AUTO-DFD: An Intelligent Data Flow Processor

K. P. TAN,* T. S. CHUA AND P. T. LEE
Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore 0511

This paper describes the design and implementation of an intelligent data flow processor, AUTO-DFD. AUTO-DFD has the intelligence to find the optimal Data Flow path between two entities of a data flow diagram. It has a high degree of automation in dynamically moving or deleting objects and their related components. It allows multi-level data flow diagrams to be concurrently created in multi-windows and merged with their parent diagrams. It performs checking on the integrity of all entities of the data flow diagram. It also checks on the balance of input and output flows between a Process and its child diagram.

Received July 1988

1. INTRODUCTION

Systems Analysis is the earliest phase in the life cycle of a system. Therefore, its significance on the later stages of systems development (design, coding, testing, implementation and maintenance) cannot be under-emphasised. Many systems do not serve their intended purposes because of poor analysis although most analysts have spent much time and money in analysing systems and their problems. With the increasing emphasis on productivity, the analysts' work can be facilitated by the use of computer-aided analysis systems such as DFD[1], SADT[2], PDL[3] and PSL/PDA.[4]

The use of Data Flow Diagrams as a tool for Structured Analysis has been discussed.[1] Demarco[1] defines a data flow diagram as a network representation of a system. Today, there are many commercially available computer-aided analysis tools that support data flow methodologies. They include POSE[5] AutoAsyst, Excelerator, Structured Architect, SA Tools, Software Engineering Workbench and USE. Some of these tools do not check consistency across diagrams and do not detect duplicates. They do not support automatic diagram generation. Moreover, they do not allow an object to be replaced by its child diagram. As the information processing environment expands and problems become more complicated, a more powerful and intelligent computer-aided tool is required.

2. REQUIREMENTS OF AN INTELLIGENT SOFTWARE TOOL

To develop high quality software, analysts need an automated tool that keeps track of each element in the information system for fast retrieval. The tool should enforce consistent definition of each element in the diagrams and detect duplicates. This will maintain the integrity and consistency of the data dictionary. It should have the intelligence to generate optimal routes for data flows so that the diagrams are visually documented for the analysts to understand easily. It must allow dynamic modification of diagrams by moving or deleting objects and their related components with minimum effort from the analysts. The tool should encourage partitioning by allowing the child diagrams to be concurrently edited with their parent diagrams in a user-friendly manner. It must enforce consistency across diagrams to maintain the integrity of the information system. The tool should also support systematic replacement of any element.

AUTO-DFD, an integrated analysis and design tool, has been developed to meet all the above requirements. It is implemented on the Apollo Domain 580 graphic workstation at the National University of Singapore. AUTO-DFD is coded in the C programming language and is supported by the Graphics Metafile Resource package which uses the PHIGS standard.[6] In summary, AUTO-DFD has the following features:

(*a*) integrates DFD and data dictionary
(*b*) detects duplicates
(*c*) enforces diagramming rules
(*d*) performs object search
(*e*) automatically generates data flows
(*f*) dynamically moves or deletes objects and related components
(*g*) supports multi-windowing to edit diagrams of different levels concurrently
(*h*) checks consistency across diagrams
(*i*) systematically replaces objects with their child diagrams
(*j*) compresses diagrams
(*k*) provides on-line help

This paper presents the architecture of AUTO-DFD. It describes the functions performed by each component. It also elaborates on the heuristic routing algorithm devised for finding optimal Data Flow route between two entities of a data flow diagram.

3. DESIGN OF AUTO-DFD

AUTO-DFD is designed to be object-oriented.[7] This is because the use of object-oriented paradigm in the field of software development will increase the user's acceptance in describing the structuring, functionality and evolution of software in a uniform manner.

Since AUTO-DFD is designed for ease-of-use and high processing power, it aims to provide a completely visual environment for analysts to model the information system by manipulating icons on screen. This ensures that even a naive user will be able to make effective use of the full potential of the system.

The architecture of AUTO-DFD is illustrated in Fig. 1. The Data Dictionary contains definitions of every entity in the data flow diagrams. It declares the data fields and types of each entity. It also defines the graphic

* To whom correspondence should be addressed.

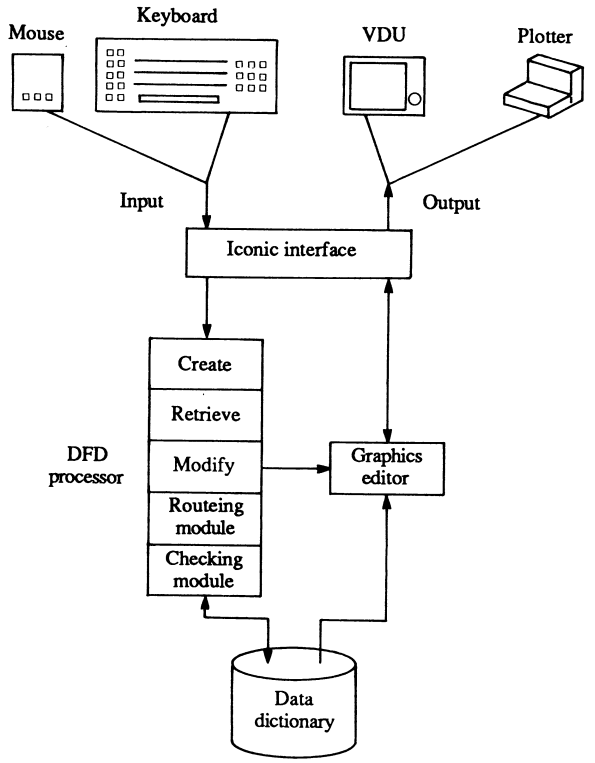

Figure 1. The architecture of AUTO-DFD.

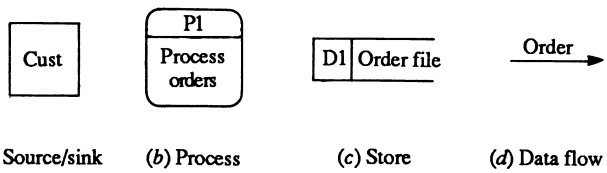

Figure 2. The DFD objects.

display parameters (colour, size, shape, position) of an entity.

The Graphics Editor creates, modifies, moves and deletes objects on the screen. It retrieves the object definitions from the Data Dictionary. It basically manipulates four types of objects – Process, Store, Source/Sink and Data flow, they are shown in Fig. 2. The use of these entities are described.[1] The functions performed by the graphics editor are supported by a powerful graphics package GMR (Graphics Metafile Resource) which is installed on the Apollo Domain 580.

The iconic interface serves as an interface between the user and AUTO-DFD. The DFD processor performs editing and checking on data flow diagrams. The structure of the iconic interface and the functions of the DFD processor will be described in the following sections.

4. THE ICONIC INTERFACE

To reduce the complexity of the information tool, an iconic interface is developed. Iconic communication uses images to convey ideas of information in a non-verbal manner.[8] The iconic interface makes pictorial communication between the user and AUTO-DFD possible by allowing the user to select from menus and pick objects with a picking device (in this case, a mouse). This serves to provide a 'user-friendly' environment to analyse the problem domain and generate the required outputs on the graphic display. The keyboard is used for keying in other textual data. Interfaces to output devices such as printers and graphic plotters will be included in future enhancements.

The screen layout displayed during a data flow diagram construction is shown in Fig. 3.

The screen is partitioned into 8 windows. The function of each window is described below:

Top menu	The Top Menu window shows the main menu of the DFD.
Side menu	The functions that can be used to manipulate objects are listed on the right of the screen.
Display	The data flow diagram is displayed in the centre of the screen.
Multi	These are smaller multi-windows opened to display and edit the child diagram of a Process or to zoom part of the data flow diagram.
Scroll	The scroll window is at the bottom right of the screen. User can use the arrow keys to scroll the display window's content.
Directory	The directory window on the left lists the existing data flow diagrams for user to select for modification or viewing purposes. The arrows at the bottom left are used to scroll the directory.
Input	This window allows users to enter textual data or command.
Message	This prompts user for action and gives any error or completion message.

5. THE DFD PROCESSOR

The DFD processor allows user to create a new data flow diagram, and retrieve or modify an existing diagram. It divides the screen into rows and columns of invisible grid for data flows, as shown in Fig. 4. It adopts the directional constraint[9] to draw horizontal and vertical lines. In addition, the screen is partitioned into subareas called slots (shown by the solid lines) for Process, Store and Source/Sink objects. The DFD processor uses the modular constraint[9] to centralise an object in a slot.

A user edits a new diagram by selecting the object type (Process, Store and Source/Sink) from the side menu and then indicating the corresponding slot for the object to be drawn. Data flows between two objects can be edited by user or automatically generated by the data flow routing module.

The routing module will generate a sequence of routes by studying the location of the source and destination objects. It then calls the checking module to determine whether a route is blocked by other objects. The routing module will terminate when an optimal path is found. This is done by assigning heurisitc values to each feasible path generated. The criteria for selecting a generated data flow is based on minimum turning points, minimum crossing with other data flows and shortest distance.

The functions supported by the DFD processor and the processings involved in these functions are described as follows.

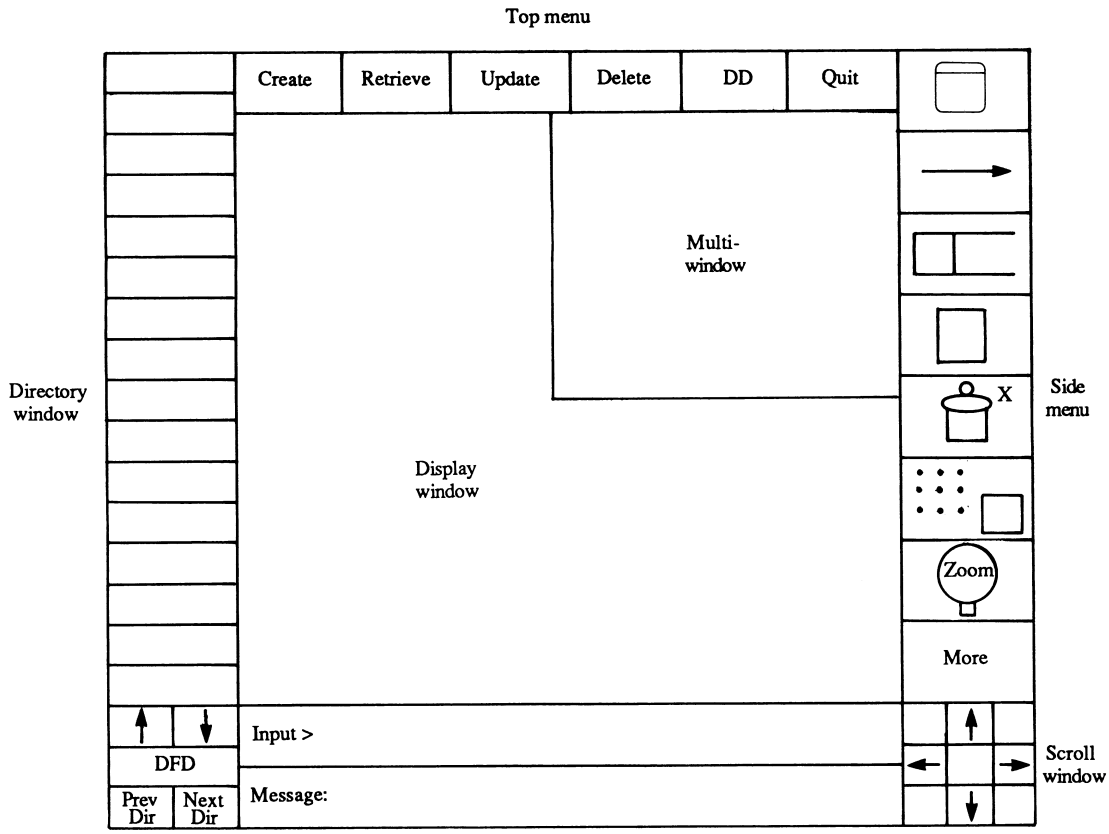

Figure 3. The iconic interface.

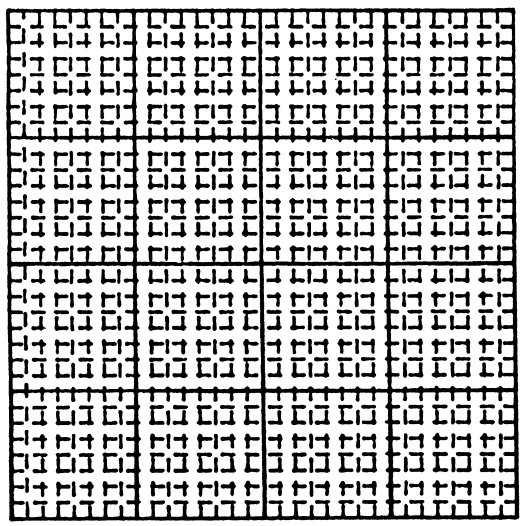

Figure 4. Internal representation of the Display window.

(*a*) Inserting an object

The checking module will first check that there is no overlapping of the new object with the existing objects and that the name of the new object is unique. For Process and Store objects, AUTO-DFD automatically generates identification numbers in sequential order. Processes will have identification of P1, P2, P3 etc and Stores will have identification of D1, D2, D3 etc. The graphics editor will then position the object in the centre of the selected slot.

In the case of editing a data flow manually, at least two points must be indicated. The checking module will also check for a valid source and destination. In the automatic mode, it is sufficient to just pick the source and destination object for the routing module to generate an optimal path. The checking module will check if there is any looping regardless of the mode used. Before saving a data flow diagram, the checking module will check that each Process and Store has valid input and output data flows.

(*b*) Deleting an object

The object to be deleted is first picked by the user and then removed from the screen. Furthermore, if the picked object type is a Process, Store or Source/Sink, all of its related data flows which have been maintained by the DFD processor are automatically deleted. The DFD processor will also update the identification numbers of affected Processes and Stores. This change will also be propagated to the child diagram of any Process affected by decrementing its prefix. Fig. 5 shows that Process P1 before and after deletion.

(*c*) Moving an object

If a data flow object is moved, the user will either re-specify the path or the routing module will automatically generate a new path depending on the mode that was specified. In the case of a Process, Store or Source/Sink object, the checking module will again check that it does not overlap with other objects on the screen. The graphics editor will then move the object to its new location. In addition, all the data flows of the moved

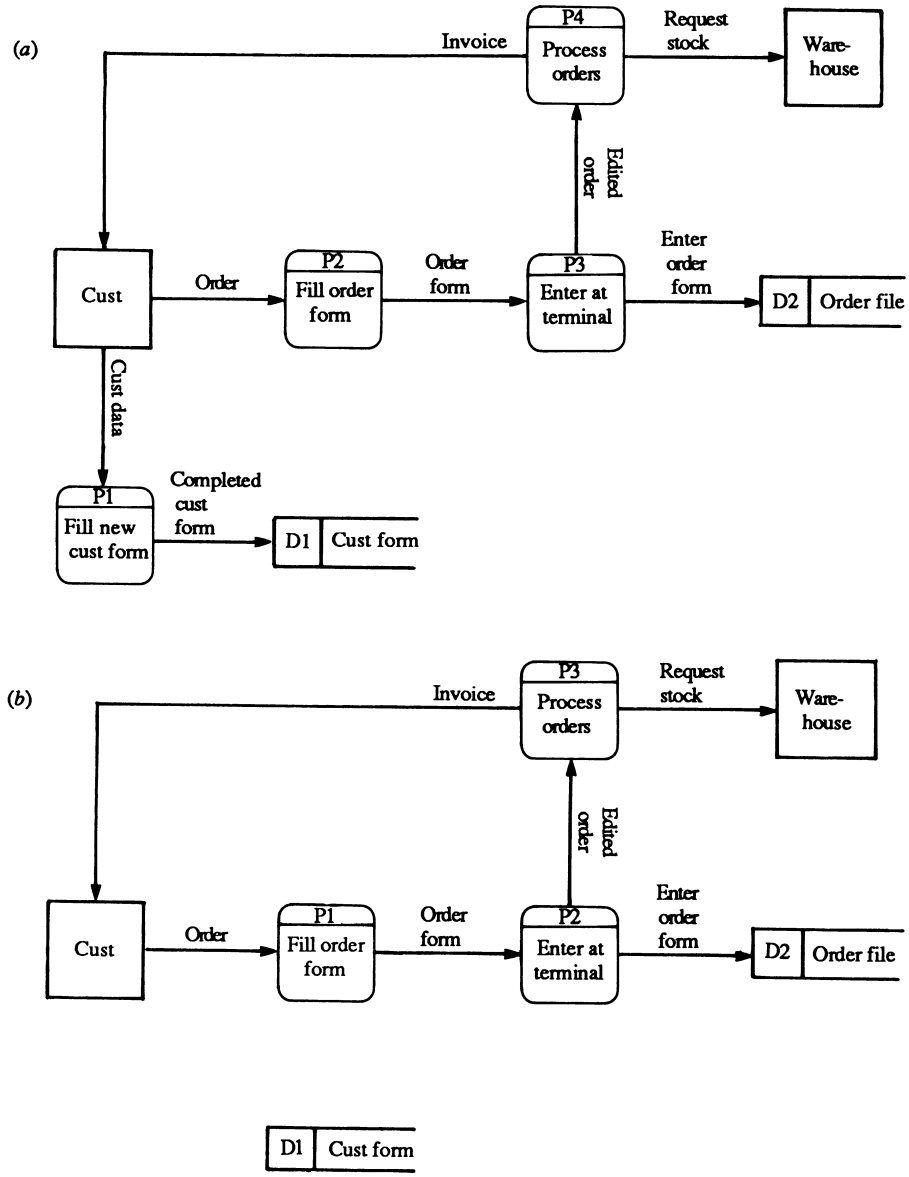

Figure 5. (*a*) An order-processing system; (*b*) after deleting process P1.

object will be automatically re-routed by the routing module. Figure 6 shows the Source CUST of Fig. 5*b* (with the previous D1 deleted) being moved to its new location.

(*d*) Creating a child diagram for a Process object

Since partitioning is a characteristic of data flow diagrams, AUTO-DFD allows the detail operations of a Process to be edited at a lower level child diagram. This features a top-down approach to analysis. As a guideline, Miller[10] has suggested an average of seven objects at any level for the human mind to process the information. The concept of boundary clashes in Jackson[11] is also applicable in determining the bottom level.

To simplify the creation of the child diagram, the parent Process is present on the screen for the user to pick its input and output data flows. Therefore, the child diagram is edited in a subwindow on the top right of the screen. Adopting the levelling convention in DeMarco[1], AUTO-DFD automatically generates a prefix of its parent Process identification number for each of its child Processes created in the subwindow. For example, the child Processes of parent Process P3 will have identification of P3.1, P3.2, P3.3 etc. assigned to them. Another characteristic of AUTO-DFD is that it allows multi-level child diagrams to be created and edited concurrently with their parent diagrams by opening multi windows.

The DFD processor checks that inputs and outputs are balanced between the Process of a parent diagram and its child diagram; that is, data flows into and out of the parent process are equivalent to those flowing into and out of the child diagram. In addition, it maintains a reference pointer between a process data flow of the parent diagram and the corresponding data flow of the child diagram. This ensures that the parent Process can be systematically replaced by its child diagram later. Figure 7 shows the creation of a child diagram for the Process P3 of Fig. 6.

(*e*) Replacing a Process object with its child diagram

AUTO-DFD allows the user to expand a data flow diagram by replacing any parent Process with its

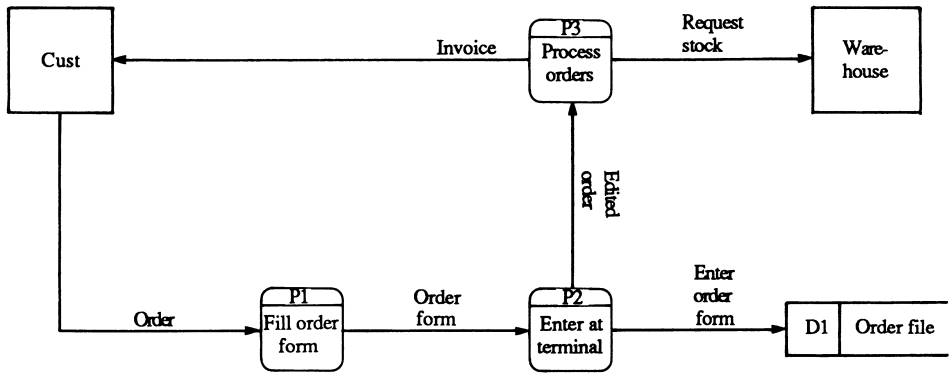

Figure 6. Moving Source CUST.

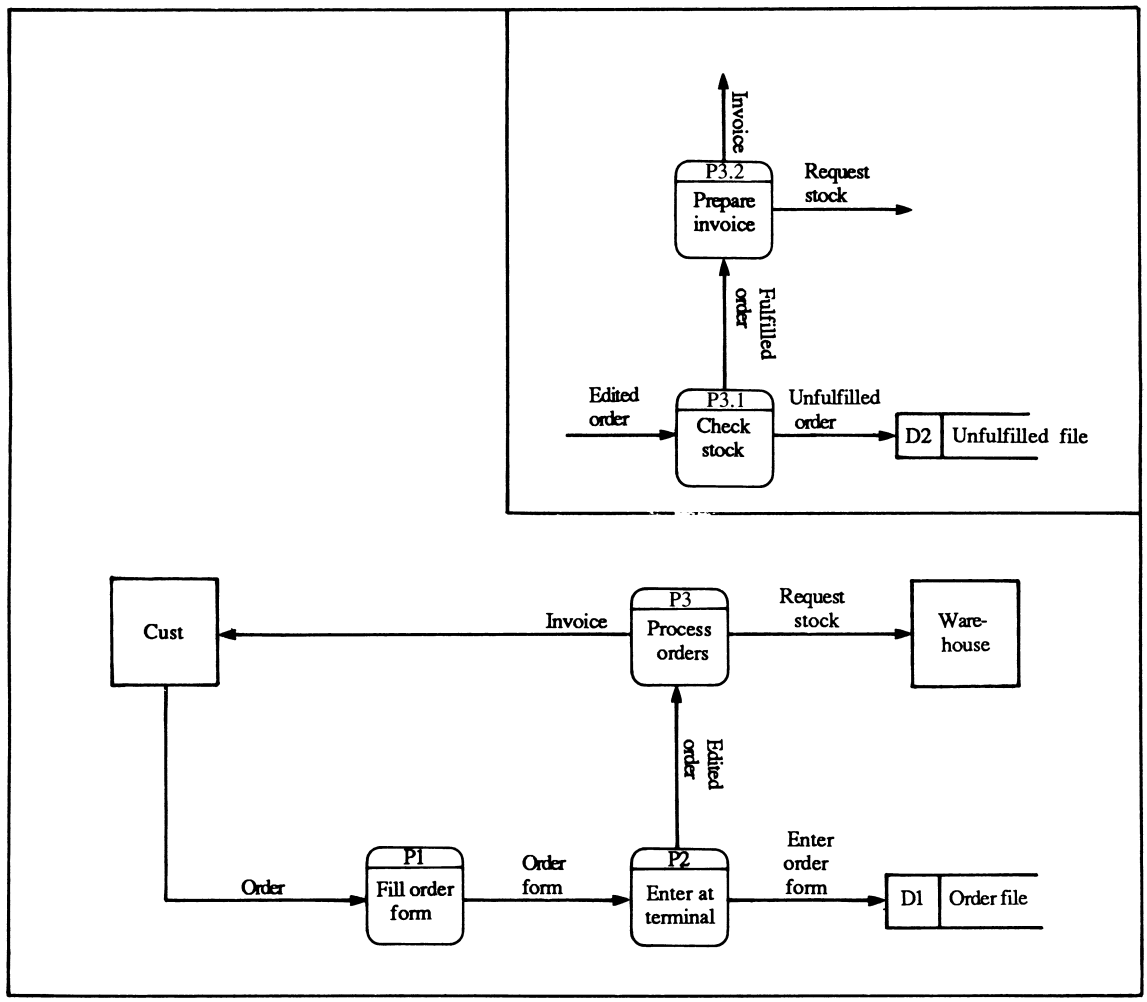

Figure 7. Creating a child diagram for Process P3.

corresponding child diagram. If a child Process is not primitive, the user can further request that the child Process be replaced by the next lower level diagram.

To replace a parent Process, the DFD Processor will first determine the space in terms of number of slots required by the child diagram. It then moves all objects that lie on the top and right of the parent Process to give sufficient space for the child diagram. The Process, Store and Source/Sink objects of the child diagram are now inserted into the space allocated. The DFD processor finally proceeds to find new routes for the data flows of all the moved parent objects and newly inserted child objects. Figure 8 shows the Process P3 of Fig. 7 being replaced by its child diagram.

6. THE ROUTING ALGORITHM

The routing problem is an NP-complete problem, for which non-deterministic (an arbitrary number of paths can be followed at once) polynomial-time algorithms are known, but for which all known deterministic algorithms

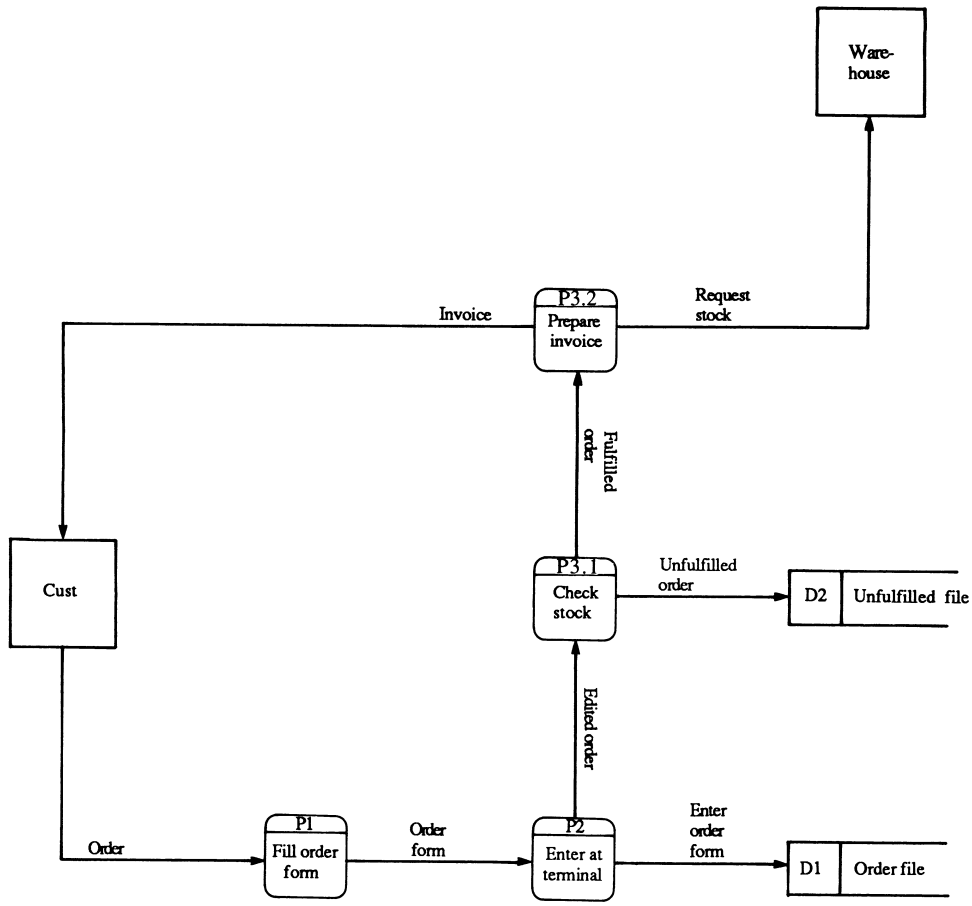

Figure 8. Replacing Process P3 by its child diagram.

are exponential.[12] The time required to find the optimal route increases exponentially with the routing area. There are many well-known routing algorithms; they include the Lee Algorithm[13] and the Hightower Algorithm.[14] These algorithms are designed to find the shortest path with no crossings between any two points in a gridded routing region containing any number of obstacles. They are used in complex area routing problems for VLSI design. When the routing area is small, both algorithms can find the optimal path in an acceptable time. In a real time system, a more powerful algorithm is needed to achieve an acceptable completion rate for larger routing areas.

With good partitioning, there are relatively less objects and connections in a data flow diagram. Therefore, it is desirable to have an algorithm that first considers 'obvious' and simple solutions by studying the location of the source and destination objects and the blockages. The algorithm should not only consider paths with minimum crossings, but also short paths with few zigzagging. In a data flow diagram, paths with less turnings maybe visually more desirable than long paths with more turnings and less crossings.

A routing algorithm which relies on heuristics has been devised for AUTO-DFD to find a visually acceptable data flow path between two objects. To find a path between any two objects, the algorithm considers routes with not more than three turning points. In each case, it will give priority to routes with minimum crossings and then shortest distance. There are mainly 3 cases considered by the Routing Algorithm. Figure 9*a*, *b* shows the trivial cases of two objects in the same row and same column respectively. Figure 9*c* shows the possible routes between two objects in different row and column. The other combinations are symmetries of the third case.

The Routing Algorithm is given below:

```
ROUTING ALGORITHM
  best_route = NIL;
  turning_point = 0;
  max_turning_points = 3;
  max_turning_reached = FALSE;
  optimal_found = FALSE;
  while ( NOT(optimal_found) AND
   NOT(max_turning_reached) )  {
    obstacle = find_obstacle (source_object,
     destination_object, turning_point);
    if NOT(obstacle)
      then call GENERATE_DATAFLOW_PATH
       (best_route, optimal_found);
    turning_point = turning_point + 1;
    if (turning_point > max_turning_points)
      then max_turning_reached = TRUE;
  }
Procedure GENERATE_DATAFLOW_PATH
 (best_route, optimal_found)
  new_route = NIL;
  destination_reached = FALSE;
  path_rejected = FALSE;
  exhausted = FALSE;
  while NOT(destination_reached) AND
   NOT(path_rejected)
```

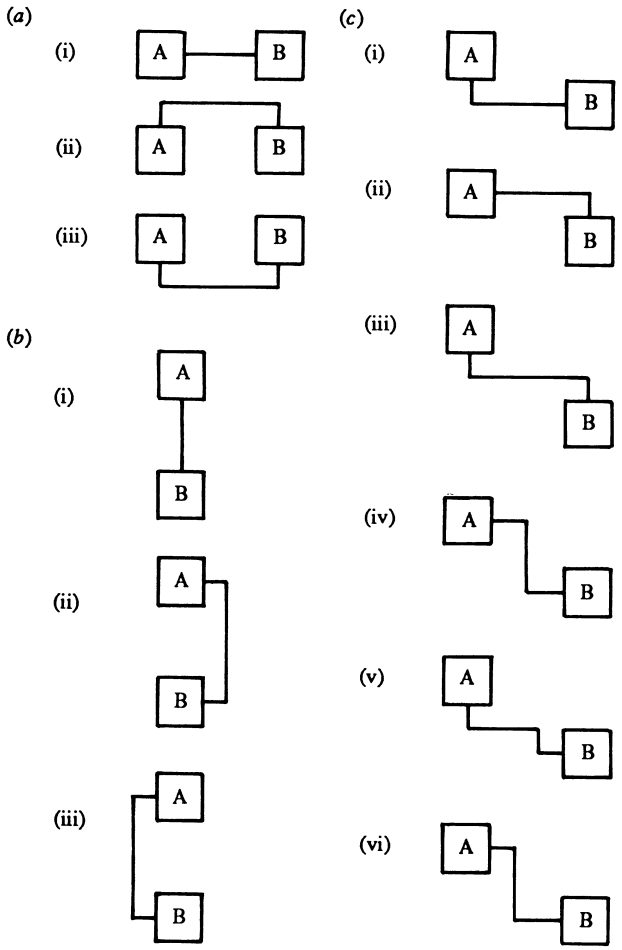

Figure 9. Possible routes generated. (*a*) Objects in same row; (*b*) objects in same column; (*c*) objects in different row and column.

```
AND NOT(exhausted) {
  /* generate a subpath */
  call GENERATE_SUBPATH (sub_path,
    exhausted, destination_reached);
  if NOT(exhausted) {
    /* check for invalid overlapping */
    path_rejected = CHECKING_ROUTINE
      (sub_path);
    if NOT(path_rejected) {
      /* concatenate partially generated route
        with subpath*/
      new_route = concat (new_route, sub_path);
      if COMPARE_HEURISTICALLY
        (new_route, best_route) < = 0
      path_rejected = TRUE;
    }
  }
}
if COMPARE_HEURISTICALLY (new_route,
  best_route) > 0
  best_route = new_route;
if (best_route.dataflow_crossing = = 0)
  optimal_found = TRUE;
else
  call GENERATE_DATAFLOW_PATH
    (best_route, optimal_found);
```

The routing algorithm first calls *GENERATE DATAFLOW PATH* to generate routes of minimum turnings. It begins by studying the location of the two objects and the obstacles so that routes that will meet obstacles are not tried. It maintains the best route found so far. The best route is defined to be one with minimum crossing with other objects and with shortest distance. If the best route with zero crossing cannot be found, routes with increasing turning points are gradually considered. The maximum number of turning points considered is three.

GENERATE_DATAFLOW_PATH is a recursive procedure which generates a series of all possible routes between the two objects with the number of turning points specified. It partitions the routing problem into subproblems of generating subpaths. If a subpath overlaps with other data flows, alternate subpaths are considered. If the partially generated new route is found to be less promising (that is, with equal or more crossings) than the current best route, it is rejected and not explored further. The function *COMPARE_HEURISTICALLY* serves to select between the previous best route and the new route by comparing their data flow crosses. If the chosen best route has zero crossing, then an optimal route is found; otherwise, *GENERATE_DATAFLOW_PATH* is recursively called to backtrack and generate alternate paths.

At the end of the execution, the routing algorithm returns the best route found so far (whether it is optimal or not) and uses it as the data flow line for connecting the two objects.

7. FUTURE DIRECTION

Future effort will be directed on the specification of the primitive Process to describe its operations. These specifications are also called mini-specs.[1] Since AUTO-DFD aims to be fully graphical, an in-depth study and research on visual languages is essential. The mini-specs should preferably be specified through a two-dimensional flowchart-like representation. Furthermore, to make these flowchart-like specifications come to 'life', animation of input test data and output data can be visually shown on the screen during execution.

In addition, analysts will be able to specify the output requirement in a graphical form language.[15,16] The form definition language will use the arrangement of graphical objects to describe both the logical and visual structure of the output format. Forms can also be used to view the elements of the data dictionary.

8. CONCLUSION

This paper has presented the development of an automated data flow processor as an analysis tool to create data flow diagrams. It explains the operations involved in dynamically creating, moving and deleting objects and their related data flows. It also describes how the DFD processor allows the detail of a Process to be specified in a child diagram. This involves the integrity checking for the balance of net data flow inputs and outputs between the parent Process and its child diagram. It then elaborates on how the parent Process can be systematically replaced by its child diagram. Finally, the powerful data flow routing algorithm which relies on heuristics to find optimal data flow path between two entities is discussed.

REFERENCES

1. T. DeMarco, *Structured Analysis and System Specification*, Prentice Hall (1979).
2. An introduction to SADT. SofTech, Inc., Waltham, MA, document 9022-78, Feb. (1976).
3. S. H. Caine, and K. E. Gordon, PDL – A tool for software design. *National Computer Conference Proceedings*, **44**, 271–276 (1975).
4. D. Teichroew and E. A. Hershey III, PSL/PDA: A computer-aided technique for structured documentation and analysis of information processing systems. *IEEE Transactions on Software Engineering*, **SE-3** (1), January (1977).
5. A. M. Goh and C. P. Wong, POSE – An analyst workstation environment. *Proc. IT Works 1987*, pp. 157–164.
6. D. Shney, D. Bailey and T. P. Morrisey, PHIGS: A standard, dynamic, interactive graphics interface. *IEEE Computer Graphics and Applications* August, 50–57 (1986).
7. A. Kramer, IconMaker: interactive user interface design. *IEEE Computer Society Workshop on Visual Languages*, 192–198 (1984).
8. K. N. Lodding, Iconic Interfacing. *IEEE Computer Graphics and Applications*, **3** (2), 11–20 (1983).
9. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*. McGraw-Hill (1979).
10. G. A. Miller, The magical number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, **63** (2), 81–96 (1956).
11. M. A. Jackson, *Principles of Program Design* Academic Press, New York (1975).
12. E. Rich, *Artificial Intelligence*. McGraw-Hill (1983).
13. C. Y. Lee, An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, September, 346–365 (1961).
14. D. Hightower, A solution to line-routing problems on the continuous plane. *Proc. Design Automation Workshop* (1969).
15. S. Kazuo, T. Kikuno, N. Yoshida and M. Takayama, An approach to the design of a form language. *IEEE Computer Society Workshop on Visual Languages*, 171–176 (1984).
16. N. C. Shu, A forms-oriented and visual-directed application development system for non-programmers. *IEEE Computer Society Workshop on Visual Languages*, **SE-3** (1), 162–170 (1972).

Announcements

16–20 JULY 1990

ICALP 90, University of Warwick, England
16th International Colloquium on Automata, Languages, and Programming

Call for Papers

The 16th annual ICALP meeting of the European Association for Theoretical Computer Science (EATCS) will take place at the University of Warwick. Papers presenting original contributions in any area of theoretical computer science are being sought.

Scope

Topic areas include (but are not limited to): computability, automata, formal languages, analysis of algorithms, computational complexity, data types and data structures, theory of databases and knowledge bases, semantics of programming languages, program specification, transformation and verification, foundations of logic programming, theory of logical design and layout, parallel and distributed computation, theory of concurrency, symbolic and algebraic computation, term rewriting systems, computational geometry, cryptography, theory of robotics.

Papers

Authors are invited to submit seven copies (preferably double-sided) of an extended abstract or draft of a full paper before 15 November 1989 to the Chairman of the Programme Committee: Professor Mike Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK. Tel: +44 203 523194.

Location

The University of Warwick has an attractive campus with parkland, woods and lakes just outside the city of Coventry. All accommodation will be on campus within 5 minutes' walk of the conference rooms. Nearby places of interest include Coventry Cathedral, Warwick and Kenilworth Castles, Stratford upon Avon and the Cotswolds.

Travel

Birmingham Airport is very close and has direct flights from many European cities. The frequent trains from London to Coventry take only 75 minutes. The University is quickly accessible from the M1, M6 and M25 motorways.

Further information

Persons submitting papers from countries in which access to copying machines is difficult or impossible may submit a single copy. Please include an electronic mail address if appropriate.

Notifications of acceptance/rejection will be sent by 9 February 1990. Final papers for the *Proceedings* are due by 6 April 1990.

Further details about the conference (and the final programme) will be sent to anyone submitting a paper, to all EATCS members and to several electronic news nets in early 1990. To add your name to the mailing list, write to: ICALP 90, Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK, or send e-mail with a note of your return postal and electronic mail addresses to ICALP@cs.warwick.ac.uk (or appropriate net address as below).

Janet: ICALP@uk.ac.warwick.cs
Darpa:
ICALP%cs.warwick.ac.uk@nss.cs.ucl.ac.uk
Uucp: ICALP%warwick.uucp@ukc.uucp
Earn/Bitnet:
ICALP%uk.ac.warwick.cs@UKACRL