1984

Automatic Generation of Data Flow Diagrams From A Requirements Specification Language

L. B. Protsko
University of Saskatchewan

P. G. Sorenson
University of Saskatchewan

J. P. Demblay
University of Arizona

Automatic Generation of Data Flow Diagrams From A Requirements Specification Language

L.B. Protsko
P.G. Sorenson
Department of Computational Science
University of Saskatchewan

J.P. Tremblay
Department of Management Information Systems
University of Arizona

ABSTRACT

Escalating manpower costs in developing systems has caused an increasing need for greater productivity in system development, particularly in the analysis and design phases. Productivity in the system analysis phase can be increased with the use of computer-aided tools such as SPSL/SPSA for specifying system requirements and methodologies such as structured analysis. A structured analysis and documentation tool—the data flow diagram—allows an analyst to model and document a system with relative ease; however, the manual production of a data flow diagram is a time consuming process. Combining the production of data flow diagrams with SPSL/SPSA produces a synergistic effect on the increases in productivity and ensures the use of standards and the completeness of the diagram. This paper describes the problems and design of the system MONDRIAN that generates data flow diagrams from an SPSA database. A variety of placement and routing algorithms that address the layout problem are discussed. The results of a preliminary study of the effectiveness of these algorithms and the adaptations required to improve and refine the prototype version of MONDRIAN are presented.

Introduction

As improvements in productivity in later phases of the system life cycle are achieved, the need increases for computer-aided analysis tools which improve productivity in the analysis phase. This paper is concerned with the development of such a tool.

A structured analysis and documentation aid—the data flow diagram (DFD)—allows an analyst to model and document a system easily. However, producing a layout for a data flow diagram manually is a time consuming problem. An analyst's productivity can be increased if a data flow diagram were automatically produced from the logical structure of the requirements definition. One facility for producing such a definition is SPSL/SPSA, meaning Simple Problem Statement Language/Simple Problem Statement Analyzer (Friesen *et al.*, 1981). SPSL/SPSA is a structured analysis and documentation aid that allows a user to define formally the logical structure and requirements of an information system, store this description in a database, and later produce from the database various documents and reports about the system structure and requirements.

SPSL/SPSA is essentially a subset of the PSL/PSA system that has been developed as part of the ISDOS project at the University of Michigan (Teichroew and Hershey, 1977). The defined subset was derived based on an analysis of the most popular features of PSL/PSA that were used by practicing systems analysts (Perkins, 1979; Wig, 1979(a)). Another major project that has stressed the development of computer-aided analysis tools is the PLEXSYS project at the University of Arizona (Nunamaker and Konsynski, 1981). Both the ISDOS and PLEXSYS projects have examined the problem of integrating structured systems analysis methodologies (e.g., Gane and Sarson, 1979, and DeMarco, 1979) with their systems. Neither project has handled the problem of automatically producing data flow diagrams in a satisfactory manner.

This paper addresses the problem of automatically generating a layout for a data flow diagram from an SPSA database. A system called MONDRIAN (Protsko, 1983) accesses an SPSA database to retrieve system flow information and generates the data flow diagram for the system. This paper outlines the design, and implementation of placement and routing algorithms to be

used in MONDRIAN. It investigates the adaptation of placement and routing techniques used in design automation for producing a layout suitable for DFDs.

It is important to list the benefits of computer-draw DFDS.

1. The time saved between manual and automated production of a DFD is in the order of many hours. For example, a DFD for the anaesthesia department example which is presented later in the paper, took several hours to layout (more time would be needed to prepare a finished product) as opposed to an estimated 25 minutes to produce a finished product using MONDRIAN. Of this estimated time, only a few minutes were required to initiate the steps thus allowing the analyst to perform other tasks while the DFD was being generated.

2. As the complexity of the system increases, the frequency of errors and omissions also increases when manually drawing a DFD. By relying on the SPSL/SPSA system to provide completeness and consistency checks, the computer-drawn DFDs should not be subject to such errors and omissions.

3. A set of guidelines or standards for drawing DFDs are incorporated into the automated package thus ensuring their usage.

Data Flow Diagrams

A data flow diagram is a structured systems analysis and documentation aid that provides a pictorial description of the flow of data through a system (Gane and Sarson, 1979). DFDs implicitly define the boundary of the system by showing the sources and destinations of data (i.e., the system's interfaces). The logical functions, the data that flow between functions, and data stores or repositories are also identified and named in a DFD.

An example data flow diagram, produced by MONDRIAN, is given in Figure 1. It is assumed in this paper that the reader is familiar with the use of data flow diagrams in conjunction with structured systems analysis techniques. For those not familiar, Gane and Sarson (1979) or DeMarco (1979) provide excellent references.

SPSL/SPSA

SPSL/SPSA (Friesen *et al.*, 1981) is a subset of the PSL/PSA package described by Teichroew and Hershey (1977) which has been developed at the University of Michigan as a system to aid in analysis and documentation. SPSL is a nonprecedural language used to define formally the logical structure and requirements of an information processing system. It permits the description of both system-related and data-related information in the form of system flow, structure, size, and dynamics, as well as data structure and data derivation information. SPSA is a software package which accepts SPSL statements as input, translates these into an internal form that is stored in a database, and generates various documentation and analysis reports.

A description of an information processing system is based on language components, called *objects*, and the *relationships* between these components. The analyst creates an object and relates it, via relationship specifications, to the rest of the system in a unit description called a *section*. In general, the ordering of sections is not significant and, therefore, SPSL is nonprocedural.

Some of the features of SPSL are shown through the use of an example which represents a portion of a hospital's anaesthesia department. Such an example is given in Figure 2. Object names are in lower case, while SPSL reserved words are in upper case. In the following discussion of SPSL components related to system flow, references to the SPSL example (Figure 2) are in the form of parenthetical comments. Each such comment contains line numbers which refer to the specific lines of the example that contain the language components. A more detailed description of these and other SPSL components not related to system flow and the production of DFDs is found in Friesen *et al.*, (1982).

System flow is expressed in SPSL by defining system components; these can be one of six types of objects, namely a *process* (lines 15, 21, 25), and *interface* (1, 3, 5, 9, 11), an *entity* (14, 20, 24, 34), and *input* (36, 37), an *output* (13, 33), and a *set* (28, 31). An *interface* represents a unit in the real world that interacts with the system. An *entity* represents a logical information unit that is maintained by the system and stands for a real world object such as an employee or a part. A *set* represents a collection of one or more types of logical information units (i.e., entities). In order to describe system flow, nine relationships are used: *generates* (2, 4, 6, 7, 8, 26), *receives* (10, 12, 18, 19), *references* (27), *creates* (16), *uses* (17, 23), *adds* (22). The *modifies*, *updates*, and *removes* relationship are not illustrated in this small example.

The primary functions of the SPSA software system are threefold:

1. It provides a facility to compile SPSL statements and store an equivalent representation of the problem statement in a database.

2. It enables modification of a problem statement stored in a database.

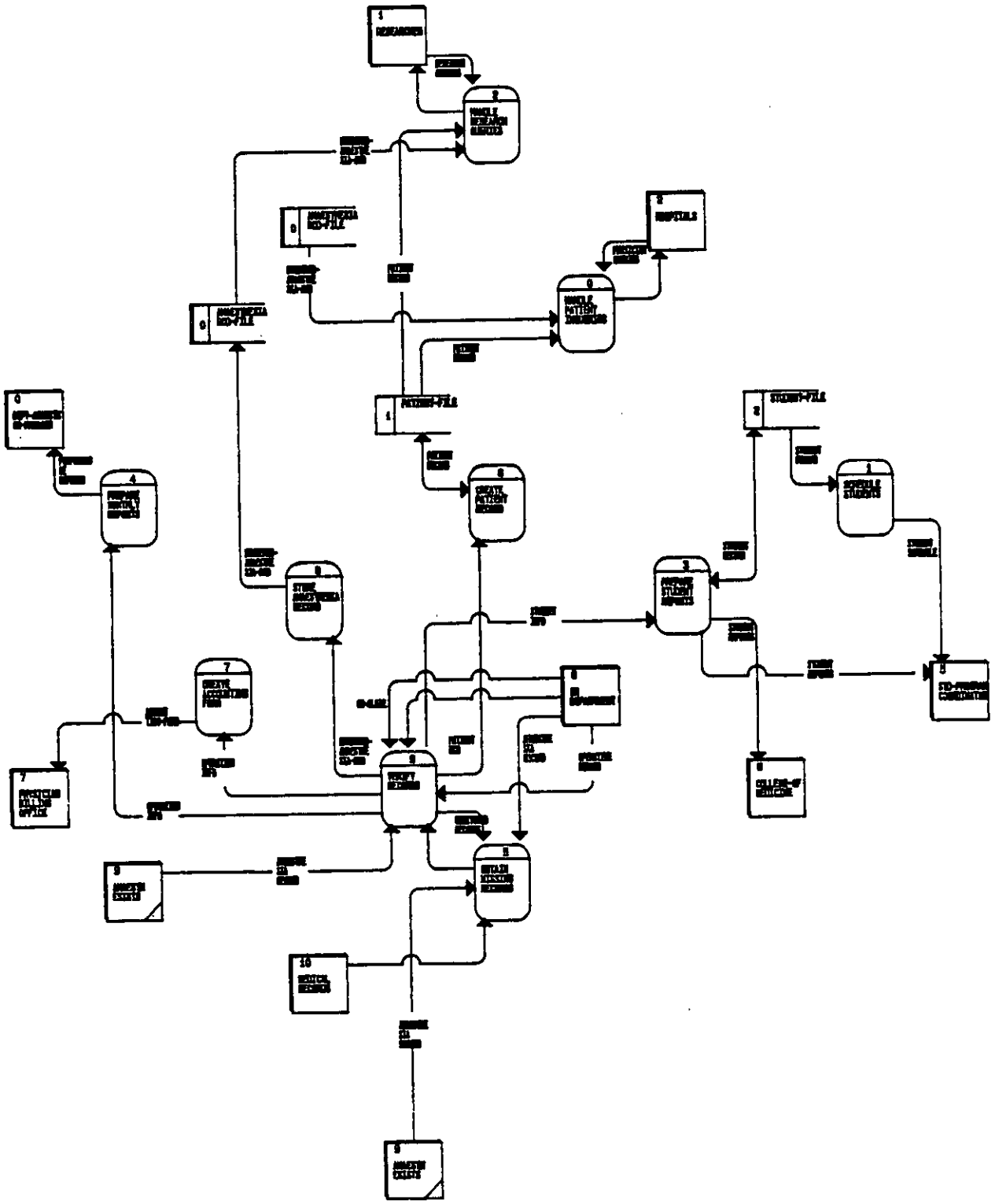


Figure 1

DFD for Anaesthesia Department Example

```
1>  DEFINE INTERFACE medical-records;
2>      GENERATES patient-info;

3>  DEFINE INTERFACE anaesthetist;
4>      GENERATES anaesth-record;

5>  DEFINE INTERFACE or-department;
6>      GENERATES or-slate;
7>      GENERATES operative-record;
8>      GENERATES anaesth-record;

9>  DEFINE INTERFACE physician-billing-office;
10>     RECEIVES accounting-form;

11> DEFINE INTERFACE std-program-
      coordinator;
12>     RECEIVES student-schedule;

13> DEFINE OUTPUT student-schedule;

14> DEFINE ENTITY unmatched-records;

15> DEFINE PROCESS obtain-missing-records;
16>     CREATES matched-records;
17>     USES unmatched-records;
18>     RECEIVES patient-info;
19>     RECEIVES anaesth-record;

20> DEFINE ENTITY matched-records;

21> DEFINE PROCESS store-anaesth-record;
22>     ADDS numbered-anaesth-red TO
          anaesth-red-file;
23>     USES numbered-anaesth-red;

24> DEFINE ENTITY numbered-anaesth-red;

25> DEFINE PROCESS schedule-students;
26>     GENERATES student-schedule;
27>     REFERENCES student-record;

28> DEFINE SET anaesth-red-file;
29>     CONSISTS numbered-anaesth-red;
30>     ORDERED BY anaesth-number;

31> DEFINE SET student-file;
32>     CONSISTS student-record;

33> DEFINE OUTPUT accounting-form;

34> DEFINE ENTITY student-record;
35>     IDENTIFIED BY student-name;

36> DEFINE INPUT patient-info;
37> DEFINE INPUT anaesth-record;
```

Figure 2

SPSL Extracts from Anaesthesia Example

3. It contains facilities to produce reports from a problem statement stored in an SPSA database and to determine if a problem statement is complete. The five reports that are produced are described by Wig (1979(b)).

The interface to SPSL/SPSA is dependent upon the nature of the installation, i.e., interactive or batch. The present implementation runs under the UNIX operating system and supports both types of interfaces. A detailed tutorial description of SPSL/SPSA is given by Friesen *et al.*, (1982).

Relationship Between SPSL/SPSA and DFDs

Objects and relationships used in SPSL to describe system flow can be equated to the components of DFDs. SPSL's interface, process, and set are equivalent to the external entity, process, and data store in a DFD, respectively. A DFD data flow may be equated to the SPSL input, output or entity. The SPSL relationships (generates, receives, references, adds, removes, creates, and uses) indicate the direction of the data flow in DFDs. The generates, adds, and creates relationships indicate an outward flow, whereas the receives, uses, removes and references relationships involve an inward flow. The update relationship, used to describe data derivation in SPSL, indicates a bidirectional flow between objects when applied to DFDs. A detailed discussion of the use of structured systems analysis in conjunction with SPSL/SPSA is provided by Channen *et al.*, (1984).

In addition to a direct equivalence between SPSL system flow descriptors and DFD symbols, SPSL/SPSA and DFD can each be used in the development of the other. A high-level DFD created initially to describe a system can be used as a starting point for the creation of an SPSL description. Further analysis would typically involve the completing of a more detailed SPSL/SPSA description. After using the SPSA report facilities to ensure completeness and consistency, a detailed DFD can then be created automatically by computer from the SPSA database.

Requirements for Data Flow Diagram Layout

In addition to the obvious goal of accuracy, an overall objective when creating DFDs, either manually or automatically, is *clarity* (i.e., readability) of the diagram. Clarity is not easily achieved, particularly when describing large systems. The main reason is the difficulty of identifying important characteristics of an easy-to-perceive diagram. A balance must be achieved between the symbols, text, and white space in a DFD. Arcs for data

flows connecting interfaces, data stores, and processes, must be long enough and sufficiently spaced to identify the name of a data flow; however, they must not be so long that the components connecting the arc are erroneously thought to be separated logically.

When developing MONDRIAN, criteria regarding replication, crossovers, length and spacing of data flow arcs and positioning of names of data flows had to be established. *Replication* is only valid for data stores and interfaces, as the replication of processes may give a false impression of the system being described, e.g., a replicated process may be thought of as two separate functions. This can cause particular problems if transformation or transaction analysis (Yourdon and Constantine, 1979) is performed because the immediate recognition of the function may be lost. For example, if the input(s) to a process are shown leaving a second icon for that process, the relationship between the input (s) and output(s), as well as the function of the process may not be understood.

Replication can be used to achieve clarity by reducing the lengths of data flow arcs, increasing the spacing between data flow arcs, minimizing the number of crossovers, and/or minimizing (or eliminating) the number of wandering arcs (arcs with excessive turns and detours between their sources and destinations). Although replication can be a powerful tool in achieving clarity, its overuse can be a hindrance to clarity. Excessive replication can result in complicated searches for replicated interfaces or data stores that may be more tedious than following long, wandering, or crossing data flow arcs.

Criteria for replications include:

1. A maximum number of data flows may originate from or terminate at any one interface or data store (currently set at 12 and 7 respectively). After this maximum has been reached, replication of that icon must occur.

2. Replication should be used when a data flow cannot be drawn without violating the criteria regarding length, crossovers, or wandering arcs.

If a process has more than a maximum number of data flows associated with it (currently set at 12), it is recommended that the description of the process be expanded to a set of lower level descriptions.

Crossovers aid clarity by allowing interrelated icons to be positioned in groups, thus giving a more accurate representation of the system being described. As with replication, excessive crossovers can detract from clarity. Criteria regarding crossovers include:

1. Crossovers should not be made over any type of icon or the name of a data flow arc.

2. There should be an upper limit to the number of crossovers that occur in a data flow diagram. This maximum should be related to the size and complexity of the system being described.

The *length of data flow arcs* also affects clarity in the sense that very short arcs do not allow enough room to position data flow names properly on an arc. Short arcs can also result in the excessive clustering of the icons. Data flow arcs which are very long can complicate the process of determining where the flow originates and terminates.

The following criteria regarding length of data flow arcs were formulated:

1. The distance between icons should be sufficiently large such that the arc's name cannot be associated incorrectly with any other arc and that the arrowhead and arc are clearly visible.

2. Data flow arcs should not be longer than the equivalent of one page length.

Names of data flow arcs are placed horizontally beside the arc. Positioning data flow arc names such that the name can only be associated with the arc is achieved by left or right justifying the name next to the arc or by breaking the arc and centering the name between the arc segments. It is felt that clarity is severely inhibited if an arc has more than two turns, and consequently, replication is adopted whenever possible to avoid arcs with excessive turns.

A General Architecture of MONDRIAN

MONDRIAN accesses an SPSA database to produce an adjancency list containing the system flow components of the database. The head nodes of the adjacency list represent the process, data store (set), and interface objects. The list nodes represent a data flow between the corresponding head nodes.

To record the placement and connection of objects, three grid systems are used. The first, the *main grid*, is used to record the placement of objects. The information contained in this grid is the object's name, type, identification number, and the number of times the object is replicated. The second grid, the *subgrid*, represents 5 X 5 subdivision of each square of the main grid. The subgrid is used to record the details of the connections (data flows) between objects. The dotted lines of Figure 1 represent the main grid for the DFD contained in that figure. The solid lines in the upper lift corner of the diagram represent the subgrid.

To account for the spacing requirements of data flow diagrams, a modification in the use of the main grid structure is required. For example, rather than allowing icons to be placed in every grid square, a template overlaying the main grid indicates which squares may be used for icon placement. The template is created by repeating the pattern shown in Figure 3 beginning in the upper left corner of the main grid. A square of the main grid on which an "X" lies is a square in which an icon can be placed. Except for those positions around the perimeter of the grid, this approach allows sixteen connecting positions for each placed node. The positions around the perimeter of the grid have at least seven of the sixteen connecting positions. No icon can have more than twelve arcs incident to it; therefore, the sixteen reachable positions should be sufficient to find positions for nodes connecting to any one icon.

The template is compressed to create a new grid structure, named the *workspace*, by dividing the grid into segments of two squared by two squares such that each segment contains one "X" of the repeating pattern (see Figure 3).

Four distinct segment types, based on the position of the "X" within the segment, are formed by this division. An array containing the relative x and y coordinates required to reach each of the sixteen alternative positions in the workspace grid is created for each of these types. The workspace square contains an indicator of its type and a value used as an index into the corresponding alternatives array. Figure 4 shows one of the segment types and its alternative positions.

MONDRIAN has three main functions which are to:

1. Extract and store information about the system,
2. Process that information to produce a layout of the DFD in the grid system, and
3. Use the grid system to produce the final copy of the DFD.

The high-level structure chart that provides an overview of the architecture of MONDRIAN is shown in Figure 5.

Module CREATE_ADJACENCY_LIST of Figure 5 is responsible for extracting the required information about the system being analyzed by accessing an SPSA database and storing it in an adjancency list. In this adjacency list a head node represents a process, data store, or interface. Each head node has a set of list nodes. Each list node represents a data flow which either originates or terminates at the object represented by the head node. A list node contains information about the data flow such as name, direction of flow, and a list of possible paths.

Figure 6b shows a small segment of the adjacency list for lines 11, 12, 13, 25, 26, 27, 31, 32 and 34 of the SPSL description given in Figure 2 and the data flow diagram shown in Figure 6a. As each segment of the SPSA object table is accessed, a head node is allocated and linked into the adjacency list. After all the head nodes have been allocated, list nodes are created for each data flow in the system. As shown in Figure 6b, two list nodes are created for each data flow. One is placed in the set belonging to the "originating" head node, while the second is placed in the set belonging to the "terminating" head node. For example, in Figure 6b the list node "student-schedule" is contained in the set for head node "std-program-coordinator" as an input and in the set for "schedule-students" as an output.

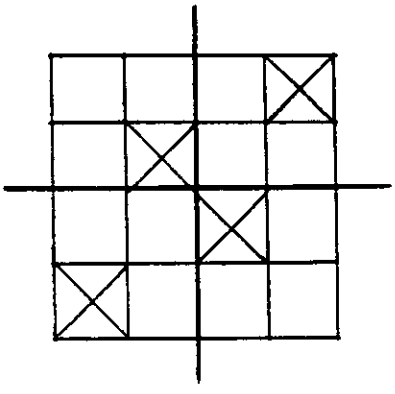


Figure 3

Repeating Pattern and Division into Segments

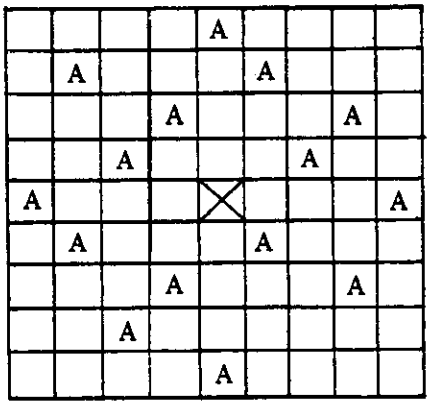


Figure 4

Type I Segment and Its Alternative Positions

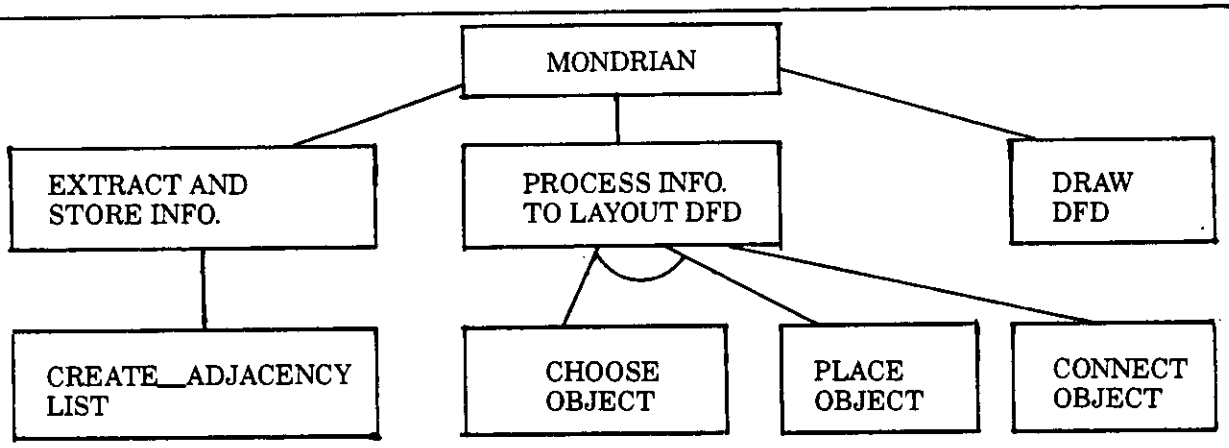


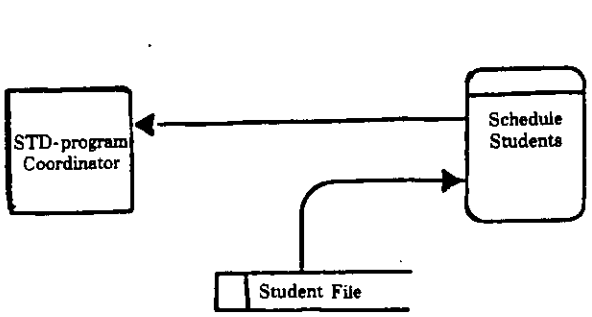
Figure 5

High Level Structure of MONDRIAN

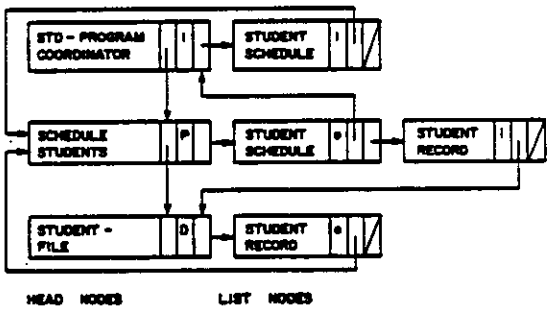
A link is established from the list node to the second head node (that is, if the list node is in the set belonging to the "originating" head node, a link to the "terminating" head node is made). This allows the placement of objects to be made in any order and connections to be made as each object is placed. In some situations different SPSL statements will produce identical list nodes (that is, they represent the same being passed in the same direction between two objects). An example of this occurs when a process creates, adds, and references the same entity. If no further processing is done, three identical data flows would exist. Therefore, after the adjacency list is completed, it is traversed and all redundant list nodes are eliminated.

When redundancy has been eliminated, the information contained in the adjacency list is processed to produce a layout of the DFD in the grid system (Figure 5—PROCESS_INFO_TO_LAYOUT_DFD).

An initial implementation of the modules CHOOSE_OBJECT, and CONNECT_OBJECT followed the guidelines outlined in Section 2. The placement algorithm (PLACE_OBJECT) did not make use of the workspace grid and therefore relied solely on the calculation of positions to create white space. Routing (CONNECT_OBJECT) was carried out as each icon was placed. Several problems became clear; the routing and labelling of some data flows lead to the inability of



(a) Data Flow Diagram



(b) Partial Adjacency List

Figure 6

Data Flow Diagram and Partial Adjacency List

routing other data flows and excessive spacing at the edges of the diagram and insufficient spacing in the center of the diagram. These problems lead to an investigation of the placement and routing techniques used in design automation. The next section discusses these techniques and their applicability to the placement and routing of DFDs.

The final module (module DRAW_DFD) represents the actual drawing of the DFD on a screen or hardcopy device. The completed grids, along with the size of the main grid that is used, are passed to the graphics package which, in the original implementation, produced the DFD on a Versatec printer/plotter. Current versions produce the DFD on a Ramtek graphics terminal, Ramtek printer, AED graphics terminal and an HP plotter. The DFD shown in Figure 1 was produced on an HP plotter.

Survey of Placement and Routing Techniques

The area of design automation has produced many placement and routing techniques for the automated generation of circuit board layouts. A survey of these techniques was undertaken by Protsko (1983) to determine which, if any, techniques could be applied to the production of DFDs. This section briefly describes this survey. Algorithms involving multiple layers were not included due to their apparent unsuitability for DFDs which have only one layer.

PLACEMENT

The placement problem consists of determining the optimal positioning of interrelated entities to fixed locations contained in a given area subject to a set of constraints. A well known mathematical problem related to the placement problem is concerned with sets of points, whereas the quadratic assignment problem is concerned with pairs of points. If all signal sets (set of points to be connected) of a placement problem contain exactly two modules, the placement problem reduces to the quadratic assignment problem. Even if the signal sets are not uniformly of cardinality two, an associated assignment problem can be derived from the placement problem (Hanan and Kurtzberg, 1982). This reduction allows any method of solution for the quadratic assignment problem to be used for the placement problem. The quadratic assignment problem has been shown to be NP-complete (Garey, Johnson and Stockmeyer, 1974). Therefore, heuristic methods are used to approximate an optimal solution.

Hanan and Kurtzberg (1972) classify the approaches to the placement problem into three classes, based upon the heuristic used:

1. Constructive initial-placement,
2. Iterative placement-improvement, and
3. Branch and bound.

The algorithms in the first and third classes are "constructive" while those in the second are "iterative". The iterative algorithms seek to improve a placement by repeated modifications. At every stage there is a complete placement available (that is, every element is placed). The constructive algorithms produce a placement layout only on termination.

The survey conducted in Protsko (1983) included algorithms from each of three classes. In particular, algorithms from the constructive initial placement class consisted of a pair-linking algorithm and a cluster-development algorithm (Kurtzberg, 1965). The iterative placement development algorithms consisted of a pair-wise iteration algorithm (Steinberg, 1961), two relaxation methods (Fisk, 1967; and Quinn, 1979) and the Monte Carlo approach of Hanan (1972). The branch and bound algorithms consisted of the matrix-scan and row-scan approaches of Gilmore (1962).

In general, the order of growth for the heuristic methods ranges from n^2 for the pair-linking and cluster-development and for some of the interchange techniques, to n^4 for the approximation of the branch and bound techniques. However, the order of growth does not identify the amount of processing time a specific problem requires or the size of problem the various methods can conveniently handle. For example, the constructive methods can handle far larger problems than the complete interchange techniques even though they are both of order n^2 (Hanan and Kurtzberg, 1972).

Hanan and Kurtzberg recommend the pair-linking or cluster-development methods for securing an initial placement, along with an iterative placement-improvement method such as pairwise interchange or Steinberg's assignment method. The pair-linking method produces a better layout (as far as circuit layout is concerned) than the cluster-development method, but requires more processing time. Therefore, the choice of method should be based upon the layout requirements and the processing time available. We elected to adapt the pair-linking and the cluster-development algorithms for DFD placement.

ROUTING

The four subproblems within the wiring problem are wire list determination, layering, ordering, and wire layout. The wire list determination subproblem creates a listing of precisely *what* wires are to be laid out. The

layering subproblem decides *where* (on which layer) each wire should be located. Ordering decides *when* each wire assigned to a layer will be laid out. The wire layout subproblem determines *how* each wire is to be routed.

In rectilinear wiring, the pairs of points to be connected are grid points. Wires must run along grid lines and no two wires can share the same grid segment. No wire can cross a grid point that is to be connected to another grid point. These constraints often require that multilayered boards be used. The determination of the minimum number of layers needed to wire a given wire set in a rectilinear fashion has been shown to be NP-hard (Raghaven, 1981).

Another wiring problem is that of wiring a given wire set on a grid subject to the additional constraint that no wire may bend more than k times (Pomentale, 1965). When k is equal to one, this problem is known as Manhattan wiring. Raghaven *et al.*, (1981) have developed on $0(n^3)$ algorithm which determines if a given wire set is Manhattan wireable in one layer. It also produces the layout within the stated time bound, if it is Manhattan wireable on one layer. They also show that the determination of the minimum number of layers needed for Manhattan wiring is NP-hard.

In the portion of the survey dealing with routing, wirelist determination, ordering, and wire layout were discussed. Layering was only briefly mentioned as it is not applicable to DFD production. Wirelist determination approaches such as minimal spanning tree (Loberman, 1957), minimal steiner tree (Hwang, 1979) and travelling salesman (Breuer, 1972) were described. The wire layout approaches that were discussed included Lee's algorithm and its variations (Lee, 1961; Fisk, 1967; and Breuer, 1972) as well as an approach to one layer Manhattan wiring (Raghavan, 1981). Raghavan's approach also included ordering of wires. Also, an approach to wire layout using a limited backtracking algorithm for 2-CNF-satisfiability (Even *et al.*, 1976) was discussed.

Lee's algorithm (1961) addresses the issue of wire-layout, but not ordering. In particular, it does not take advantage of the fact that the position of the nodes are known. It was felt that Lee's algorithm would not produce significantly different routing than would the algorithm used in the initial prototype with an ordering routine. Because of this similarity and the large amount of processing time required, Lee's algorithm was not considered for evaluation as a DFD routing algorithm.

Raghavan's algorithm addresses both the ordering and wire layout problems. Although this algorithm is not designed to handle crossovers and replication, it can be modified to do so. An important feature of Raghavan's algorithm which is needed for DFD routing is the ordering of the wire layout. The nature of the upper and lower paths used in MONDRIAN is similar to that of Raghavan's upper and lower paths. However, the selection of paths is more complex due to the greater number of upper and lower paths for each icon. Raghavan's algorithm does not allow crossovers and replication. The combination of ordering and wire layout in one algorithm was the main reason for including this algorithm in the evaluation.

RELATIONSHIP OF CIRCUIT LAYOUT PROBLEM TO DATA FLOW DIAGRAM PRODUCTION

The purpose of any placement technique is to produce the desired layout. The desired layout for design automation placement (modules, boards or sets of boards) and data flow diagrams are quite different. One of the most frequently used constraints in design automation placement is minimizing the total wire length. This constraint tends to cause a clustering of nodes. Often, this clustering takes the form of every slot (cell) on the board being occupied. This heavy clustering, considered valuable in design automation, is not desired in DFDs. The difference between the white space required in DFDs and circuit boards is based upon the intended use of each product. The design automation product is to transmit electrical signals, whereas the data flow diagram is to convey information about a system in pictorial form. As a consequence, the most effecient design automation placement techniques (that is, ones which produce the heavy clustering), are not suitable for the productions of DFDs without substantial modification.

Another major difference between the two areas is the meaning that can be derived from the interconnection arc. The connections in the circuit layout problem are not differentiated nor directed as they are in DFDs.

The boards (planes) used in the circuit layout problem are often multilayered or are two-sided, whereas the data flow diagram is contained on a single plane. Thus, the approaches which assume a multiplane environment are not appropriate for the production of DFDs.

The data flow diagram has some features which are not included in the circuit layout problem. One such feature is the labelling of the data flow arcs. Labelling could be incorporated into a standard routing algorithm by adjusting the incremental distance of the grid to include room for the label. This, however, would require very large icons to be used or dramatic reduction of the number of arcs incidental to any one icon.

Another feature of DFD production which is not present in design automation methodology is the replications of interface and data store icons. One advantage of replica-

tion is that if a path cannot be found for an interconnection (and if one icon represents a data store or interface), a replicated version of that icon can be placed in a grid position so that the connection can be made. The use of replication increases the difficulty of the procedure as decisions must be made about when to stop trying to find a path and replicate the node and to which icon position future arcs are to be connected.

A feature which is allowed in some routing methods and data flow diagrams (but not in Mahattan or recticlinear wiring) is the crossover. As in the methods which allow crossovers, their use in the production of DFDs should be minimized. The fact that Manhattan wiring does not allow crossovers does not preclude its use for DFD routing.

Placement Algorithm Selected

The proper placement of icons is critical in the production of easy-to-understand data flow diagrams. A poor placement strategy can position icons such that a routing algorithm, no matter how sophisticated, cannot position the data flow arcs subject to the guidelines set out previously. Although two placement algorithms (pair linking and cluster development) were considered for inclusion in upgrading the system MONDRIAN, only the pair-linking algorithm is outlined here since it performed best in the preliminary evaluation. A detailed description of both algorithms can be found in Protsko, (1983).

An adaptation of the design automation pair-linking algorithm to the DFD layout problem begins by choosing, as a nucleus, the pair of nodes which have the largest sum of common weighted signal sets. Unplaced nodes are selected on the basis of high connectivity to a placed node and positioned as close as possible to that placed node. The steps of the following general algorithm are elaborated upon in this subsection.

1. Determine connectivity of each pair of nodes.
2. Select initial pair of nodes for placement.
3. Position initial pair of nodes.
4. Repeat steps 5 and 6 until all nodes are placed.
5. Select next node for placement.
6. Place next node.

The P-matrix is an n by n array, where n is the number of nodes to be placed. I is initialized by traversing the adjacency list. For each listnode of the adjacency list, the value in the xy position is incremented by p. The x and y coordinates represent the numbers assigned to each headnode connected by the listnode. The value, p, identifies the types of two headnodes connected by the listnode. If they are both processes, p equals three. If they are a process and a data store, p equals two and if they are a process and an interface, p equals one. These values are used to derive a priority for the placement of process nodes and to assist in minimizing the amount of replication of data stores. A larger p value produces a larger value in the P-matrix and the largest value in the P-matrix is chosen for placement. This is a deviation from the design automation pair-linking method which is designed to compensate for the difference in the nature of the signal sets between design automation and DFDs. Specifically, signal sets in design automation consist of two or more elements, whereas those in DFDs have exactly two elements. Without this modification there would be little, if any, difference in the P-values as most node pairs are connected by only one arc. Very rarely are node pairs connected by more than two arcs.

The initial pair of nodes is determined by finding the x and y coordinates of the P-matrix containing the maximum value. Ties are broken by finding the pair with the highest total weighted connectivity. The weighted connectivity is calculated by summing a weight factor for each incident node. The weighting factor depends on the type of the incident node. In the current implementation, process nodes are assigned a weight of 3, data stores 2 and interfaces 1. If this calculation does not resolve the tie, the first pair is selected. These weights were chosen to promote the placement of process nodes before data stores or interfaces.

All positioning of nodes in this algorithm uses the workspace grid described in Section 3. The initial pair of nodes are placed near the center of this grid. These positions were chosen such that placement can proceed in all directions.

To aid in the selection of unplaced modules, a vector of boolean values is maintained. These values are used to record whether the corresponding node had been placed. To select the unplaced node with the highest connectivity to a placed node, each row of the P-matrix, which has a truth value in the corresponding position of the placed-vector, is examined. The entries for which the corresponding position of the placed-vector has a value of false are compared to find the one with the largest value. The x position of the maximum value gives the number of the unplaced node, while the y position gives the number of the connecting placed node. Ties are broken in a similar fashion to tie-breaking in the selection of the initial pair. The only difference is that if a tie occurs in the calculation of the weighted connectivity of the pairs, the pair containing the unplaced node which is to be connected to the greatest number of currently-placed nodes, is selected.

The positioning strategy algorithm is somewhat modified from that used in the design automation pair-linking algorithm. This modification was required to accommodate the special use of the workspace grid and the two element signal sets of DFDs.

If the node being placed is only to be connected to the placed node of the pair, the current alternative on the alternative list for the placed node is chosen. The current alternative is found by using the index value and type indicator of the workspace position occupied by the placed node. The type indicator specifies which of the four alternative-position arrays to use and the index value specifies which of the sixteen positions in the array to use. The relative x and y coordinates found in the alternative-position array are added to the x and y positions of the placed node to produce the workspace coordinates for the unplaced node. The index value is then incremented for the next placement of a node. If the calculated position has been previously assigned a node, the incremented index value is used to determine the next alternative position.

If the node being placed is connected to more than one node but only one of those nodes has been placed, a determination of the area in which the unplaced nodes should be located is made and the alternative closest to that area is chosen. The determination is made by looking at each of the connecting but unplaced nodes. If the connecting unplaced node is within three links (arcs) of a placed node, the coordinates of that placed node are used in the calculation of an average position. The alternative position for the placed node of the pair which is closest to this average position is chosen for the unplaced node.

When more than one of the connecting nodes are currently placed, the average position of these nodes is determined. If this position is reachable from the placed node of the pair, it is selected for the unplaced node. If the averaged position is not reachable from the placed node of the pair, the alternative of the placed node closest to the average position is selected as the position for the unplaced node.

Routing Algorithm Selected

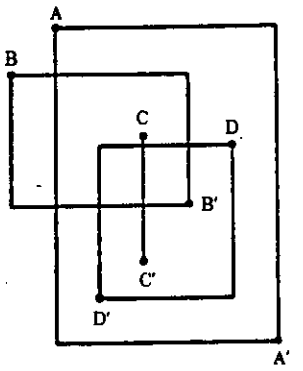
Two approaches to DFD routing were selected; namely an adaptation of one layer Manhattan wiring and 2-CNF-satisfiability; another based on predetermined endpoints for the DFD arcs. Since the later method outperformed the former in preliminary evaluations, it is the only method outlines here.

PREDETERMINED END-POINT ALGORITHM

MONDRIAN's original routing algorithm routes each arc in the order determined by an ordering routine and considers only how the path affects those arcs which have already been placed.

The description of this algorithm begins with an overview of the ordering routine used. This is followed by a discussion of the method for finding all possible paths for each arc and choosing the most appropriate one. Next, the assignment of the paths to the subgrid is outlined. The presentation discussion is concluded with a description of replication and relocation.

The ordering routine is used to produce the sequence of arcs for which paths must be determined. This routine is based on the ordering algorithm described by Akers (1972). An order table is created to record the sequence. It is represented by an array with a row for each arc to be routed. Each row has an entry for the address of the listnode, the address of the headnode, and a point value. To create the sequence, the adjacency list is traversed. For each listnode, a rectangle is created using the coordinates of the two connecting headnodes as opposite corner points. Every other listnode is examined. If one of the connecting headnodes falls within the rectangle, the point value in the order table position representing the listnode for which the rectangle was created, is incremented by one. Figure 7 shows a set of points, their squares, and the resulting point values.



POINT VALUE	ORDERING
A:5	4TH
B:1	2ND
C:0	1ST
D:2	3RD

Figure 7

Rectangles and Point Values for Ordering Algorithms

After each listnode has been examined, the order table is sorted in ascending order of the point value using a quick sort.

Paths are found for each arc by examining each row of the sorted order list. Finding all valid paths for a particu-

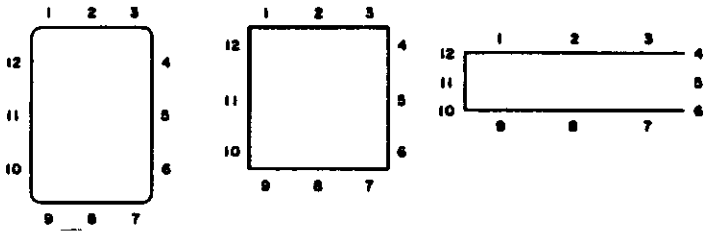


Figure 8

Possible Starting or Ending Positions for Arcs

lar arc begins by checking for the presence of another node along the main grid squares in which the path may lie. If a node is present, all paths in that direction are considered blocked. If both directions are blocked, replication or relocation of a node is considered.

Once the possible directions of paths are determined, the subgrid coordinates for the starting, ending, and corner points of the individual paths are calculated. A pathnode for each path is allocated and linked into a path list attached to the listnode representing the arc. Figure 8 shows the possible starting or ending points for each icon type. In this level of path assignment, only six paths between icons are possible. These paths are shown in Figure 9.

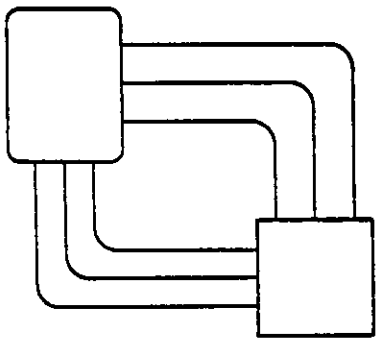


Figure 9

Possible Paths Between Nodes

After all possible paths at this level have been assigned to pathnodes, each path is checked for validity. To be valid, the path must not cross through an icon or share a corner position with another arc. To check for validity, the contents of each square of the subgrid in which the path would pass is examined. If the square is empty, the examination moves to the next square on the path. The only acceptable content of a square is the indicator of another arc if the coordinates are not those of the corner of the arc. Any other content indicated the path is invalid.

The second level of assigning paths is only used when one of the six paths from the first level are suitable and when the first level of path assignment is completed for all other arcs. In this level, the other possible combinations (shown in Figure 10), are used to determine the remaining paths between the nodes rather than allocate pathnodes for each alternative, the subgrid squares around each node are examined. If the square is empty, a path could begin or end there. If such a square exists for both nodes and if they are on appropriate sides of the node, a pathnode is allocated and the coordinates assigned. When all possible paths have been assigned, each is validated as in the first level.

After all possible paths at the particular level have been allocated for a node, they are rated and the path with the highest rating chosen. Paths are rated in terms of the following criteria:

1. Position on the icon and corner type,
2. Direction of path,
3. Number of crossovers produced, and
4. Length of path.

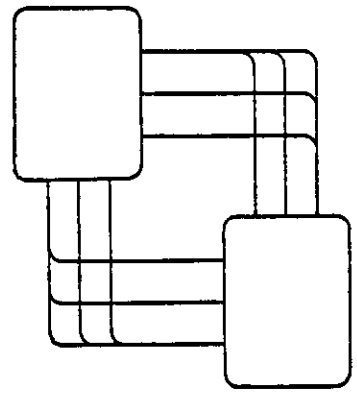


Figure 10

Second Level Path Possibilities

The first two criteria are used to produce balanced routing in the diagram. The third is used to minimize crossovers and the fourth is used to minimize the length of paths.

As each path is chosen, it is assigned to the subgrid. Using the starting, ending, and corner coordinates, each subgrid square on the path is examined. If the square is empty, an indicator of the direction (horizontal, vertical or corner) is placed in the square. If the square contains the directional indicator of another arc (other than a corner), the contents are changed to show a crossover. If the square's contents indicate a corner of an arc or a node, an error has occured. All indications added or changed as a result of this path are negated and a new path is selected and the process repeated. If all the paths from the first level (see Figure 8) are eliminated, the path for the remaining arcs are assigned. When first-level paths for all arcs have been assigned and an arc (or arcs) remain unrouted, the second level of path assignment is invoked. The second level of path assignment uses the additional possible paths shown in Figure 9. If this fails to produce the required paths, the arc is considered unroutable.

Replication is used when a path between two nodes (one being an interface or data store) does not exist, is too long, or creates too many crossovers. The position that the replicated node will occupy is calculated in a similar fashion to the regular placement procedure but with the position of the process node of the offending arc having a greater weight in the calculation. Once a new position is chosen, placement and routing proceed in the usual manner.

TEXT POSITIONING

Text is positioned on arcs in the order determined by the ordering routine used in the predetermined-endpoint routing algorithm. Text is positioned in a block of subgrid squares. The size and location of a block varies with the type of placement. If the text is to be placed along a horizontal line, the block consists of three consecutive squares along the arc as shown in Figure 11a. These three squares may be right on the arc or immediately above or below the arc. When the block is above or below the arc, it is desirable to leave a buffer of three squares above (or below) the text block so that two arc names do not overlap and the name can be uniquely identified with its arc. If the text is to be positioned along a vertical line, three options (Figure 11b) are available. The first two options involve a two by two block of squares on either side of the arc. When the diagram is drawn, the text is right or left justified so that it lines up against the arc. The third option involves a two by two block of squares centered on the arc. The arc will be broken at this point and the text centered between the arc segments.

The selection of the position for the block is restricted to those positions on the arc which would not result in the text being placed in the beginning or ending square of the arc or extending over a corner of the arc.

To begin the selection of the position for the text the center point of a straight arc or the center point of the longest segment of an arc with a turn, is chosen. If this position cannot be used, a position slightly off-center for straight arcs or the center point of the shorter segment of an arc with a turn is chosen. If this position also fails, an exhaustive search of all positions on the arc is undertaken. The first two positions are selected to try to produce a balanced positioning of text on the arcs. If no position along the arc can be used, the arc will be unlabelled when the diagram is drawn and a message indicating this is given to the user.

The routing algorithm described in this section and the Raghaven-2-CNF-based algorithm were paired with the placement algorithm and the cluster-development

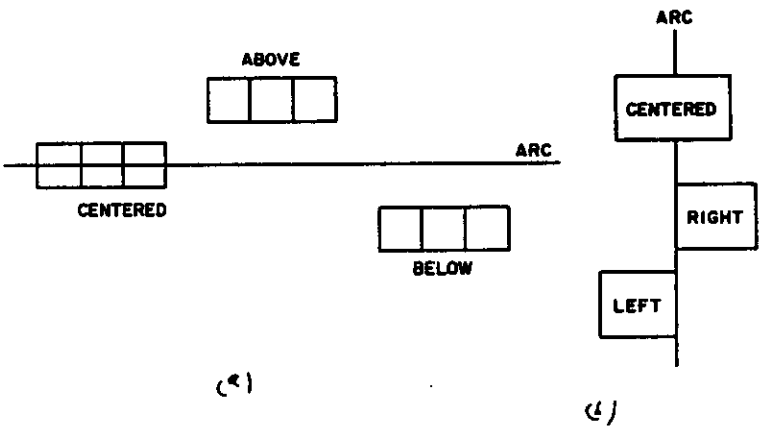


Figure 11

Possible Positions of Text Blocks a) Horizontal b) Vertical

algorithm in order to be evaluated. Based on a preliminary evaluation (see Protsko, 1983), the algorithm pair of pair-linking placement and predetermined-endpoint routing are recommended as the pair to be used for production of DFDs.

Conclusions and Proposed System Enhancements

In this paper, the problem of producing computer-drawn data flow diagrams was introduced. A general architecture for MONDRIAN, a prototype system developed to layout DFDs automatically, was presented. Various placement and routing algorithms that addressed the layout problem were described. The pair-linking placement and predetermined-endpoint routing algorithms were thought to be the most promising strategies and are currently being adapted to an improved, refined prototype. To exploit more fully the benefits of automated production of DFDs, future enhancements should be completed in two main areas: improvements to MONDRIAN and extending MONDRIAN.

A valuable extenstion to MONDRIAN would be an interactive component which would allow the analyst to modify the generated diagram. This component would allow the user to route the arcs which were not routed by MONDRIAN. When modifications to the diagram are the addition or deletion of objects (processes, data stores, or interfaces), it would be desirable to concurrently update the SPSA database. When making additions to the diagram, the analyst would not have to be overly concerned with the placement and routing involved, as the extended system would produce a new layout when the additions were completed.

A major improvement to the prototype would be the capability of producing DFDs in tiers. Many of the analysts evaluating the DFDs suggested this for the larger systems. To accomplish this, two extensions to MONDRIAN would be required. First, MONDRIAN should support the production of a series of DFDs showing the hierarchical structure of the system description. That is, it should produce a DFD for each successive refinement of the system description. The second extension would have MONDRIAN support the logical grouping of elements in the system being described. This would allow the user to specify (perhaps by use of the SPSL concept *keyword*) which objects are logically related and which should be placed in one area of the diagram or in a separate diagram.

A major improvement in the implementation of the MONDRIAN prototype would involve changing the adjacency list structure to a database structure which is retained after the generation of the diagram. This would greatly facilitate the extensions that were just proposed for MONDRIAN.

Finally, some further testing and evaluating of the quality of the layouts produced by the improved prototype must be undertaken to ensure that the generated diagrams are comparable to a well-planned manually produced layout.

REFERENCES

Akers, S. B. "Routing," *Design Automation in Digital System: Theory and Techniques, Volume 2,* Chap. 6, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

Breuer, M. A. "Recent Developments in the Automated Design and Analysis of Digital Systems," *Proceeding of the IEEE,* Volume 60, Number 1, January 1972, pp. 12-27.

Channen, E., Sorenson, P. G. and Tremblay, J. P. "Structured Systems Analysis Using SPSL/SPSA," Department of Computational Science Technical Report 877, University of Saskatchewan, Saskatchewan, Canada, 1984.

DeMarco, T. *Structured Analysis and System Specification,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

Even, S., Itai, A., and Shamir, A. "On the Complexity of Timetable and Multi-Commodity Flow Problems," *Society for Industrial and Applied Mathematics-Journal of Computing,* Volume 5, Number 4, December 1976, pp. 691-703.

Fisk, C. J., Caskey, D. L. and West, L. L. "ACCEL: Automated Circuit Card Etching Layout," *Proceedings of the IEEE,* Volume 55, Number 11, 1967, pp. 1971-1982.

Freisen, A. W., Soreson, P. G. and Tremblay, J. P. *SPSL/SPSA Primer Version 1.1,* Department of Computational Science Technical Report 82-5, University of Saskatchewan, Saskatchewan, Canada, 1982.

Gane, C. and Sarson, T. *Structured Systems Analysis: Tools and Techniques,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

Garey, M.R., Johnson, D. S. and Stockmeyer, L. "Some Simplified NP-Complete Problems," *Proceedings of the 6th Annual ACM Symposium on the Theory of Computing,* Seattle, Washington, April 1974, pp. 47-63.

Gilmore, P. C. "Optimal and Suboptimal Algorithms for Quadratic Assignment Problems," *Journal of the Society for Industrial and Applied Mathematics,* Volume 10, Number 2, June 1966, pp. 305-313.

Hanan, M. and Kurtzberg, J. M. "Placement Techniques." *Design Automation of Digital Systems, Theory and Techniques, Volume 1,* Chap. 5, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

Hwang, F. W. "An O(n log n) Algorithm for Suboptimal

Rectilinear Steiner Trees," *IEEE Transactions on Circuits and Systems*, Volume CAS-26, Number 1, January 1979, pp. 75-76.

Kurtzberg, J. M. "Algorithms for Backplane Formation," in *Microelectronics in Large Systems*, Spartan Books, 1972.

Lee, C. Y. "Algorithm for Path Connections and Its Applications," *IRE Transactions of Electronic Computers*, Volume EC-10, Number 3, September 1961, pp. 346-365.

Loberman, H. and Weinunberger, A. "Formal Procedures for Connecting Terminals With a Minimal Wire Length." *J. ACM*, Volume 4, October 1957, pp. 346-365.

Nunamaker, J. F. and Konsynski, B. "Formal and Automatic Techniques of System Analysis and Design," in *Systems Analysis and Design: A Foundation for the 1980's*, W. Gotterman *et al.*, (ed.), North Holland Publishing Company, Amsterdam, The Netherlands, 1981, pp. 291-320.

Perkins, D.R. "The Design of a User-Oriented Interface for a Minicomputer Implementation of a Computer-Analysis and Documentation Package," M.Sc. Thesis, Department of Computational Science, University of Saskatchewan, Saskatchewan, Canada, June 1979.

Pomentale, T. "An Algorithm for Minimizing Backboard Wiring Functions," *Communications of the ACM*, Volume 8, Number 11, November 1965, pp. 699-703.

Protsko, L.B. "Placement and Routing Algorithms for the Automatic Generations of Data Flow Diagrams," M.Sc. Thesis, Department of Computational Science, University of Saskatchewan, Saskatchewan, Canada, 1983.

Quinn, N.R. Jr. and Breuer, M.A. "A Forced Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Transaction on Circuits and Systems*, Volume CAS-26, Number 6, June 1979, pp. 377-388.

Raghaven, R., Cahoon, J. and Sahni, S. "Manhattan and Rectilinear Wiring," Technical Report 81-5, University of Minnesota, Minneapolis, Minnesota, March 1981.

Steinberg, L. "The Backboard Wiring Problem: A Placement Algorithm," *SIAM Reu*, Volume 3, Number 1, January 1961, pp. 37-50.

Teichroew, D. and Hershey, E. "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Systems," *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, 1977, pp. 48-58.

Wig, E. "PSL/PSA Reporting and Prototype Output Generation Facilities in a Mini-Computer Environment," M.Sc. Thesis, Department of Computational Science, University of Saskatchewan, Saskatchewan, Canada, June 1979a.

Wig, E. "Computer-Aided Systems Analysis Reporting," Department of Computational Science Technical Report, University of Saskatchewan, Saskatchewan, Canada, September 1979b.

Yourdon, E. and Constantine, L. *Structured Design*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.