opening]`

Where first argument of each of the pair is entity name and second is its attribute.

The display of temperature distribution is carried out on separate diagram, and it is activated by the copy of same program but with different name DISPLAYT instead of DISPLAY. Which uses the same syntax as described above, but with corresponding static picture file and the **`Parameter-list`** has only pair as

`[Temperature Distribution]`

Obviously without [] brackets.

Creation of static picture file

The easiest possible way to create the static file is by using any drawing package which can export (encapsulated) PostScript files. We used Draw Perfect 1.1 to create the provided sample static files. While drawing important points to remember are:

a. Draw the figure as close as possible to the original schematic of the plant,

b. Decide for the locations on the diagram where data is to be output, and mark them with 'stubs' using exactly same names as instance names in the shown *DISPLAY.DAT* file,

c. Choose the desired appearance for text (e.g. *italics,* or underline etc) when entering stubs. It would be easier for the reader later on to recognise if all flow values have one appearance and pressures the other,

d. Enter the stubs; 'Something' in the drawing to output heading which will be generated by the DISPLAY using arguments from parameter list; and 'Reference' to output the comment line identifying the displayed output. These two should have exactly the same case and can have appearance as desired by the user. For example the heading or title can have large and bold appearance to be visible, whereas comment can be in very small or fine font to avoid viewer's distraction but readable for careful reader,

e. After completing the drawing, export it as PostScript or Encapsulated PostScript as is facilitated by drawing package and name it as **`Picture.fil`**.

Figure A.5 shows a sample static picture file generated by the above procedure, for flow and pressure distribution output.

Working of DISPLAY:

Conceptually, working of DISPLAY is very straight forward. For example the command line:

```
DISPLAY -P Picture.file -D DISPLAY.DAT -Q Node Pressure Stream Flow Valve Opening Fan Pressure
```

would produce an output like the one shown in Figure 5.3, which in fact involved following steps;

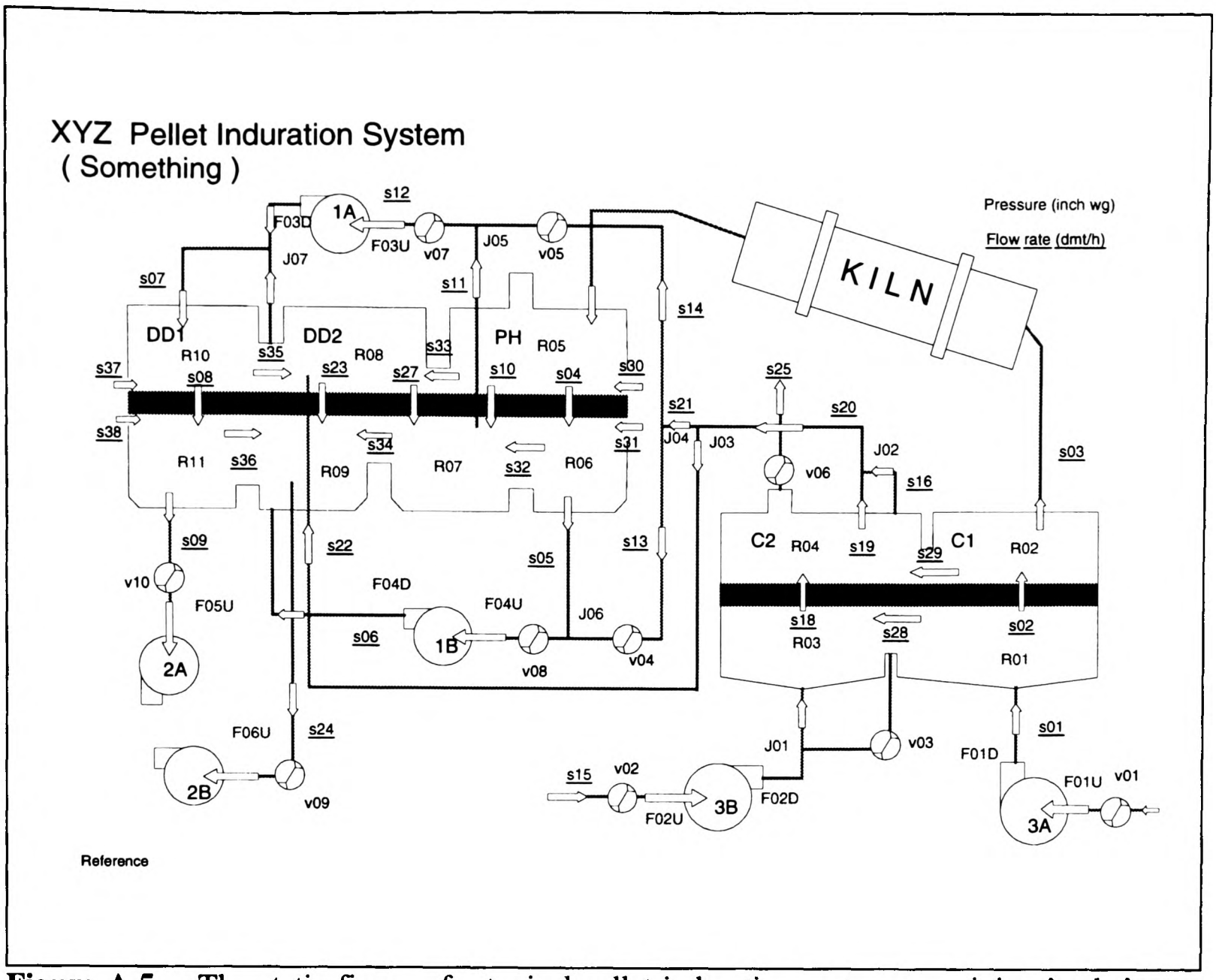

Figure A.5 The static figure of a typical pellet induration system containing 'stubs'

1. First a SPECFILE is generated by parsing the attributes corresponding to parameter list. In this case it will contain the string

   '**Node Pressure, Stream Flow, Valve Opening, Fan Pressure**'

   The commas between the different groups will be inserted by DISPLAY,

2. According the this generated SPECFILE, the program will scan the *DISPLAY.DAT* file, look for the required entities the instance names and their values and copy them down into another temporary file *SEDSCPT*,

3. It will edit the *Picture.fil*, using *SEDSCPT* file, each of the instance name stub by its value,

4. The edited *Picture.fil* named temporarily as *PICOUT* would be scaled and rotated according to the destination device, screen or printer, and displayed if it is screen.

In Figure A.5, the replacement of 'Something' with the contents of *SEDSCPT*, and other stubs by their values from the *DISPLAY.DAT* can be noted in the produced output like Figure 5.3. All utility programs required by DISPLAY program are placed in directory UTILITYS.

For easier and efficient running of DISPLAY, the above stated command line is placed in another batch file, called *SHOW-F&P.BAT* to show flow and pressure distributions, and similarly *SHOW-TPR.BAT* to show temperature distribution in the sample network. Both of these does not require any arguments so they can be easily typed and are less prone to errors.

A.5 INDSYS and GASFLO interaction

These two software tools, though are related to induration system simulation and are from same group. However, they address different aspects of induration systems and are developed independently. They have different user interface and slightly different semantics.

INDSYS, INDuration SYstem Simulator, is a product of late 1980's, well used in industry, written in GW-BASIC, simulates heat concentration in the system, and now stands out as a mature reliable tool (Cross 1988, Cross and Englund 1987). Its computation takes into account the involved chemical reactions, heat exchange due to solid-gas interaction and the heat sources. The pellet induration system is modelled as combination of 'zones', where each zone is either a grate (i.e. an enclosure containing packed bed) or a kiln. A zone has an on-gas, the process gas coming into the zone, and off-gas, the gas going out of the zone, which are basically input and output of the enclosed packed bed. The on-gas of a zone is either from atmosphere or a combination of off-gas from other zones, and it is provided in

terms of the respective zones' percentages. The on-gas to some zones can come in as external leakage directly from the atmosphere.

INDSYS simulation requires the magnitude and temperature of the on-gas and external leakage flows for all zones, and computes the spatial temperature distribution of solids as well as gas in packed bed. Assuming symmetry in third dimension (i.e. in the direction of width of packed bed) the results are provided in two dimensional discretised space (i.e. in length and height of bed). Length-wise the zone (or the packed bed associated to it) is discretised into, say ***n***, intervals of equal length, and height-wise it is divided into 10 layers. Proper mixing of gas temperature is assumed in the zone regions, above and below the packed bed, and for the gas travelling in different streams, so an average temperature of on-gas is assumed for input to each zone. The computed results provide the temperature profiles for gas and solids, in tabular form by the *.OUT* file which can be presented graphically on screen using the post-processor program GINDSYS. A simulation of INDSYS for a typical induration system, takes about 5 - 10 minutes on a 486, 33 MHz PC machine.

INDSYS does not simulate components like pipes/ducts, fans or valves etc, instead it simulates packed beds only. Also for a system like shown in Figure 5.3, it sequentially assigns numbers 1 to 6 to zones DD1, DD2, PH, kiln, C1 and C2; which does not conform to the numbering conventions used by GASFLO (as will be discussed shortly).

GASFLO computes flow, pressure and temperature distribution in the whole induration system network at a macroscopic level (i.e. at the interface of all components of the network), based on (already discussed) boundary conditions and average temperatures provided by INDSYS for off-gas from the packed beds. In GASFLO, instead of zones, packed beds are treated explicitly. For a network shown in Figure A.5, the packed beds are numbered as 1 to 6 for zones C1, C2, PH_1, PH_2, DD2 and DD1. Since PH has two output regions so its associated packed bed is split into two. From airflow prospective the kiln does not play any significant role, except the temperature of process gas passing through the kiln is raised by certain amount, so this behaviour is embedded into the pipe containing kiln, rather than simulating it as an exclusive identity as done by INDSYS. The flow through bed is modelled

by Ergun's equation, which needs average temperature of gas through the bed, which is worked out inside the GASFLO from the read input values ***TBEDIN*** and ***TBEDOU*** for respective bed.

The airflow distribution provided by GASFLO is required as input by INDSYS and the gas temperatures for packed bed computed by INDSYS are needed for GASFLO computation. Thus for realistic and complete simulation of an induration system the two tools should be run interactively and iteratively using each others outputs. This iterative process should be continued until a converged temperature distribution is achieved. The interactive and alternate running of INDSYS and GASFLO is possible with their existing states, by manually extracting the required data from the respective output files and editing the corresponding input files. It will require following steps:

Step-1: Establish association between

a. the zones of INDSYS and packed beds of GASFLO, by comparing the bed related data, their heights, widths and lengths and their connectivity to other components

b. external leakage and off-gas zone flows of INDSYS to the stream numbers of GASFLO

c. the grid points for on-gas or first layer and for last layer of each bed in INDSYS output file to the ***TBEDIN*** and ***TBEDOU*** of respective beds of GASFLO

Step-2: Assume flow distribution and run INDSYS,

Step-3: Extract gas temperatures relevant to each packed bed manually from INDSYS output file, work out ***TBEDIN*** and ***TBEDOU*** by averaging for all beds. Edit the input file for GASFLO to substitute these bed temperatures,

Step-4: Run GASFLO, and compute the flow, pressure and temperature distributions for the network. Extract flow and temperature for all streams (including leaks) coming into

the regions upstream to each bed. Edit the input data file for INDSYS to substitute these available values,

Step-5: Run INDSYS (with recent flow distribution)

Steps 3 to 5 might be continued, until converged temperature distribution in the system is achieved. This takes about 2-5 iterations for a typical pellet induration system.

A.6 Known Failures or Errors

DISPLAY: Error with message

```
sed: SEDSCPT (line 2): garbage after command
```

on screen is due to input of some reserved character such as '/', which is delimiter for the 'sed', in the typed-in comment line. This can be corrected by editing the *DISPLAY.DAT* file to eliminate such characters from the comment line, and re-running the DISPLAY (or the other batch files SHOW-F&P etc which call DISPLAY).

CMPNET: Gives self explanatory error messages.

- Errors due to mismatch of counters, numbers of instances of the modelled entities. The network information file generated by PRPNET carries different number than the corresponding number provided by *COMPNET.INP* data file.

  These errors can be corrected by editing the *COMPNET.INP* file for correct number.

- Errors due to wrong format of input data say real values for integer variables or for example specification of date in format other than dd-mm-yy may lead to program

crash. Such errors can be cured by input of variables in the right format and re-running the program.

A.7 Intended Improvements for Version 3.0

GASFLO has been developed on the software engineering principles. It is extensible, more entities can be added. It can configure and partition the network from the input connectivity. The data for all instances can be input interactively or through a pre-edited file. More entities can be added to the system by writing separate modules according to a specific template provided, which enables to embed data and methods together. In the existing entities the mathematical model or their numerical method (or both) can be changed or replaced, and different modules can have different computational methods. Even an entity can have multiple mathematical models and different models can be selected for different instances at run time. These facilities make it an invaluable tool for mathematical modellers. It is very fast, takes less than a minute to simulate a typical pellet induration system on a 486 PC. It provides flow, pressure and temperature distributions in the whole network, including variables like leakages, which are though qualitatively sensed by practitioners but they along with some other variables can not be quantified by proper measurements due to hostile environment.

However, still it lacks the following facilities which are intended to be provided in the future version:

- Graphical User Interface, through which the user can select the available entities, by combining different instances of these entities one could draw the network, input the required data be editing through provided forms, and simulate the network. The boundary conditions and component parameters etc could be changed by clicking on respective components.
- Facility for adding user defined entities, by writing them as FORTRAN modules and compiling and linking them to the already existing entity library.

- Refinement of existing entities' mathematical models to more specific ones, which are developed as a necessity with insufficient data available about their nature, the examples being the fans and valves and these have been tested for their physical results.
- The process gas has been treated as single phase incompressible medium for simplicity reasons. Whereas in practice, in drying and heating stage the presence of water vapours necessitates that it should be dealt as two phase flow. Secondly, the temperature range shows that there is significant variation of air density, which emphasizes that compressibility of air should be taken into account.
- Validation of input data and provision of default data for some standard parameters to lessen the burden on user for pre-requisite knowledge of these variables.

Appendix - B

Derivation of the Used Correction Terms

In this appendix we will derive the correction terms used by the primary solution algorithm (Figure 3.3) discussed in section 3.3, these terms were proposed by Boyne 1970. We will first develop these terms for a single loop network described in section 3.2.2, and later generalise them to general multiple loop network.

According to the steps mentioned in the primary solution algorithm (cf. Figure 3.3), first the network is partitioned into tree and cotree structures, then using known sink node flows (a boundary condition) and an assumed cotree flow distribution the flows in tree streams are evaluated. In steps 5.0 and 6.0 the pressures at tree nodes and flow in cotree branches are computed, which are then used to evaluate the residuals at all the nodes. If these residuals are not within the specified tolerance then the flows are corrected and steps repeated. The residual at the ith node, f_i, will be a function of flows from the incident streams and given by equation 3.1 namely,

$$f_i(F_1, F_2, \ldots, F_{N_{TSTR}}) = \sum_{j=1}^{N_{TSTR}} a_{ij} F_j \qquad \textbf{(B.1)}$$

where F_j is flow in the jth stream and a_{ij} is an element of the node-stream incidence matrix.

This residual will be non-zero and will mainly depend on the values of flows in the cotree branches because at the current iteration the cotree flows have been evaluated from the

recent pressure distribution, which are different and lead to the flow imbalance at the nodes connected by the cotree branches. The final solution to the network will provide such a value of cotree flows i.e. F^*_{cotree} which will reduce this f_i to zero for all internal nodes and hence satisfy all the loop equations 3.2. i.e.

$$f_i(F^*_{cotree}) = f_i = 0 \qquad \textbf{(B.2)}$$

For our example single loop network (Figure 3.2), the cotree is comprised of stream *s03* only and the recent computed flow F_{s03} introduces non-zero residuals of amount f_{J01} and f_{J02} at respective nodes both of these are equal and opposite in sign.

We need to determine the value of F_{cotree} which reduces the nodal residual f_i to zero. By Newton Raphson method or Taylor's theorem, the value of F_{cotree} can be determined iteratively, for (K+1)th iteration it can be approximated as

$$F^{K+1}_{cotree} = F^{K}_{cotree} + \Delta F^{K}_{cotree} \qquad \textbf{(B.3)}$$

Where

$$\Delta F^{K}_{cotree} = - \frac{f_i(F^{K}_{cotree})}{f_i'(F^{K}_{cotree})} \qquad \textbf{(B.4)}$$

In GASFLO, using the device centred approach the relationship $f_i(F^K_{cotree})$ is dependent on the mathematical models of the network components making the cotree branch, so the derivative f_i' with respect to $F_{cotree}{}^K$, in the denominator of the above equation is not readily available and would require numerical differentiation which is computationally expensive and complicated. By contrast, the previous iteration values of cotree flows and node residuals are available so using these the derivative can be approximated as

Substituting this into equation B.4, we have

$$f_i'(F_{cotree}^{K}) = \frac{f_i(F_{cotree}^{K}) - f_i(F_{cotree}^{K-1})}{F_{cotree}^{K} - F_{cotree}^{K-1}} \qquad \textbf{(B.5)}$$

$$\Delta F_{cotree}^{K} = - \frac{f_i(F_{cotree}^{K}) \times [F_{cotree}^{K} - F_{cotree}^{K-1}]}{f_i(F_{cotree}^{K}) - f_i(F_{cotree}^{K-1})} \qquad \textbf{(B.6)}$$

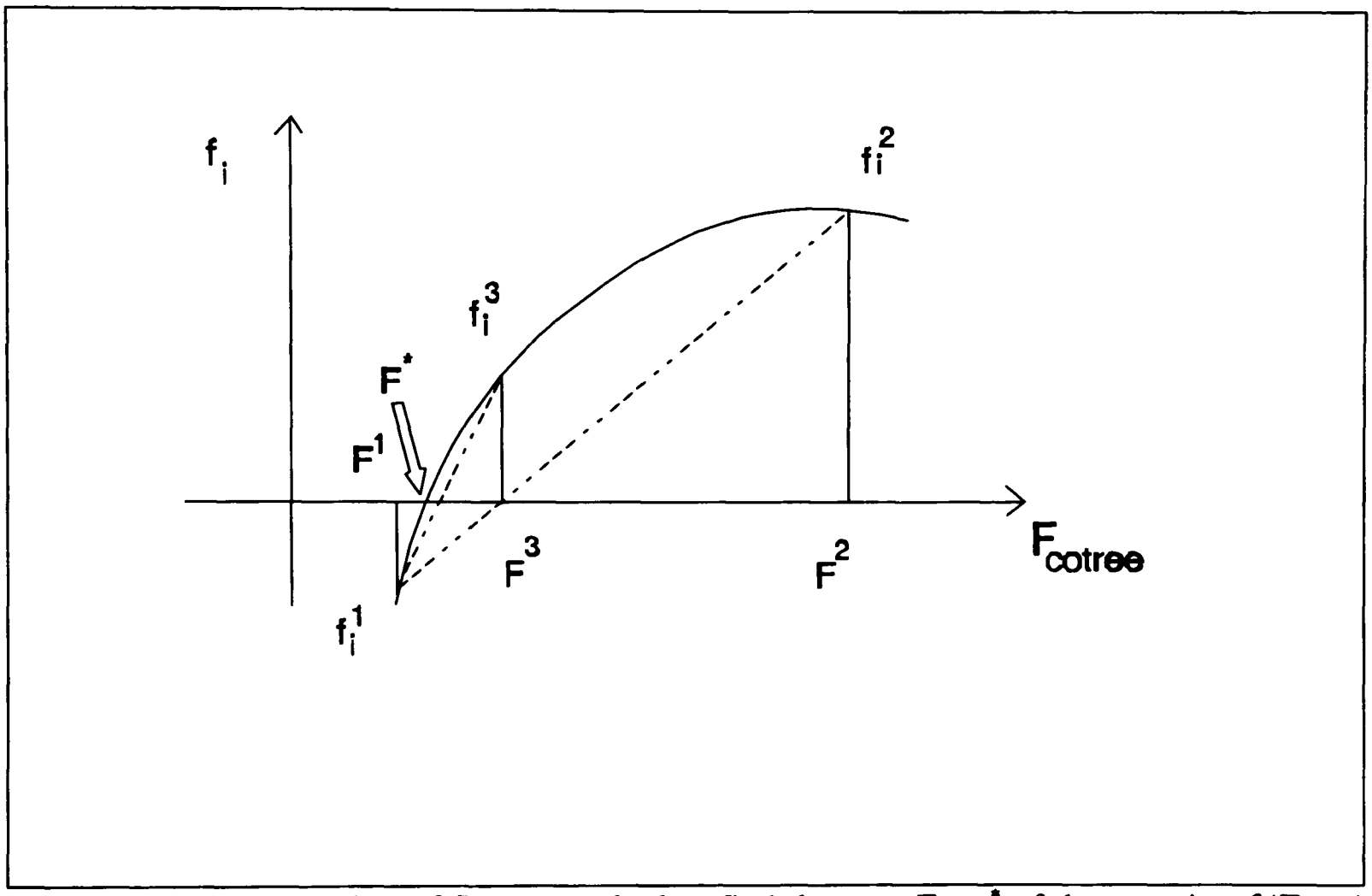

Figure B-1 Application of Secant method to find the root F_{cotree}^{*} of the equation $f_i(F_{cotree}) = 0$

The equations B.3 and B.6 are in fact the well known Secant method which states that the next estimate of F_{cotree} would be the value where the secant to the curve f_i (drawn between its previous two values) intercepts the F_{cotree} axis. This is shown graphically in Figure B-1. Using the definition $F_{cotree}^{K} - F_{cotree}^{K-1} = \Delta F_{cotree}^{K-1}$ the equation B.6 becomes

$$\Delta F_{cotree}^{K} = - \frac{f_i(F_{cotree}^{K}) \times [\Delta F_{cotree}^{K-1}]}{f_i(F_{cotree}^{K}) - f_i(F_{cotree}^{K-1})} \quad \textbf{(B.7)}$$

In principle this correction is applied to the cotree flows to obtain a better estimate which alters the flow in tree branches. For our single loop network this increase of ΔF^{K}_{cotree} in cotree branches will require; an increase of flow in all tree branches between the node *J01* and the source node i.e stream *s01* by an amount of ΔF^{K}_{cotree}, as well as, a decrease of flow by the same amount in all the branches of tree between the other end (*J02*) of the respective cotree branch and the source node of the network i.e. the streams *s02* and *s01*. These corrections can be applied directly to tree branches by treating it as a node load at the respective nodes. Let δq_i^{K} be the node load for the ith node and at Kth iteration which is same in magnitude as cotree flow correction ΔF^{K}_{cotree} (also changing ΔF_{cotree}^{K-1} to δq_i^{K-1} in the equation), then

$$\delta q_i^{K} = - \frac{f_i(F_{cotree}^{K}) \times [\delta q_i^{K-1}]}{f_i(F_{cotree}^{K}) - f_i(F_{cotree}^{K-1})} \quad \textbf{(B.8)}$$

The same sign convention is adopted for these node loads as for the other streams i.e it is positive for the flow coming into the node and negative for the outgoing flow. As mentioned in section 3.2.2 the negative sign with the above node load reflects the desired increase or decrease of flows in respective tree branches hence

$$\delta q_i^{K} = - \frac{f_i(F_{cotree}^{K}) \times |[\delta q_i^{K-1}]|}{|f_i(F_{cotree}^{K}) - f_i(F_{cotree}^{K-1})|} \quad \textbf{(B.9)}$$

Now we evaluate the overall system correction, which is sum of corrections at the individual nodes. Let this be ΔQ^{K} for Kth iteration. For our single loop network, there is only one cotree branch linking the two nodes, so

$$\Delta Q^K = |\delta q_{J01}^K| + |\delta q_{J02}^K| = \sum_{i=1}^{2} |\delta q_i^K|$$

These node loads are always equal for the respective end nodes of a cotree branch, so for the single loop network and for (K-1)th iteration the above system correction can be re-written as

$$\Delta Q^{K-1} = 2 \times |\delta q_i^{K-1}| \qquad \textbf{(B.10)}$$

Substituting this in equation B.9 , the node load becomes

$$\delta q_i^K = - \frac{f_i(F_{cotree}^K) \times [\Delta Q^{K-1}]}{2 \times |f_i(F_{cotree}^K) - f_i(F_{cotree}^{K-1})|} \qquad \textbf{(B.11)}$$

The absolute term in the denominator term is also same for both nodes *J01* and *J02*, hence the equation can be re-written using summation form as

$$\delta q_i^K = - \frac{f_i(F_{cotree}^K) \times [\Delta Q^{K-1}]}{\sum_{i=1}^{2} |f_i(F_{cotree}^K) - f_i(F_{cotree}^{K-1})|} \qquad \textbf{(B.12)}$$

Using this node load term the system correction for Kth iteration for the single loop network can be evaluated as

$$\Delta Q^K = \sum_{i=1}^{2} |\delta q_i^K|$$

$$= \frac{\{\ |f_{J01}(F_{cotree}^K)| + |f_{J02}(F_{cotree}^K)|\} \times [\Delta Q^{K-1}]}{\sum_{i=1}^{2} |f_i(F_{cotree}^K) - f_i(F_{cotree}^{K-1})|}$$

or

$$\Delta Q^K = \frac{\sum_{i=1}^{2} |f_i(F_{cotree}^K)| \times [\Delta Q^{K-1}]}{\sum_{i=1}^{2} |f_i(F_{cotree}^K) - f_i(F_{cotree}^{K-1})|} \qquad \textbf{(B.13)}$$

The relationship between the individual node load and the overall system correction at Kth iteration can be obtained by dividing equation B.12 by equation B.13, and re-arranging as

$$\delta q_i^K = - \frac{f_i(F_{cotree}^K)}{\sum_{i=1}^{2} |f_i(F_{cotree}^K)|} \times \Delta Q^K \qquad \textbf{(B.14)}$$

These relations can be extended for a multiple loop network having N nodes. In fact only the nodes connected to cotree branches will have non-zero loads i.e. the flow imbalance and others will have zero node loads but for computational ease the summations can be extended to all internal and atmospheric pressure nodes (since these are linked to the system by leak streams which are treated as cotree branches) of pellet induration systems. The general relations for overall system error, $\Delta\ Q^K$, and distributed node load, $\delta\ q_i^K$ give the Boyne's corrections used in section 3.3. These are

$$\Delta Q^K = \frac{\sum_{i=1}^{N} |f_i(F^K_{cotree})| \times [\Delta Q^{K-1}]}{\sum_{i=1}^{N} |f_i(F^K_{cotree}) - f_i(F^{K-1}_{cotree})|} \quad \textbf{(B.15)}$$

$$\delta q_i^K = - \frac{f_i(F^K_{cotree})}{\sum_{i=1}^{N} |f_i(F^K_{cotree})|} \times \Delta Q^K \quad \textbf{(B.16)}$$

The equations B.15 and B.16 form a complete set for the iterative computation. For simplicity of the notation since $f_i(F^K_{cotree})$ is the node error, worked out using Kth iteration flows irrespective of tree or cotree branches. Thus, in the text, the variable f_i^K is used instead of $f_i(F^K_{cotree})$.

To start with (that is for K=1) Boyne suggested to use

$$\Delta Q^1 = \frac{\sum_{i=1}^{N} |f_i^1|}{2} \quad \textbf{(B.17)}$$

where f_i^1 is the node imbalance computed from initial flow values of respective cotree branches.

Appendix - C
Convergence of Primary Solution Algorithm

According to the primary solution algorithm, which is described in section 3.3 and shown in Figure 3.3, the improvement of stream flows in tree branches is continued until the ***Error*** i.e. flow imbalance at each of the internal nodes becomes less than a pre-set tolerance, *Tolrnc*. Mathematically

$$\textit{Error} = \max(f_1, f_2, \ldots f_{N_{int}}) = \max_{i=1}^{N_{int}}(f_i)$$

If the convergence is not achieved then further iterations are terminated after performing certain pre-set maximum number of iterations, say *MaxItr*.

For a realistic simulation the leak areas cannot be opened to full extent in one go, since these may lead to very large flows which can crash the program. Thus, from the initial and final values of respective leak areas the required number of steps (of fixed size) are worked out and the simulation is carried out step-wise feeding the finally converged values of state variables (stream flows and node pressures) of current step as the initial values to the next step plus the new set of leak areas. The nodal errors and overall system error are used to update tree stream flows, until a converged solution is achieved for respective (incremental

leak opening) step. This procedure is continued until final leak areas for all leaks are reached (the computation procedure is explained in section 3.6 and algorithm given in Figure 3.14).

The per iteration values of (1) ***Error*** i.e. maximum of the nodal flow imbalance of all of the internal nodes, (2) name of the node responsible which generated this error, and (3) the overall system error i.e. ΔQ^K by equation B.15 are shown in Block C.1 for one of the steps performed for the simulation of a real-life pellet induration plant whom pressure and flow distributions are shown in Figure 5.5.

Block C.1: The per iteration values of ***Error***, *ResNode*, and ΔQ^K for one of the steps of the simulation

Itr	Error	RNode	DELQK	Itr	Error	RNode	DELQK	Itr	Error	RNode	DELQK
1	12.9	R05	28.0	30	1.93	J08	0.690	59	0.545E-01	R08	0.102
2	48.9	R11	24.9	31	10.8	J07	0.580	60	0.553E-01	J07	0.145
3	330.	J08	19.2	32	3.30	J08	0.175	61	1.66	J08	0.142
4	53.0	J07	4.44	33	0.623	R06	0.594E-01	62	1.79	J07	0.705E-01
5	19.4	R08	4.72	34	0.544	R08	0.119	63	0.489E-01	R08	0.382E-02
6	8.05	R07	2.82	35	0.496	R08	0.892	64	0.397E-01	R08	0.172E-01
7	17.8	J07	1.97	36	1.43	J08	0.974	65	0.138E-01	R06	0.975E-02
8	33.1	J08	1.50	37	16.7	J07	0.886	66	0.644E-01	J08	0.979E-02
9	9.45	J07	0.340	38	4.76	J08	0.228	67	0.127	J07	0.701E-02
10	3.21	J07	0.327	39	0.797	J07	0.520E-01	68	0.324E-01	J08	0.242E-02
11	3.16	R07	1.66	40	0.419	R06	0.409E-01	69	0.142E-01	R08	0.193E-02
12	3.31	J07	2.54	41	0.352	R06	0.223	70	0.136E-01	R08	0.223E-01
13	35.4	J08	2.44	42	0.154	R07	0.259	71	0.351E-01	J08	0.242E-01
14	28.5	J07	1.05	43	1.10	J08	0.263	72	0.400	J07	0.226E-01
15	5.60	J07	0.341	44	4.34	J07	0.204	73	0.168	J08	0.741E-02
16	1.62	R08	0.165	45	0.641	J08	0.368E-01	74	0.110E-01	R08	0.927E-03
17	1.42	R08	0.301	46	0.151	R08	0.181E-01	75	0.103E-01	R08	0.667E-02
18	1.35	R08	0.536	47	0.152	R08	0.997E-01	76	0.932E-02	R07	0.589E-02
19	4.06	J07	0.591	48	0.137	R08	0.312	77	0.323E-01	J07	0.549E-02
20	7.63	R07	0.479	49	1.71	J08	0.313	78	0.626E-01	J08	0.434E-02
21	3.42	J07	0.165	50	5.59	J07	0.237	79	0.357E-01	J07	0.182E-02
22	1.08	R08	0.137	51	0.267	J08	0.157E-01	80	0.559E-02	R08	0.606E-03
23	1.07	R08	0.453	52	0.906E-01	R08	0.983E-02	81	0.518E-02	R08	0.201E-02
24	1.25	J07	0.928	53	0.880E-01	R08	0.520E-01	82	0.505E-02	R08	0.868E-02
25	8.42	J08	0.908	54	0.828E-01	J07	0.113	83	0.281E-01	J07	0.891E-02
26	11.5	J07	0.464	55	0.934	J08	0.112	84	0.162	J08	0.778E-02
27	1.35	R06	0.104	56	1.69	J07	0.717E-01	85	0.299E-01	J07	0.134E-02
28	0.918	R06	0.167	57	0.846E-01	J08	0.720E-02	86	0.383E-02	R08	0.307E-03
29	0.879	R06	0.610	58	0.569E-01	R08	0.808E-02				

In GASFLO, the *Tolrnc* is chosen as 5.0E-03, since the nodal errors i.e. flow imbalance at internal nodes is worked out in Kilogram per second (Kg/sec) whereas at actual plants the airflows are measured in tonnes per hour (tph), and with maximum accuracy of ±1.0 tph. The specified tolerance is less than 0.02 tph which is sufficiently small to produce reasonably good physical results.

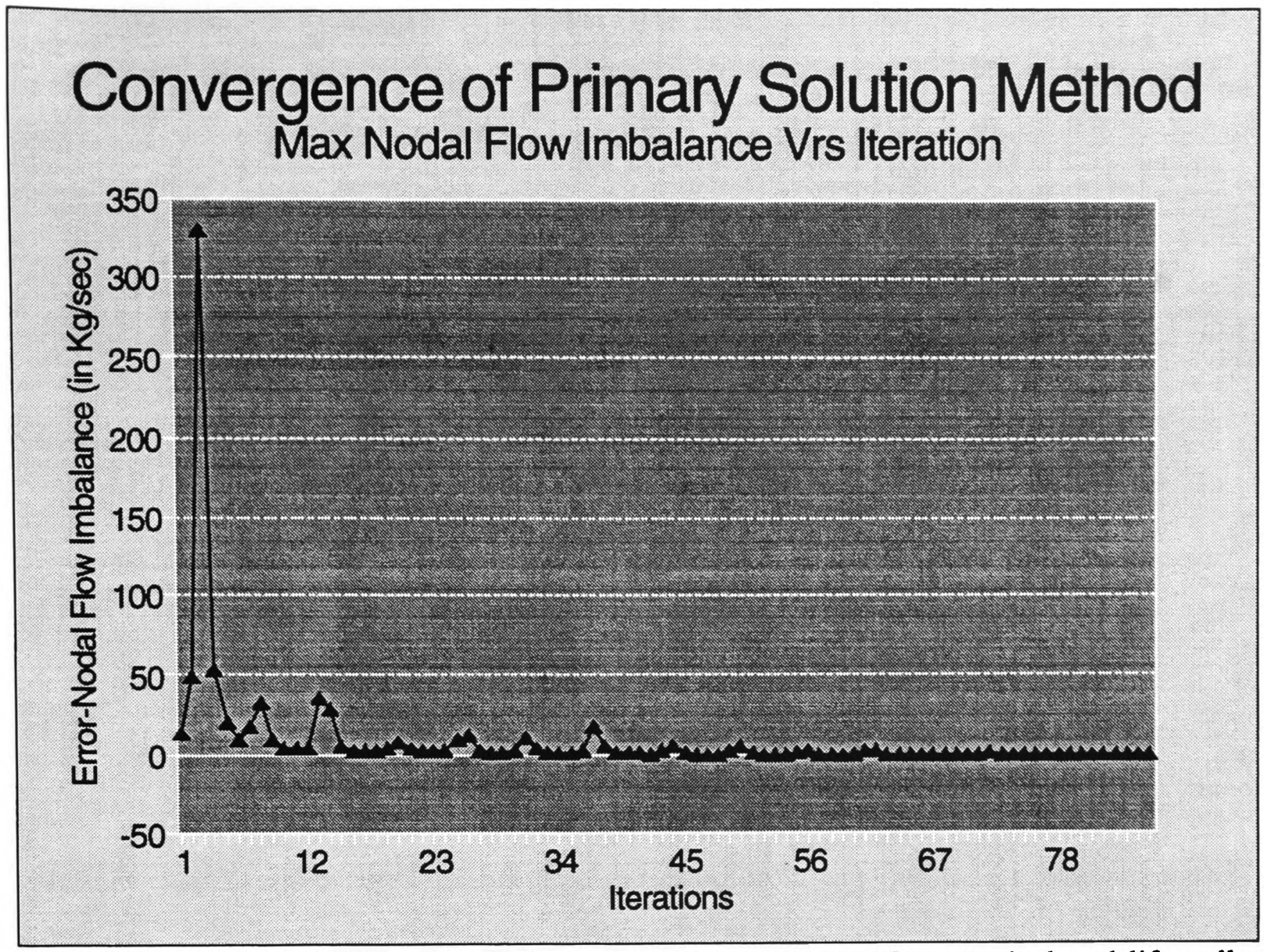

Figure C.1 The behaviour of maximum nodal error, ***Error***, for a typical real-life pellet induration system network

Figures C.1 show the behaviour of ***Error*** i.e. the maximum nodal error with respect to successive iterations. The system effectively converges in 45 to 50 iterations. Initially the system has smaller error but it increases significantly in next few iterations to accommodate the new leak flows and it dies down as the tree flows are accordingly adjusted. Some visible oscillations are due to the dependence of error on different nodes as each iteration may be due to a different internal node (see Block C.1). However, the magnitude of the error decreases on successive iterations. The nodal errors could be either positive or negative, but the ***Error*** being the maximum, always has positive value, as nodal errors alternate sign at the opposite ends of the respective cotree branch, and thus there always exist some positive value (see Appendix-B).

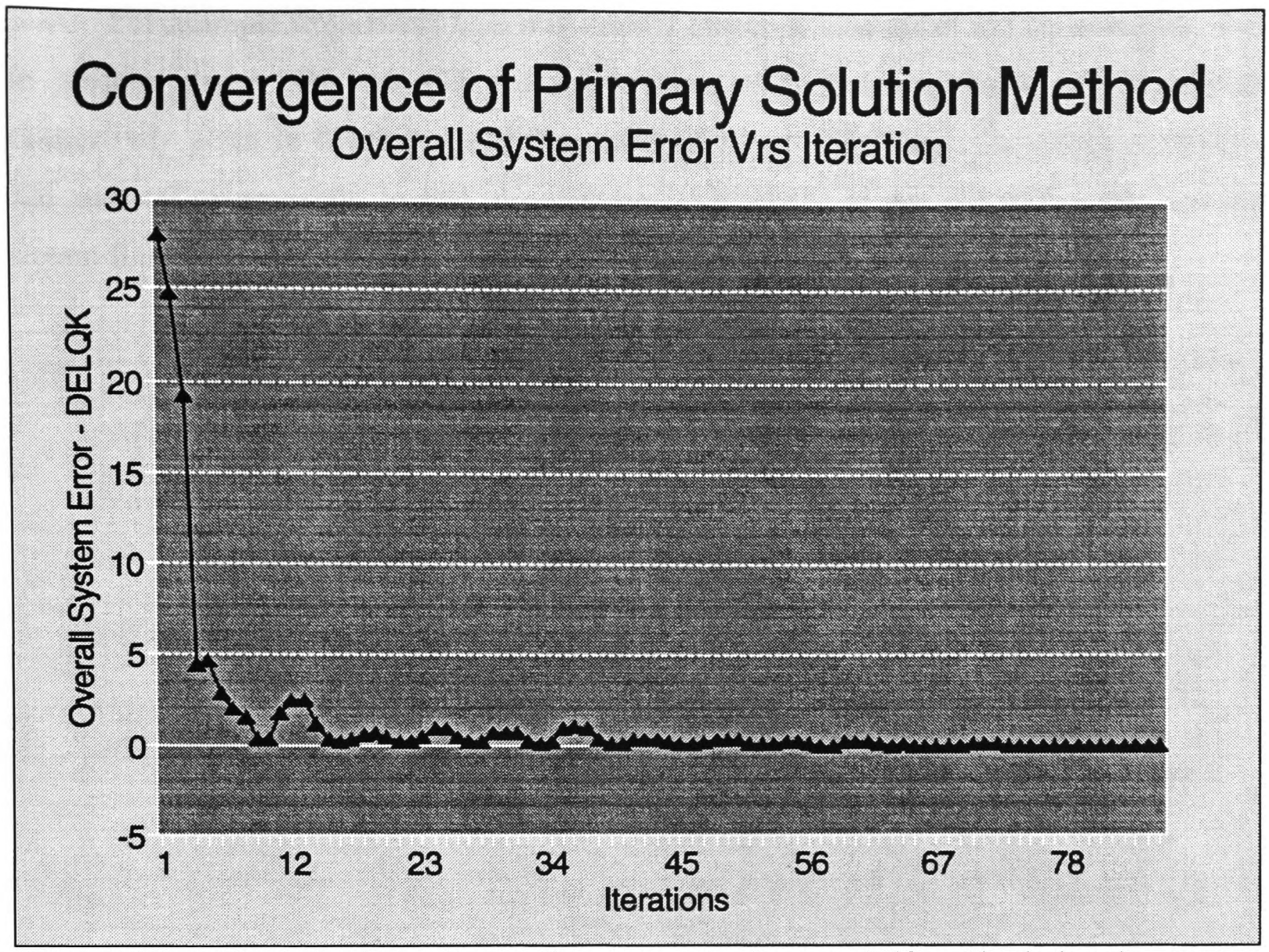

Figure C.2 The behaviour of overall system error for a real-life pellet induration system simulation

Figure C.2 shows the overall system error ΔQ^K, being some aggregate function of all nodal values; it is positive, less violent and more smooth than the ***Error***. Like Figure C.1, it shows that the system converges in about 45 to 50 iterations.

The spreadsheet compatible debugging files for GASFLO, mentioned in section A.3.2 and briefly described in Figure A.3a, provide an excellent facility to monitor and study any of the system variables at run time. The desired node pressures and errors, and stream flows etc can be selected at run time through data input file for any step and for any range of iterations. Later these files can be imported into any spreadsheet package (e.g. Lotus 1-2-3, VP Planner, Quartro Pro or Microsoft Excell) and the respective variables can be analyzed using the graphical facilities provided by the package. The graphs presented in this appendix are generated using this facility.

For example from Block C.1, it is evident that error at some of the iterations was due to junction nodes *J08* and *J07* and region node *R09*. Referring back to the network connectivity given in Block A.4 and illustrated in Figure A.5, these nodes connect streams *s26* and *s07* respectively. In fact, the junction node *J08* is at the exit of fan 1A, and the stream linking it to node *J07* is *s26*.

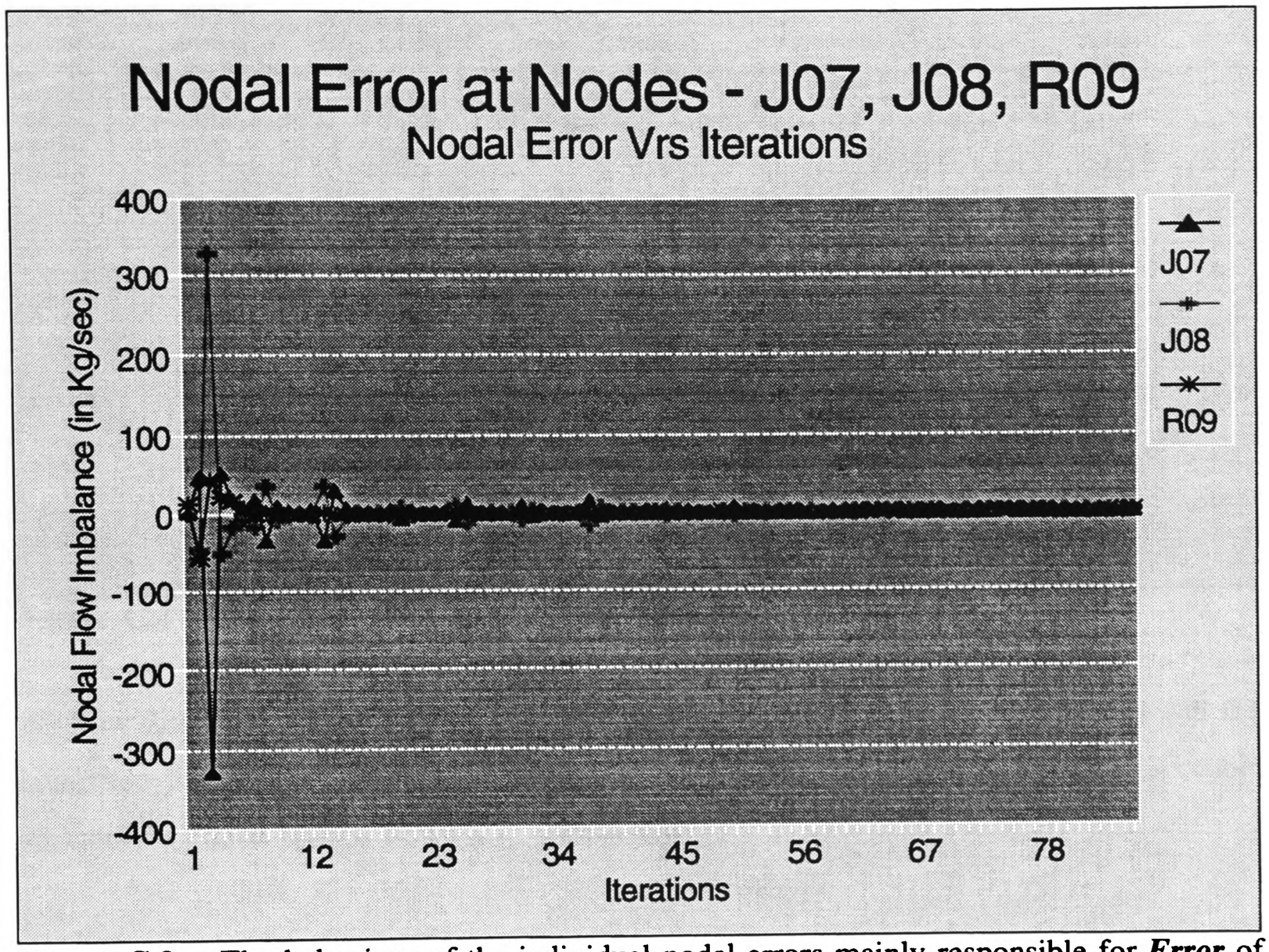

Figure C.3 The behaviour of the individual nodal errors mainly responsible for ***Error*** of the system

Figure C.3 shows the behaviour of individual errors at nodes *J08*, *J07* and *R09*. The disturbing network component could also be tracked down from these monitored variables.

Figure C.4 shows the behaviour of flows in streams *s26* and *s07*. It should be noticed from the network connectivity that *s26* is a cotree branch, and its flow is computed from the respective end-node pressures i.e. pressures at nodes *J08* and *J07*. This shows that although

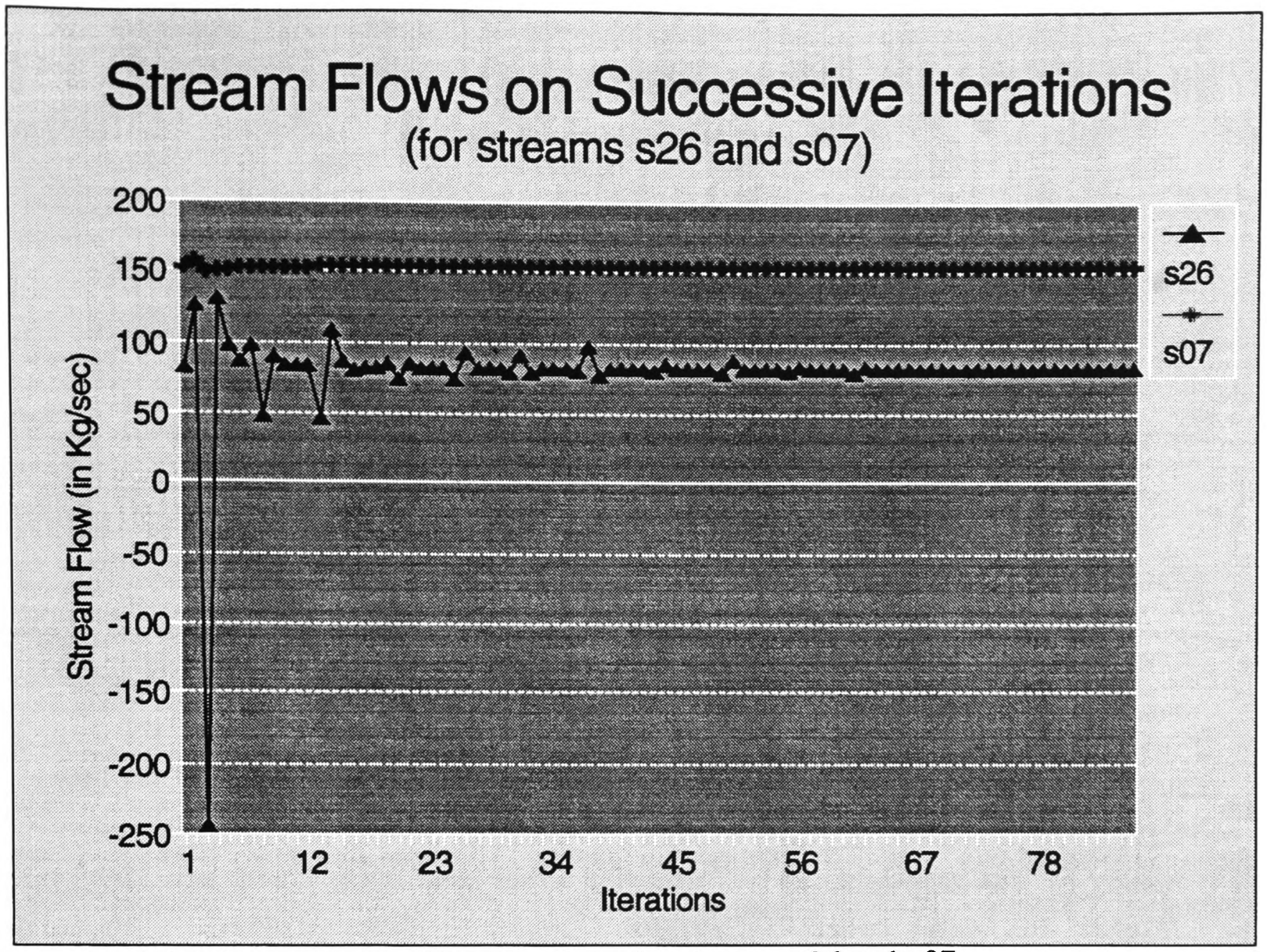

Figure C.4 Improvement of stream flows for streams *s26* and *s07*

the flow distribution in tree branches is comparatively smooth (i.e. flow in *s07*) but still the computed junction pressures at *J08*, *J07* are appreciable enough to generate a noticeable variation in cotree branch flows (e.g. stream *s26*).

It is noticed that at initial steps, the system needed more iterations as compared to later steps. Since in the later steps the converged values of system variables are used from the previous steps.