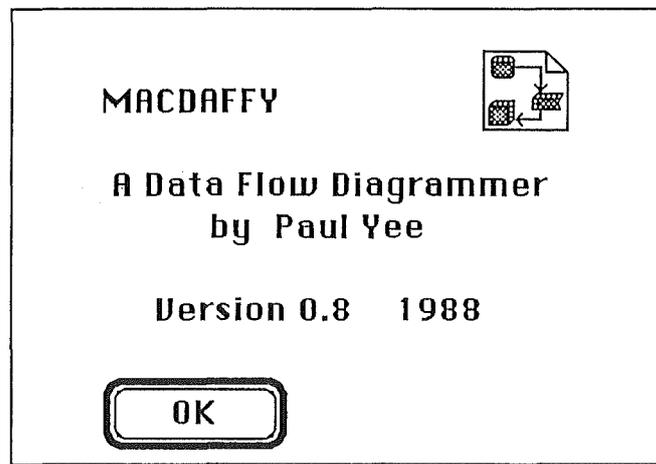


COSC 460 Honours Project  
Department of Computer Science  
University of Canterbury  
1988

# A Computer Tool for Structured Analysis



Supervisor: Dr. Neville Churcher

Co-Supervisor: Paul Ashton

Paul Yee  
4 October 1988

# Table of Contents

---

|   | Page      |
|---|-----------|
| <b>1.0 Introduction</b>                     | <b>1</b>  |
| <b>2.0 Background</b>                       | <b>2</b>  |
| 2.1 Systems Development                     | 2         |
| 2.2 History of Past Methods                 | 3         |
| 2.3 Structured Methods and Techniques       | 4         |
| 2.4 Computerization                         | 5         |
| <b>3.0 Data Flow Diagrams</b>               | <b>6</b>  |
| 3.1 Introduction                            | 6         |
| 3.2 Detailed Description of Symbols         | 6         |
| 3.3 Notation: Gane & Sarson vs DeMarco      | 7         |
| 3.4 Rules and Levelling                     | 9         |
| 3.5 DFDs: Pro's and Con's                   | 11        |
| <b>4.0 Manual/Automated Techniques</b>      | <b>13</b> |
| 4.1 Disadvantages of Manual Techniques      | 13        |
| 4.2 Advantages of a Computerised System     | 14        |
| 4.3 The Ideal Tool?                         | 16        |
| <b>5.0 Existing Diagramming Tools</b>       | <b>17</b> |
| <b>6.0 Design Issues</b>                    | <b>19</b> |
| 6.1 Exploiting the Macintosh User Interface | 19        |
| 6.2 Choice of Notation & Symbols            | 19        |
| 6.3 User Interface Design                   | 20        |
| 6.4 Operations                              | 21        |
| 6.5 Functionality of Cut, Copy & Paste      | 24        |
| 6.6 Routing                                 | 25        |

|                                  |           |
|----------------------------------|-----------|
| 6.7 Labelling                    | 26        |
| 6.8 Cursors                      | 26        |
| <b>7.0 Implementation Issues</b> | <b>28</b> |
| 7.1 Programming Environment      | 28        |
| 7.2 Data Structures              | 29        |
| 7.3 File Structures              | 32        |
| 7.4 Moving and Updating Objects  | 32        |
| 7.5 Problems Encountered         | 33        |
| 7.6 Specific Notes               | 34        |
| <b>8.0 The Solution</b>          | <b>38</b> |
| 8.1 Capabilities                 | 38        |
| 8.2 Examples of Use              | 38        |
| <b>9.0 Conclusion</b>            | <b>40</b> |
| <b>References</b>                | <b>41</b> |

## **Appendix I : Sample Output**

## **Appendix II : MacDaffy - User Guide**



MacDaffy

## 1.0 Introduction

---

The last couple of decades have seen a tremendous advance in computer technology. Computer information systems are now used in every facet of life for information storage and retrieval. In the commercial environment particularly, computer information systems have enhanced a business's ability to cope with large amounts of data, hence increasing its efficiency and potential scope of operation.

Unfortunately systems analysis techniques have not kept pace with the rapid expansion in size and complexity of computer information systems, resulting in an increased cost of development and maintenance [COUGAR]. The advent of structured techniques (refer section 2.3) during the mid 70's helped to bridge this gap, and the advent of computer tools should reduce the gap even more.

This report looks at the need for structured techniques and more, the need for computer implementations of these techniques. In particular I look at data flow diagrams (refer chapter 3), one of the most important analysis tools and document my own design and implementation of a computer tool for creating and manipulating data flow diagrams. **MacDaffy** (Macintosh data flow diagrammer) is aimed at increasing the productiveness of data flow diagrams as an analysis tool.

**Chapters 2,3** and **4** provide a background to systems development and introduce the need for computerized tools in general. I deal with data flow diagrams specifically in chapter 3. **Chapter 5** contains a brief survey of what tools are currently available and in **chapters 6** and **7**, I document the design and implementation of MacDaffy to overcome the problems which are discussed in the first 3 chapters.

## 2.0 Background

---

### 2.1 Systems Development

#### To Build and Maintain Information Systems

As organizations vary in size, complexity and functionality so the computer systems organizations use must vary in size, complexity and functionality. Systems Development is the term used to describe the process of developing an Information System to suit and cope with the information requirements of an organization. Systems development can be broadly split up into three iterative phases [HAWRYSKIEWYCZ].

- 1) Systems Analysis
- 2) Systems Design
- 3) Systems Implementation.

#### Systems Analysis

During this phase the analyst gathers data about the existing system (this may be a manual system requiring computerization or a computerized system requiring upgrading or redevelopment) if there is one and attempts to develop an understanding and a model of the user's requirements. Typically this requires the analyst to interview many of the people within the system, and to gain a comprehensive overall view of how the organization operates. "...he (the analyst) will be attempting to determine the essential nature of present information flows." [ALEXANDER]

#### Systems Design

During this phase, a design for the new information system is produced, based on the model resulting from the Systems Analysis phase.

#### Systems Implementation

As the name indicates the design of the new system which results from the previous phase is put into implementation here.

## **Maintenance**

It must be stressed however that information systems require on-going maintenance and on-going developments in response to changes and developments in the organization well beyond the initial development cycle. [HAWRYSKIEWYCZ]

Because the output of each phase is used as input to the next phase, it can be argued that the Systems Analysis phase is the most important. It forms the foundation on which the new system is created so if the model of the user's requirements is fallacious, then no matter how good the design and implementation of the new system, it will not do the job required. [GANE] shows that the cost of correcting errors increases the later it is detected. An error detected in the operating phase might cost fifty times more than if it had been detected in the requirements phase.

The discussion I have presented thus far has concentrated on the development of Information Systems, but the case is entirely analogous with program development especially as the distinction between "programs" and "information systems" is becoming less clear.

## **2.2 History of Past methods**

### **Techniques**

Prior to the mid 1970's, few formal techniques for describing systems and user requirements existed. Existing methods largely revolved around the use of natural language eg. functions within a system being described via an English narrative. During the 1960's the flow chart was a popular tool for systems analysis but fell out of favour during the 70's.

### **Problems with past methods**

The main problem with these techniques has been their lack of formalized structure. A lack of standards and communication means that the results can easily be ambiguous and difficult for people to follow and understand. Natural language is not suited for describing systems. English narrative is just too vague and long winded. [GANE]

Joe Celko [CELKO] describes the system flowchart as "almost completely useless" because rather than giving an overall logical view of the system, it describes the physical implementation of the system which is more the job of the designer.

## 2.3 Structured Methods & Techniques

During the mid 1970's, as systems began to grow larger and more complex, the idea of using structured modelling techniques to describe systems arose. Not only did this formalize techniques for describing systems, but it paved the way for providing formalized techniques for moving between phases as well (eg. converting the logical requirements into a physical design.). Hence ensuring that a much surer correspondence between analysis and design is achieved. [HAWRYSKIEWYCZ] discusses this more closely.

For the purpose of maintaining information systems structured, formalized tools which produce good documentation are essential. Especially when you consider that maintenance is often not carried out by the original developers but by different people many years down the track. Particularly important for maintenance is the use of active documentation tools, where changes in code are automatically reflected in the documentation, as opposed to just passive documentation. In the end, tools which generate executable specifications are the ultimate goal.

### **Tools**

A number of different structured tools exist and are discussed in [MARTIN]. Michael Jackson diagrams for instance are hierarchical diagrams which show the structure of input and output streams. Warnier-Orr diagrams model the decomposition of activities. No one tool satisfies all of the analyst/programmer's needs [MARTIN] so a number of different tools are used to model different aspects of the system.

DeMarco [DEMARCO] suggested in 1978 that data flow diagrams be the first tool used by the analysts to gain an overall view of the system (refer chapter 3).

## 2.4 Computerization

I have already mentioned the effect computer information systems have had, but computers allow us to do many other things faster, better and more efficiently as well. Not the least of the advances has been the advent of CAD (Computer Aided Design) packages for engineering and architecture.

The future of systems analysis and software engineering lies very much along the same lines with computerized implementations of structured development tools and this is reflected in the increasing number and popularity of computer tools. CASE/CASA (Computer Aided Software Engineering / Computer Aided Systems Analysis) is currently in its infancy but rapidly expanding as more and more companies enter the market. [GOERING]

## 3.0 Data Flow Diagrams

---

### 3.1 Introduction

Data flow diagrams were one of the structured diagramming techniques which became popular during the late 1970's, as an aid towards structured analysis. The data flow diagram was first used back in the 1920's in France to reorganize an office full of clerks. A diagram was created with a bubble for each clerk and an arrow for documents passing between clerks [PAGE]. Hence the first data flow diagram.

Data flow diagrams were formally introduced in 1978 by DeMarco [DEMARCO] for modelling data flowing through a system and the transformations they undergo. Gane & Sarson [GANE] in 1979 also helped to popularize data flow diagrams as an analysis tool, introducing a variation on the DeMarco notation.

By modelling data flows we obtain an overall view of the system as a whole. We can immediately realize the procedures and their data interfaces. Today the data flow diagram is one of the most popular diagramming methods.

### 3.2 Description of Symbols

Data flow diagrams are comprised of four basic symbols, the process, the data store, the external entity and the data flow.

#### **The Process**

The process represents an activity that transforms its input data into its output data. A process represented by a box on the diagram contains two pieces of information, a unique id to distinguish it from other processes in the diagram and a title defining its function.

#### **The Data Store**

The data store is a temporary repository of data within the system. No data is transformed here, it is merely an intermediate file for data travelling elsewhere. Like the process, data stores also have an id and a title.

## The External Entity

The external entity represents entities that are outside of the system being modelled. They are also called terminators, and sources & sinks because these entities are where the data originally comes from and where it eventually goes on leaving the system. A terminator might represent another department or the managing director. Terminators are associated with a title describing its function.

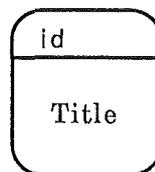
## The Data Flow

The data flow represents a flow of information between and connecting the other objects in the system (ie. processes, data stores and external entities.) Data (information) flows between objects, being transformed by processes and stored temporarily in data stores, eventually moving out of the system to the external entities.

### 3.3 Notation : Gane & Sarson vs DeMarco

Two notations are in common use today. The DeMarco notation [DEMARCO] and the Gane & Sarson notation [GANE]. The DeMarco notation (also known as the bubble chart because processes look like bubbles) was introduced in 1978, followed by the Gane & Sarson notation a year later.

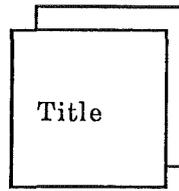
**Gane & Sarson:** A round cornered rectangular box for processes.



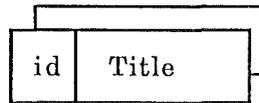
Horizontal and vertical lines for data flows.



A square box for external entities.



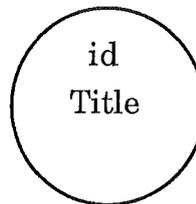
A rectangles for data stores.



Data flow diagrams are mainly a tool for modelling the logical functions of a system but the Gane & Sarson notation has optional additional features that allow you to model material flows as well eg. the physical flow of documents.

**DeMarco :**

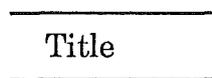
A circle for processes.



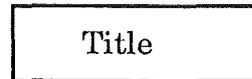
Straight lines (diagonal as well as horizontal and vertical) for data flows.



A pair of horizontal lines for data stores.



A rectangular box for external entities.



### **The Context Diagram**

Data flow diagrams are drawn starting with a context diagram. This diagram represents the entire system being modelled as one process and shows its connections to external entities. This single process can then be continually decomposed to show the systems functions in greater detail.

### **3.4 Rules & Levelling**

Levelling is the decomposition of processes into child diagrams with diagrams at successive levels showing more and more detail. The idea here is that in addition to inducing a logical top down process towards problem solving, levelling is an enormous help to the readability of the diagram.

Data flow diagrams have rules associated with them in order to maintain integrity of the diagram. [CELKO] These are an aid towards drawing correct diagrams which is in turn an aid towards the correct design of systems.

#### **Conservation of Data**

No object within the system (processes and data stores) generates data, it merely transforms it or stores it. The number of basic data elements is the same in the outputs and inputs. This means that for data stores and processes any data that comes out must first go in.

#### **Data Must be Moved via Processes**

Data cannot move directly between data stores and/or external entities. It must be moved via a process. Strictly speaking since we are only modelling the system, data flows outside the system do not concern us so there are no connections between external entities.

## No Control Elements

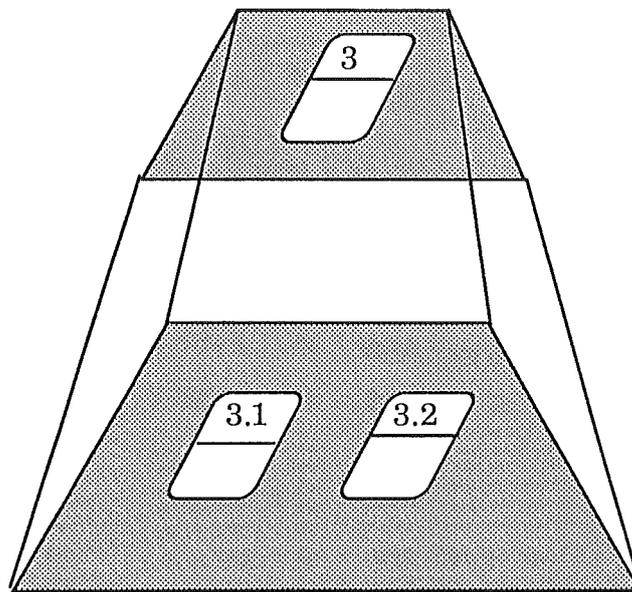
Data flow diagrams should not contain control elements. This includes signals to other processes, and looping control structures. Control elements belong in the lower level process specifications.

## Naming

Hawryskiewicz [HAWRYSKIEWYCZ] thought the assigning of names to objects important enough to dedicate a whole section to it. Although I shan't try to emulate this, it is important to realise that because there is only a small amount of text associated with data flow diagrams, it is important to make that text as meaningful as possible. Poor naming conventions reduce the DFD's usefulness as a communication and documentation tool.

## Duplicates

Joe Celko [CELKO] recommends that users duplicate external entities and data stores on a diagram to minimise the number of line crossings for the sake of clarity. Processes should on no account be duplicated on the same diagram, and any external entity that appears in a diagram should also appear in the context diagram.



## Labelling

By convention processes are labelled with a unique id that is derived from the parent and a unique number in its home level.

The above figure shows a parent of id 3 with a child containing two processes with id's 3.1 and 3.2

### **7±2 objects**

A rule that is generally adhered to is that the number of objects in a level (diagram) should be restricted to 7±2 (the magic number). There are three reasons for this magic number :

1) [MILLER] showed that 7±2 is the limit to what the human mind can assimilate at one time.

2) 7±2 objects is a reasonable fit onto one A4 page. The objects can be represented in a reasonable size.

3) Restricting the number of objects to 7±2 encourages top down decomposition. 7±2 is a non-trivial but understandable amount of information to put on one level.

### **Data Dictionary**

A data dictionary is necessary to record the actual details of the data flows, and data stores and the process specifications of the lowest level processes. [DEMARCO] describes the data dictionary in some detail.

## **3.5 DFD's : Pros and Cons**

### **Simplicity of use**

The simplicity of data flow diagrams is easy to see. Because there are only four symbols, data flow diagrams are extremely easy to learn, understand, use, and teach. The integrity rules that govern their use are intuitive.

Data flow diagrams, like the Macintosh user interface, can be described as user friendly. I think they make a good team.

### **Enhances Communication between Analyst and Clients.**

Ease of learning and simplicity of use are important factors to take into account when evaluating data flow diagrams. The very nature of the analyst's work requires that he interact with a large number of people.

It is important then that all parties be able to understand the diagram, so all parties can contribute effectively towards it.

Data Flow Diagrams are a natural and obvious way of modelling information flows and they have since become one of the most essential and widely used of system analysis tools.

## 4.0 Manual and Automated Techniques

---

### 4.1 Disadvantages of Manual Technique

In the past data flow diagrams have been drawn using pencil, paper and templates. The problems which arise from this are numerous.

#### **Potential size of diagrams**

In real life, data flow diagrams can become very large and complicated with the number of processes in a sizeable real life system in the order of thousands [PAGE]. Manipulating either a single diagram the size of a room, or a large number of inter-related, smaller diagrams would be time consuming, frustrating and error prone.

#### **Amount of Time required**

Drawing data flow diagrams by hand is time consuming compared to automated applications which only requires a number of mouse clicks.

#### **Maintenance and Editing of large diagrams**

No diagram, no matter how talented the analyst or how much insight he/she has, is going to be correct and complete after the first draft. Data flow diagrams require maintenance and constant revision. Not only within the development cycle, but also during the entire maintenance life cycle of the computer system. Diagrams need to be changed, objects need to be moved, deleted and added. Data flows may need rerouting or reconnecting, by themselves or as a result of other operations. Doing this with pencil and paper is awkward, untidy and non trivial. The problem here is that pencil and paper drawings (also drawing applications like MacDraw and Cricket Draw) are static in nature, they are merely objects on a page that have no attributes or relationships to other objects.

The disadvantages of manual techniques are most apparent where the need for a good tool is greatest i.e. for complex real life systems. The more complex the system, the greater the need for an efficient tool capable of producing good quality work. "As computer systems become more complex the probability of on-time completion, or even successful completion decreases at an alarming rate." [HUNGERFORD]

The problems which I have mentioned here are not only frustrating, but can seriously affect the usefulness and the popularity of data flow diagrams as an analysis tool. If a tool becomes too tedious to use then people will simply stop using it.

## **4.2 Advantages of Computerized Tools**

Drawing data flow diagrams is a job which is obviously suitable for a computer. In addition to overcoming the problems of the manual techniques mentioned above, computers have the ability to do a far more comprehensive and a far more effective job than possible with pencil and paper.

### **Simplicity**

A computer can relieve the analyst of the more tedious aspects of diagramming and leave him to concentrate on the important job at hand. A data flow diagram should be an aid to his/her work, not a job in itself. As in all things automation can improve the productivity and the effectiveness of a tool.

### **Speed of Development and Maintenance**

It is much easier to point and click than to hand draw using templates and it is not hard to appreciate the fact that it is infinitely easier to maintain a diagram with a dynamic tool, than it is to do with a static tool. Even programs like MacDraw which are professional and look tidy are too general to be of much use.

### **Display only Relevant Objects**

The limited size of the computer screen, which is so often a disadvantage, could be an advantage here because it encourages top down decomposition. It gets away from the monolithic single page diagrams of the past. The importance of top down decomposition cannot be underestimated because it is such an aid towards development and understanding.

### **Data Dictionary Support**

A computer can easily maintain a data dictionary, and perform cross referencing of entries to check that all references have been defined somewhere in the dictionary.

### **Ability to communicate with other diagrammers**

The potential for diagrammers to interface with other applications is a big plus in their favour. For instance linking in a data flow diagrammer with a Nassi-Schneiderman diagrammer [COUCH]. The data flow diagrammer provides an overall view and the resulting system specifications can be used as input to a Nassi-Schneiderman diagrammer to produce executable code. The future for automated diagrammers is bright especially considering the insufficient supply of programmers to cope with the ever increasing demand for software.

### **Integrity Checking**

As noted by Martin & McClure [MARTIN], drawing data flow diagrams byhand can be error prone. The computer can perform checks to ensure that the integrity of the diagram remains intact. For example, checking that there are valid connections between a parent and its children and that data flow connections are valid (refer sections 6.4, 6.5).

### **Routing Algorithms**

By implementing a routing algorithm to maintain flows between objects, a large weight can be removed from the shoulders of the analyst. A good line router can ensure that connections have the shortest path, have a minimal number of crossings, a minimum number of corners and are generally aesthetically pleasing. For a large diagram, a computer can do this far more efficiently and effectively than the human eye.

## PPP vs PPP

What it basically boils down to is the Paper and Pencil methods of the Past versus the Professional and Practical methods of the Present. It seems ironic that software design packages for hardware designers have been around in numbers for a lot longer than software packages for software designers. CASE/CASA tools have only come into prominence in the mid '80's [MARTIN]. The reason for this has been the late emergence of structured design techniques for software and systems development that have been around for a long time in hardware logic development.

### 4.3 The Ideal tool?

The ideal tool should embody all of the advantages described above in addition to being user friendly. I think that it is important to bear in mind that a computer diagrammer is a tool to enhance the users ability to do his/her job. It should be easy to learn to use and easy to use for all people not just the creators.

The simplest (friendliest) form of user interface is a completely visual one, with some sort of pointing device for user input. Much easier than directing with arrow keys.

Editing and integrity checking operations are all important to the maintainability of a diagram. Being able to check, move, delete, collapse and expand subtrees would make the users life much easier, and functions along these lines should be implemented. Chapter 5 documents the design of MacDaffy along the lines of the ideal tool.

## 5.0 Existing Diagramming tools

---

A number of graphics applications are available already, ranging in power, features, and usefulness.

### **Drawing tools MacDraw etc.**

Applications like MacDraw, MacPaint, Cricket Draw, and SuperPaint to name just a few are fully capable of drawing good looking data flow diagrams, but anybody who has tried to draw diagrams or any other pictures with these tools will appreciate the difficulty involved. (It's not as easy as it looks!) So apart from the time factor involved in development (which by itself is probably prohibitive), the fact that these programs are static in nature means that maintenance is tedious and integrity checking impossible. Moving an object will require the user to explicitly move the associated data flows him/herself. If you wanted to draw a single diagram for display purposes alone then these applications would suffice, but certainly they are insufficient for the professional analyst or software designer who requires a dedicated tool.

### **Auto-DFD**

Auto-DFD is a data flow diagrammer implemented on an Apollo Domain 580 graphic workstation at the University of Singapore. [CHUA] Chua, Tan & Lee have also realised the importance of a computerized tool for analysis. Auto-DFD is described as an intelligent data flow processor which has "a completely visual environment" with an iconic interface that will allow a naive user to "make effective use of the full potential of the system". I think that for this purpose, which is an admirable purpose, the Macintosh better fulfils the needs. An application developed on a lower end Mac (a personal computer) would be compatible with the powerful Mac II. The Macintosh interface is ideally suited for user friendliness.

## Commercial Tools

The increasing popularity of CASE has seen an increase in the number of tools available on the market. A number of them like Nastec DesignAid, Index Technology Corp. Excelerator, and Yourdan Analyst Designer Toolkit (all powerful analysis and design packages for the IBM PC's and compatibles) have been reviewed in [HALL]. These packages are expensive and designed for large and complex project development. eg. "Nastec DesignAid is a sophisticated package targeted at large engineering and data processing projects." [HALL]

The power and size of these packages can also mean that they have a steep learning curve. eg. "After two days away from the program (Nastec DesignAid) we found that we had forgotten most of the commands and had to relearn them". "... nonstandard keyboard interface and the plethora of options that confront the user." [HALL] This is where a simpler tool like MacDaffy can come into its own (refer chapter 6).

## 6.0 Diagrammer Design Issues

---

### 6.1 Exploiting the Macintosh User Interface

One of the reasons I decided to implement the data flow diagrammer on the Macintosh was what I considered to be the suitability of the Macintosh's famous user interface. In interactive graphics applications today, some sort of input device is common. In the past light pens and roller boards have fulfilled this role, but these have largely been superseded by the mouse. The mouse is as familiar and at home on the Macintosh as keyboards are on most other personal computers.

Another reason for implementing on a Macintosh is because of the increasing popularity of the Macintosh (especially the Mac II) in CAD/CAM graphics applications. The Mac II is being compared to and competing with workstations like the Sun 3/60G ("The Mac II and the Sun 3/60 offer similar performance, but the Mac II is easier to use and more responsive to user input.") and the Apollo DN/3000 ("The Mac overlaps with the Apollo in the following areas: Computer Aided Publishing, Mechanical Engineering, and Software Engineering") [SHEBANON]. The application would be compatible with the Mac II and the extra features of the Mac II like colour and screen size could be used to great advantage. Macintoshes, or at least the entry level Macintoshes, are reasonably affordable personal computers that I believe provide a good environment for diagrammers.

### 6.2 Choice of notation & symbols

The decision whether to use DeMarco's notation or Gane & Sarson's notation (refer section 2.3) was based on a number of practical reasons.

- 1) Diagonal lines on the Macintosh and other raster display computers of comparable power look jagged and unappealing.

2) It is easier to connect lines to rectangular sides than it is to circles because it is easier to calculate the intersection point of the data flow with the edge of a rectangle than the circumference of a circle.

3) It is easier to write text in a rectangle than it is in a circle, for a similar reason to that above. Alternatively it is possible to put more text in a rectangle than in a circle, which may sound a little trivial at first, but when you consider the size of the objects, then it becomes a little more important.

4) The data store icon in Gane & Sarson is clearer and better defined, there is no chance of confusion between a data flow and a data store.

5) It is generally easier to manipulate rectangles than it is to manipulate circles. eg. Checking whether a mouseclick is on an object. The cornerless circle might provide a few more problems on accuracy here.

6) Routing is an important feature of automated tools and horizontal and vertical lines lend themselves towards line routing algorithms. [BATINI]

### 6.3 User interface design

The user interface should be kept as simple as possible and still accommodate the applications needs. This means not abusing the Macintosh to the extent that the sheer volume of different user input possibilities overwhelms the simplicity and the advantages of the Macintosh interface.

The palette of icons has become a standard tool in graphics applications. This reflects their popularity and the intuitive ease with which they can be used. They are easy to access and easy to see what they do. Once again the idea that a picture is worth a thousand words. Only a small number of icons is required and this reflects the simple nature of data flow diagrams.

Pulldown menus are important because they are a way of implementing a large number of commands without taking up precious screen space.

## 6.4 Operations

The following are features which I decided were amongst the most important and the most realistic to implement. They aim to provide the application with a reasonably powerful set of functions with which to manipulate DFDs.

Double Clicking & Levelling : The ability to double click on the Macintosh is a powerful feature that should be put to the most effective use. Two possibilities for it are to bring up the information dialog or opening up a child diagram. I considered levelling to be the more effective use because it is the more common and more necessary operation. When a process is levelled the parent should be marked in some way to distinguish it from unlevelled processes. A little ampersand next to the id, is adequate.

Edit : The cut, copy and paste functions are powerful editing facilities that allow the user to do a lot and these form the foundation around which the programs features revolve. So this menu has a high priority.

File : An application would be much the poorer if it couldn't save, load and print diagrams after they had been drawn, so the standard Mac file menu is very important. The new and open operations should only refer to the context diagram, because all other windows are created by levelling (double clicking) an existing process.

Collapse : The ability to collapse objects and flows into a child diagram. It would almost be possible to do this just using the cut, copy, and paste operations, but considering that it is quite likely to be a common operation then collapse is a desirable and powerful editing operation to have.

Expand : This is the converse of the collapse function. It should merge the child diagram with its parent's diagram and remove both the child and the parent. Likewise this could also almost be done using cut, copy and paste.

Note: I say almost because the cut, copy, and paste operations would not reconnect the data flows to maintain integrity of the diagram, which both of the above functions should.

Remove children : The only way to remove a child diagram totally, is by deleting an entity and then creating a new one that hasn't been levelled but then you have to recreate the data flows and specifications as well, so this option is not as trivial as it might at first seem.

Reduced View / Normal Size : Map the entire level into the currently visible window so the user can see and manipulate the entire diagram in one window all at once. A very useful option indeed which is popular in many graphics applications.

Integrity Checking : Both batch and interactive integrity checks. Interactive checks are both user friendly, because they communicate with the user, and an aid towards keeping the diagram in a correct state. Some integrity checks can only be carried out when the diagram is completed. eg. Whether the number of flows connected to a parent is consistent with the number of flows in the child so these checks should be implemented in a batch option.

Routing : An algorithm to route data flows effectively and in a manner that is visually pleasing. Very important and possibly the most difficult implementation aspect to do well particularly with a reasonable response time.

Workspace : A separate workspace for designing mini diagrams which could then be pasted into the main diagram. With the proposed editing functions above, a workspace takes on a lesser importance . Any window could serve as a workspace, because you can cut and paste as you please.

Help : A help option is an attempt to make an application more user friendly. Since the facilities of this application are relatively simple and straightforward to understand, this feature takes a lesser priority.

Data Dictionary : Some means of letting the user specify low level information like process specifications and data flow elements. Macintosh dialog interaction is probably the simplest method of inputting text.

Traversing the Tree : Two extra operations implemented are the *Find* and *Context Diagram* options. These are designed to overcome the problem of numerous overlapping windows in a DFD's tree structure described in [HENDERSON]. Because a diagram may to be levelled (decomposed into more detailed diagrams) a large number of windows can be open at the same time, with the result that windows could become lost. *Context Diagram* brings the context diagram to the front and activates it, while *Find* asks the user for the id of an object and brings the window containing that object to the front. An alternative approach would be to have a menu item for every window that the user opens, as LightSpeed Pascal does. The problem here is that unlike LSP where the number of open windows is restricted to eight, the diagrammer can have any number of windows open, and such a menu list would be confusing and messy. *Find* in addition would have a dual purpose of being able to search out for a particular object if it exists.

Font & Style : The need for font and style menus in an application like this is debatable. The major reason for having font and style menus is to aid and enhance textual display material. While this is a graphical display program along the lines of MacDraw, MacDraw is a general all purpose static drawing application, whereas MacDaffy is much more. It is a specialized dynamic application that maintains relationships between objects, maintains integrity rules and assigns meaning and attributes to the various objects.

Text on a data flow diagram is largely restricted to the titles of objects and flows where I feel that uniform text would be a better choice for display purposes.

Certainly Font and Style menus are not difficult to implement but I consider them to be less important than other operations which should be supported first.

### 6.5 Functionality of cut,copy,paste.

These standard Macintosh editing commands should be implemented. Cut and copy should affect entire subtrees (a subtree refers to a parent process and all of its descendants) rather than just the objects at one level. So for example if you cut an object at the top level then all of its descendants would be cut and can subsequently be pasted back any where in the diagram as a single unit.

This is obviously much more powerful, and I consider, preferable to a one dimensional cut, copy. Being able to directly manipulate subtrees is desirable because all the levels descended from a process are related to it, and form all of that process's definition. (The idea of top down decomposition.)

Data flows should not be able to be copied or pasted, only cleared. This would have the effect of permanently deleting a flow from the diagram. The rationale behind this is that data flows form a unique dynamic relationship between two objects, it is not possible that they be pasted to different objects and there is no guarantee that both the original objects will still be there later on.

The effect that cut, copy, and collapse have on data flows that are connected to objects that have been cut, copied or collapsed should be different again. Flows can be put into two categories. Category 1 flows that are connected at both ends to objects in the same level and category 2 flows that have ends connected at different levels.

When an object or a group of objects are collapsed then all flows (category 1 and category 2) connected to any object within that group should also be collapsed. This clearly helps to maintain the integrity of the diagram. (See Appendix I for sample output)

Cut and copy are different from collapse because the application has no control over where the objects will later be pasted. So only those flows that are completely contained within the group should be copied.

This includes all category 1 flows that are self contained within the group and all category 2 flows within the group, but not category 1 flows that are not completely within the selected grouping.

## 6.6 Routing

Automatic routing is an extremely important feature in a data flow diagrammer because of the large number of lines which appear on it. Automatic routing is a problem which is well known in integrated circuit design. Numerous papers have been written about this topic and references for it would be overwhelming. However Peter Bromley's honours project [BROMLEY] contains a good discussion on routing in integrated circuits and contains some good references.

The main aim of an automatic router on a DFD is two fold.

- 1) To place lines in a diagram in a way which aids the readability of the diagram.
- 2) To place lines in a diagram in a way which is aesthetically pleasing to the eye.

Unfortunately the layout algorithms devised for integrated circuit design are not necessarily compatible for data flow diagrams because they follow slightly different aims, like minimising the circuit area.[BATINI] Batini, Nardelli, and Tamassia [BATINI] have suggested five ordered aesthetics relating specifically to data flow diagrams which satisfy these aims, and a routing algorithm which executes them.

Minimisation of crossings between lines.

Minimisation of the global number of bends in lines.

Minimisation of the global length of lines.

Minimisation of the area of the smallest rectangle covering the diagram.

Placement on the external boundary of symbols representing interfaces.

A line router is an essential component of a data flow diagrammer and the one proposed in [BATINI] which is designed for DFDs rather than for circuit design is promising. Unfortunately the sheer complexity of routing algorithms makes implementation of a complex one non viable. Perhaps this algorithm could be implemented for a future project. Incorporating a new algorithm into the program should not prove too difficult.

The solution is to port an already existing line router and modify it for my purposes. The router I chose was Li Sim Choong's line router for her Entity Relationship diagrammer [CHOONG]. I chose this router because it was available and could be implemented without too much difficulty. The modifications required concern the difference between data flow diagrams and ER diagrams. ER diagrams have different flow endings representing the cardinality of the relationship. Data flows only have directional arrowheads and require that all flows be represented individually.

## 6.7 Labelling

One of the design issues that came up was the representation of an object's unique id. The problem here is that a deeply levelled object's unique id can run into dozens of characters long. This is far too big to be able to fit it all into one small box. The answer is to put just the last integer (an id unique to the level) into the box, and the rest of the id can be picked up from the title bar of the window.

## 6.8 Cursors

The use of cursors in this application is an interesting point. Many Macintosh applications do follow a standard whereby the cursor represents the icon that is currently active. For a number of reasons I have decided not to do this.

I felt that the arrow cursor is an excellent general purpose cursor, because it is extremely obvious where the hot point is. In my own experience, not knowing exactly where the hot point is can be off-putting. Li Sim Choong's ER-diagrammer [CHOONG] seems to suffer from this fate slightly.

The other main reason for keeping with an arrow cursor, is that I wanted an icon to have a number of functions. For instance if the user clicks on an object icon then he can create the object in a window but he can also use it to select objects, move objects and level processes. The rationale here is that the user shouldn't have to keep going back to the palette every time he wants to move an object or level a process etc. I felt that changing the cursor to the icon would make this confusing.

For the same reason I do not think a hand icon, that would explicitly force the user to go back to the palette every time he wanted to move an object, is desirable.

## 7.0 Implementation Issues

---

### 7.1 Programming Environment

#### LightSpeed Pascal

This application was undertaken in LightSpeed Pascal with the Extender libraries. Pascal was chosen as the implementation language for two reasons :

1) It is the language with which I am most familiar. It is difficult enough to program the Mac without having to become familiar with another language.

2) The Macintosh toolbox routines are described in Pascal in Inside Macintosh [INSIDE] (the "bible").

Having had experience programming in Turbo Pascal on the Macintosh, It was obvious to me that LightSpeed Pascal is the superior programming environment.

LightSpeed Pascal's debugging facilities are excellent. The ability to step through the program, to observe the value of variables, and to set the value of variables during execution are invaluable.

Automatic formatting is something you only realise the value of when you are developing a reasonably large application under time pressure. Luckily LSP does provide a formatter. In addition LSP's separate compilation facilities make recompiling less of a chore.

LSP has a reputation for being the faster compiler, although I am not able to affirm or refute this claim.

#### Extenders

The Extenders are a library of routines that aims at providing an interface to programming the Macintosh. The Macintosh is notoriously difficult to program applications in and the Extenders does an excellent job at providing routines to do jobs that would otherwise have been very difficult and required a lot of technical knowledge.

The Extenders is both powerful and flexible at the same time. By default it uses its own routines to do many things like window manipulation, but at the same time you can add your own code on top of this (through the user hooks) or use your own code in place of the Extenders (through the Exception units).

The major dissatisfaction with the Extenders was that some of the routines did not seem to work properly. For instance, the SFTransfer procedure which would have made the transfer command ridiculously simple and the savepaint command did not work. Another minor grievance was that not all the source code for the Extender routines was provided. This caused problems later on. (refer section 7.5)

## Manuals

The Extender manuals [EXTENDER] were good in general, although slightly sloppy with a number of errors being found. eg. Procedures which were documented as functions etc.

The Inside Macintosh manuals [INSIDE] are excellent. Although the Extenders made them a little less important than they might otherwise have been, I still found them informative, helpful and readable. I would recommend them as the first place to look to start learning about Macintosh programming.

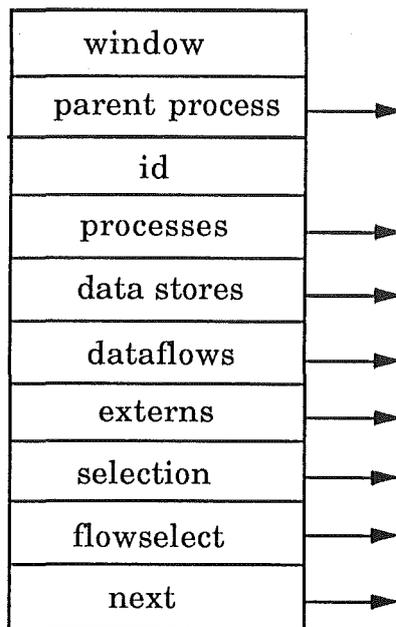
The LightSpeed Pascal manual [LSP] was useful for coming to grips with the LSP programming environment, but other than that was not really needed.

## 7.2 Data Structures

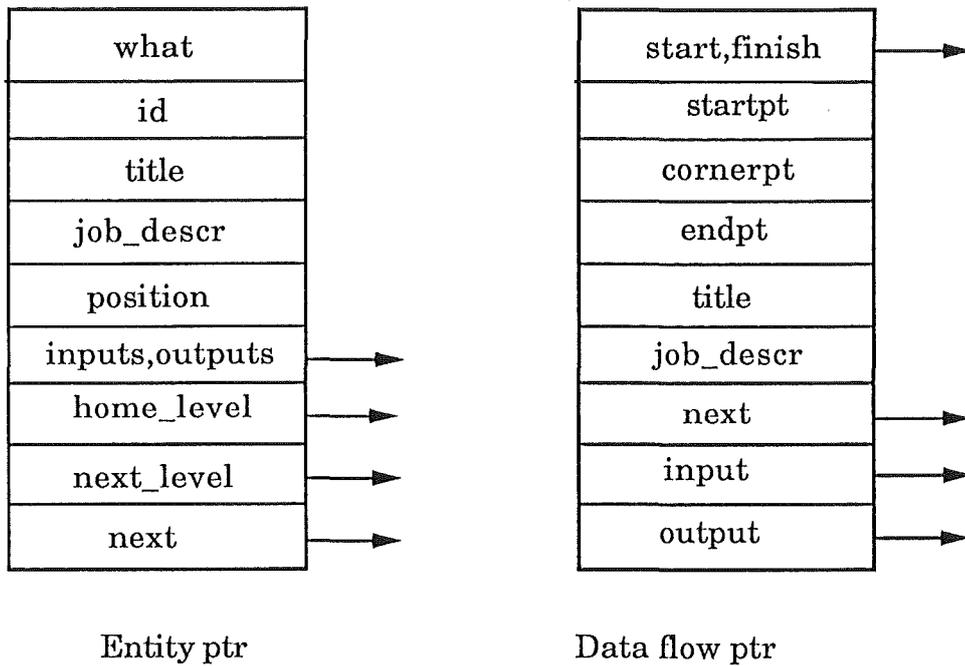
Complex interrelated data structures were required to represent the tree structure and the dynamic nature of data flows. This resulted in a large number of pointers and linked lists. The problem with a linked list of course is that once it gets very large then response time will be affected. However the requirements of a tree structure, with a dynamic number of elements, and with an eye on memory space made linked lists the best option. This is another good reason for keeping the number of objects in a diagram to  $7 \pm 2$ .

My original design for the data structures was based around five distinct pointers. One for each object (process, store, entity), one for the flows, and one for the levels (diagrams). This soon became unwieldy because it meant that three copies of each procedure were required that did the same thing but with different pointers. I later merged the three object pointers into one 'entity\_ptr'.

The subsequent three pointers (level, data flow, entity) then form the basis around which the program revolves. Each level pointer contains four linked lists of processes, data stores, terminators, and data flows. The processes, data stores and terminators are in turn related and connected via the data flows. So as you can see pointers proliferate.



Level ptr



### The Level

The *selection* and *flowselect* pointers in the level record are for saving selected objects for each level. This hasn't been implemented yet, but the pointers are there (selected objects are deselected when the window is deactivated). *Parent process* is a pointer to the levels parent process in the level above.

### The Data Flow

The data flow record has five pointers. *Start* and *end* point to the objects the data flow connects. *Next* connects the flows in a linked list so we can easily traverse all the data flows in a level. *Input* and *output* connect all the flows connected to an object so we can quickly determine and access all of an objects inputs and outputs. A data flow can only be an input or output to one object.

In DFDs, it is possible for a data flow to have one of the ends in empty space. This represents a connection to an object at a higher level. These ends have a dummy object associated with it which allows the user to drag "unconnected" ends. Dummy objects are distinguished from real objects by setting their *what* field to type *flow*.

## The Entity

The entity record has five pointers. *Inputs* and *outputs* are linked lists of an objects input data flows and output data flows respectively. So we don't have to redraw all the data flows if we are only moving one object and also for integrity checking. *Next\_level* is a pointer to a child process if one exists (obviously for data stores and external entities this pointer will always be nil), *next* is a pointer to the next entity in the levels linked list. *Home\_level* is a pointer to the level which contains the entity. *What* is an enumerated type(*process, store,flow,extern*) specifying what the entity is.

## 7.3 File Structures

The file structures are a modification of the program's internal data structures, with the pointers replaced by strings, so I can reconnect structures when they're read back in. In addition each level record I save has a field specifying the number of objects of each type in that level. The first record in the file is a header specifying the number of levels in the tree, so I know how many levels and objects to read back in.

Data flow diagrams are a top down decomposition tool, and I make use of this fact when saving and loading diagrams. Because diagrams are top down decomposed, I can save them going down a linked list, and when I read all the objects in a level back in, I know that the level (window) will already have been created.

One real problem with this though is that because there are a large number of strings to be saved, the files can get very large quite quickly. The way to get around this would be to compress the text, and/or save only the relevant part of the strings.

## 7.4 Moving and Updating Objects

The drawing and redrawing of objects is an important consideration in a dynamic graphics application like MacDaffy.

One possibility is to simply make the change in your data structures and redraw everything.

A consequence of this is the flickering of part or all of the window, where objects are being redrawn unnecessarily. Another problem is that it tends to be slow because it has to go through all the data structures and redraw all the objects.

An alternative (which is what MacDaffy does) is to only redraw the area of the screen that has been altered by the users actions (by setting the clipping region or otherwise). For example, if the user moves an object from one part of the screen to another then erase the old object and redraw it at the new position directly. With this method however you have to be careful that whenever you move an object, anything underneath gets redrawn. There are however two complications to this scheme, data flows and data flow titles.

As I have mentioned before MacDaffy is a dynamic application that maintains relationships between objects and their data flows. So when an object is moved the data flows must be moved with it. Because the routing algorithm allows data flows to go through objects, we want objects to be drawn on top of the data flows. (The other way around looks ungainly). If we only want objects to be redrawn if they have been directly manipulated by the user, then we have to check that when we redraw the data flows, anything that was under the new data flows gets redrawn on top, and this is what MacDaffy does by checking for intersection between data flows and objects. The other complication are the data flow titles. The same discussion applies for data flow names as for data flows, only trying to calculate intersections for names as well is slow, complicated and not all that effective. Therefore when data flow names are being shown the whole window gets redrawn when the user makes changes. (refer section 7.6)

## 7.5 Problems encountered

### Updating

One problem I encountered using the Extenders was the update procedure. The Extenders provide their own default update procedure, to which you can add your own code in the user hook procedures. Because the source code for the update procedure was not provided, it took a long time for me to realise that the Extender's update was doing a beginupdate; ... endupdate; and removing the update region for me.

The solution was to put the update procedure in exception routine rather than the user hook.

### **Printing**

Another problem which may have been more easily solved with the appropriate source code, was that the *printpic* command seems to do something funny with the clipping region, so that any command to set the clipping region immediately after execution of "printpic" (before execution of some other event I think) causes an out of memory error to occur. A consequence of this error is that the option to print all the diagrams does not work at all. There is no way to remove all the cliprects inbetween printing numerous diagrams. Even worse because the application does manual scrolling and updating, there is a constant need to offset the origin and reset the clipping region. It is possible to print one diagram but then the LSP program crashes. Printing in the application is flaky at best.

### **Application Building**

A major problem encountered during the development of this application, was that the build application option in LSP didn't work. The result was that a "resource not found" error occurred. The solution was to rebuild the entire project, using the volume two project instead of the volume one project which I had been using up to then.

## **7.6 Specific Notes**

### **Editing**

Because of the tree structure of leveled data flow diagrams, recursion was used a fair bit in the application. Especially for cutting and copying of subtrees (processes and their children) and for deleting of subtrees. The editing facilities revolve around the two functions copy and clear. A cut is a copy to the clipboard followed by a clear of the diagram. A paste is a copy from the clipboard to the diagram, and a collapse is essentially a cut followed by a paste. So as you can see copy and clear are very powerful functions. Clear deletes the objects from the data structures, while copy makes a copy of the selected objects and appends them to the lists of the clipboard.

## Moving Objects

I noted that in MacDiammer [CHOONG] if you move an object over another, it gets erased, and everything is redrawn once the user has stopped moving. This results in the whole window flickering. An unappealing feature. I have tried to get around this by doing two things. Rather than dragging the actual object, just dragging an inverted box and making sure that I only redraw what needs to be redrawn.

## Scrolling

A similar idea is used in the scrolling. One possibility was to create a picture and associate it with a window. Then all the scrolling and updating could be done by the Extenders because it knows what to redraw. Unfortunately this results in a flickering of the whole window, when the picture is redrawn. What I did instead was manually redraw the contents of the window whenever an update event was sent.

## Show/Hide

Two extra options not discussed above have been implemented. The first, show/hide dialogs turns off the dialog boxes which come up when a new object is created. There are two reasons for the implementation of this option.

1) During the process of developing and testing this application, there were times when I needed to create a large number of objects and didn't want the dialogs popping up while I created them.

2) It allows the user to create the outline of the diagram before adding the specifications, this is simply another option for the tastes of the user.

The second option show/hide flow names is for speed sake. Drawing flow names complicates things, (refer section 7.4) so turning off the flow names means that redrawing is done noticeably faster and without the flickering. Diagrams can be developed with flow names turned off, and then turned on at the end.

## Icon Palette

Another consideration was the palette of icons. The ideal solution for this would be to have a separate window of icons which would behave like an ordinary window, so you could move it around and send it away. Unfortunately, because you can only have one active window at a time on the Macintosh, this results in windows activating and deactivating every time you click on an icon and having to click twice to get to the icon. Activation and deactivation is very obvious from the highlighting of the title bars. The solution here was to put the icons in their own grafport, but not a window. This means that movement of other windows has to be prevented from moving over the palette. It won't get redrawn by the toolbox, because unlike a window, the toolbox doesn't know about it.

## Expand

Ideally this option should perform the exact reverse of collapse. Where collapse maintains integrity by ensuring that all the data flows are properly reconnected, the expand option currently does not. This should be a minor extension.

## Workspace/Clipboard

Note that although a workspace (scrap window) has not been implemented as such, the clipboard does allow the user to manipulate objects as any other window, with the exception of levelling. So you can create objects and flows in the clipboard which can then be pasted into the main diagram. Cut, copy and paste do not work on the clipboard, but clear does.

A special clipboard (*myclip*) not accessible by the user is used for collapsing and expanding.

## Highlighting

Some way of distinguishing the highlighted objects/flows is needed. Using little black boxes around the corner of the selected objects is standard for the Macintosh and this is what's done. This won't work for data flows though and turning them a different colour seemed the best option.

### Reconnecting

Copying data flows around is difficult because it means finding its newly created owners and reconnecting them. In aid of this, every object has a special unique id (different from the *id* the user sees which is not guaranteed unique) that is assigned on creation of a new object. One tricky point is that it is possible for the user to paste objects back on top of the originals. So any copied data flows will find the originals before the copied versions and become connected to the originals. Even worse the user can make multiple copies of the same thing to the same diagram. These problems are overcome by making a slight amendment to the guaranteed unique id's of the clipboard copies just before pasting (a '.c0', '.c1', ... is appended).

### Resizing

Shrinking and resizing objects on the Macintosh is easier than it sounds. The Macintosh toolbox provides a number of functions to map points from rectangle to rectangle, and this is what I do when diagrams are put into reduced view. I map all the objects from the virtual window into the visible window. In addition no text is drawn while in reduced view. This is mainly because the objects are too small to be able to support any reasonable amount of text.

## 8.0 The Solution



MacDaffy

---

### 8.1 Capabilities

MacDaffy as it stands at the moment is a practical friendly application with potential for expansion. Obviously MacDaffy is not the complete ideal tool but I believe that it goes some way to satisfying my original objectives (refer sections 3.2, 3.3). MacDaffy is simple to use, has good drawing facilities, provides powerful editing features, maintains integrity, performs automatic line routing, and supports a crude data dictionary.

Of the operations discussed in chapter 6, not implemented are the font & style menus, and the workspace window. (Although the clipboard is very flexible (refer section 7.6)). The other operations are implemented and the result is an application which is capable of the development, maintenance and error checking of data flow diagrams. MacDaffy overcomes many of the problems which are so evident with the paper and pencil methods of development and certainly improves the productivity and usefulness of DFDs (refer chapter 1).

### 8.2 Future Extensions

An obvious extension that could be performed would be to try and integrate MacDaffy with another computerized analysis tool, possibly either NS-Chart [COUCH] or MacDiammer [CHOONG] both diagrammers for the Macintosh.

Something I am keen to see get done, is the implementation of a powerful routing algorithm. The one MacDaffy uses at the moment suffices but is rather restrictive.

Printing of diagrams is an important feature and since the extenders routine doesn't work properly, this will probably have to be done the long way. I am not sure how difficult this is.

A number of smaller features I would also like to see implemented are :

- Automatic scrolling when dragging a data flow or an object outside the window.
- Using rubber band boxes to select a number of objects.
- Printing of the lower level process specifications.
- The reconnecting of data flows when a child diagram is merged with its parent.

## 9.0 Conclusion

---

This report began by describing the importance and the need for structured analysis tools in today's environment where systems are larger, more complex, and more numerous. It goes on to describe the need for computer implementations of these tools to improve productivity, quality and integrity of diagrams, and the potential of diagramming techniques in general.

There can be no doubt that CASE/CASA tools are becoming more and more prominent as professionals become increasingly aware of their usefulness and potential. (Especially with the drop in price of increasingly powerful personal computers and workstations.)

In this report I have described the design and implementation of such a computer tool that tries to match the ideals of my ideal tool, for developing and maintaining data flow diagrams. One of the most popular and important diagramming techniques.

In MacDaffy I have designed and implemented a user friendly tool that makes the most of a computer's ability to automate otherwise tedious manual processes. MacDaffy is an application that allows DFDs to be developed and maintained, and has potential to be further developed and integrated with other tools.

In conclusion I would like to express my gratitude to my supervisor, Dr. Neville Churcher, for the interest, advice and help he has provided during the course of my project and to my co-supervisor Paul Ashton for his expertise on data flow diagrams.

## References

---

- [ALEXANDER] Alexander M. J.  
'Information Systems Analysis'  
Science Research Associates 1974
- [BATINI] Batini C. Nardelli E., Tamassia R  
'A Layout Algorithm for Data Flow Diagrams'  
IEEE Transactions on Software Engineering  
Vol. SE-12, No. 4  
April 1986
- [BROMLEY] Peter Bromley  
'A Diagrammer for the EXSYS Data Model'  
Honours project report  
Department of Computer Science  
University of Canterbury 1985
- [CELKO] Joe Celko  
'Data Flow Diagrams'  
Computer Languages  
Vol. 4, No. 1  
Jan. 1987
- [CHOONG] Li Sim Choong  
'MacDiammer, A Graphical Design Tool'  
Honours project report  
Department of Computer Science  
University of Canterbury 1987
- [CHUA] Chua, Tan, Lee  
'Auto DFD: An intelligent Data Flow Processor'  
Department of Information Systems and  
Computer Science  
National University of Singapore.

- [COUCH] Andrew Couch  
'N-S Chart'  
Honours project report  
Department of Computer Science  
University of Canterbury 1988
- [COUGAR] Cougar J. D. & Knapp R. W.  
'System Analysis Techniques'  
John Wiley & Sons, Inc. 1974
- [DEMARCO] DeMarco T.  
'Structured Analysis and System Specification'  
Yourdan Press 1978
- [EXTENDER] 'Programmer's Extenders'  
Vol. I & II  
Invention Software Corporation
- [GANE] Gane C. & Sarson T.  
'Structured Systems Analysis: Tools and  
Techniques'  
Prentice Hall, Inc. 1979
- [GOERING] Richard Goering  
'Design Tools Advance to Keep Pace with  
System Complexity'  
Computer design  
December 1987
- [HALL] Hall & Keuffel  
'Software Reviews - System Design from the  
Ground Up'  
Computer Languages  
January 1987.

- [HAWRYSKIEWYCZ] Hawryszkiewicz I.T.  
'Introduction to Systems Analysis and design'  
Prentice Hall 1988
- [HENDERSON] Henderson D. Austin & Card Stuart K.  
'Rooms: The Use of Multiple Virtual  
Workspaces to Reduce Space Contention  
in a Window-Based Graphical User  
Interface'  
acm Transactions on Graphics  
Volume 5, number 3  
July 1986
- [HUNGERFORD] Ken Hungerford  
'Stations Popular as a CASE Platform'  
Computer World (Australia)  
August 19 1988
- [INSIDE] Inside Macintosh  
Vol. I,II  
Apple
- [LSP ] Lightspeed Pascal Programmers Manual
- [MARTIN] Martin J. & McClure C.  
'Diagramming Techniques for Analysts and  
Programmers'  
Prentice Hall 1985
- [MILLER] Miller G. A.  
'The Magical Number 7 plus or minus two:  
some limits on our capacity for  
processing Information'  
Psychological Review 63  
pg 81-97 1957

[PAGE]

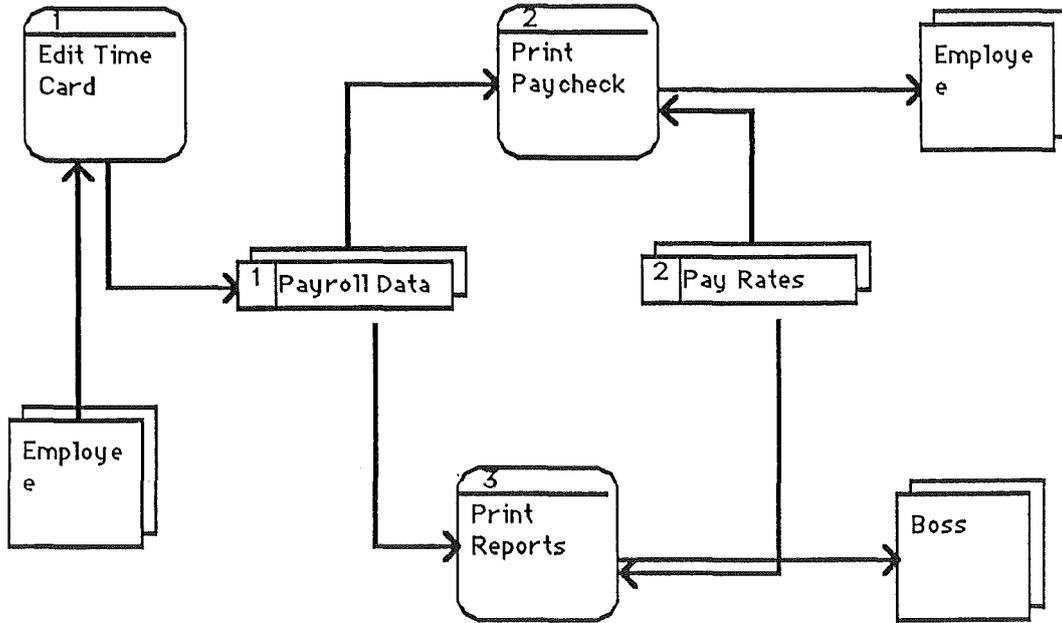
Page-Jones M.  
'The Practical Guide to Structured Systems  
Design'  
Yourdan Press 1980

[SHEBANON]

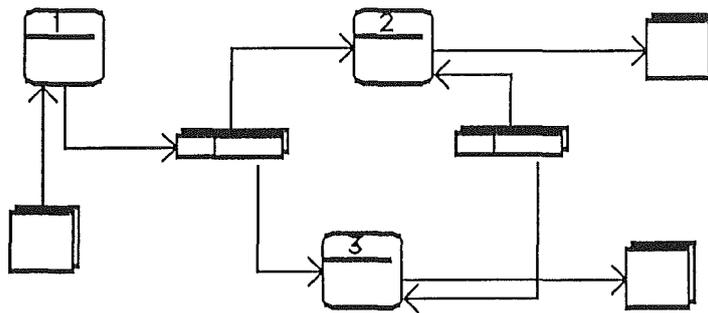
Andrew Shebanon  
'Pitting the Macintosh II against the Sun 3/60G'  
&  
'Round two: Macintosh II versus Apollo's  
DN/3000'  
Computer World (NZ)  
August 1988 No. 79

# Sample Output

---

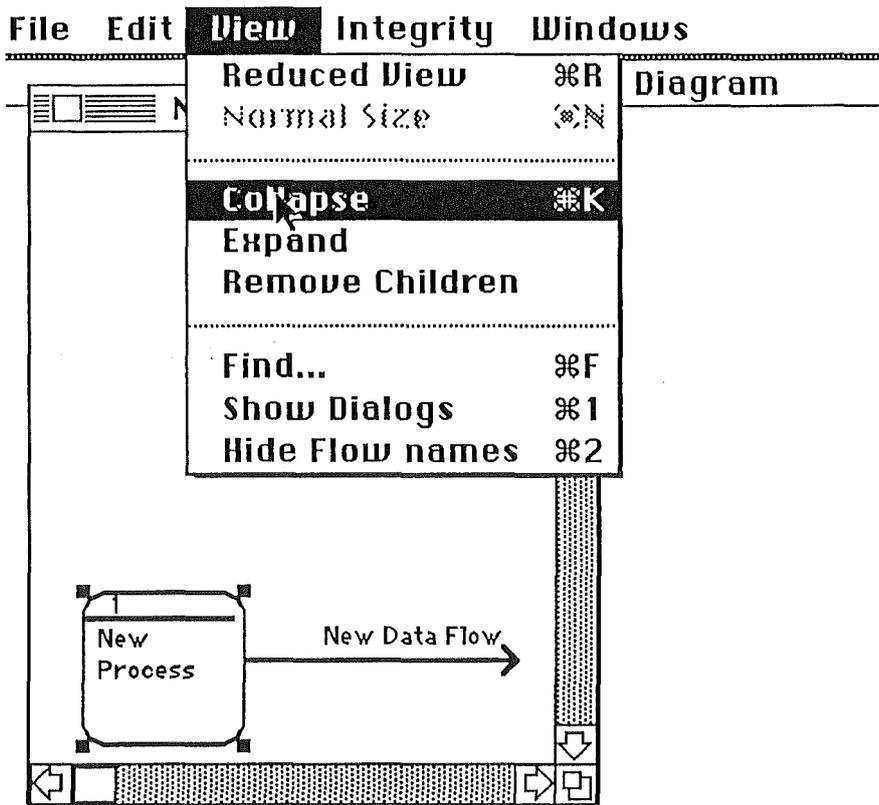


NORMAL SIZE DIAGRAM



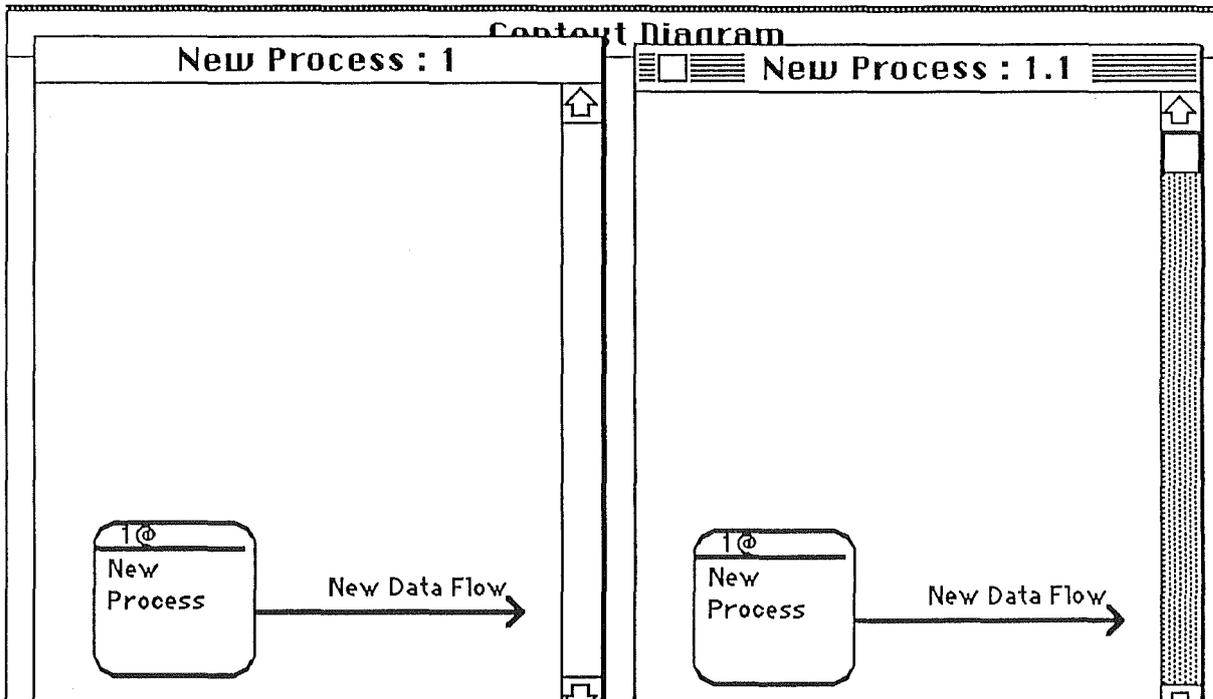
REDUCED VIEW DIAGRAM

BEFORE COLLAPSE

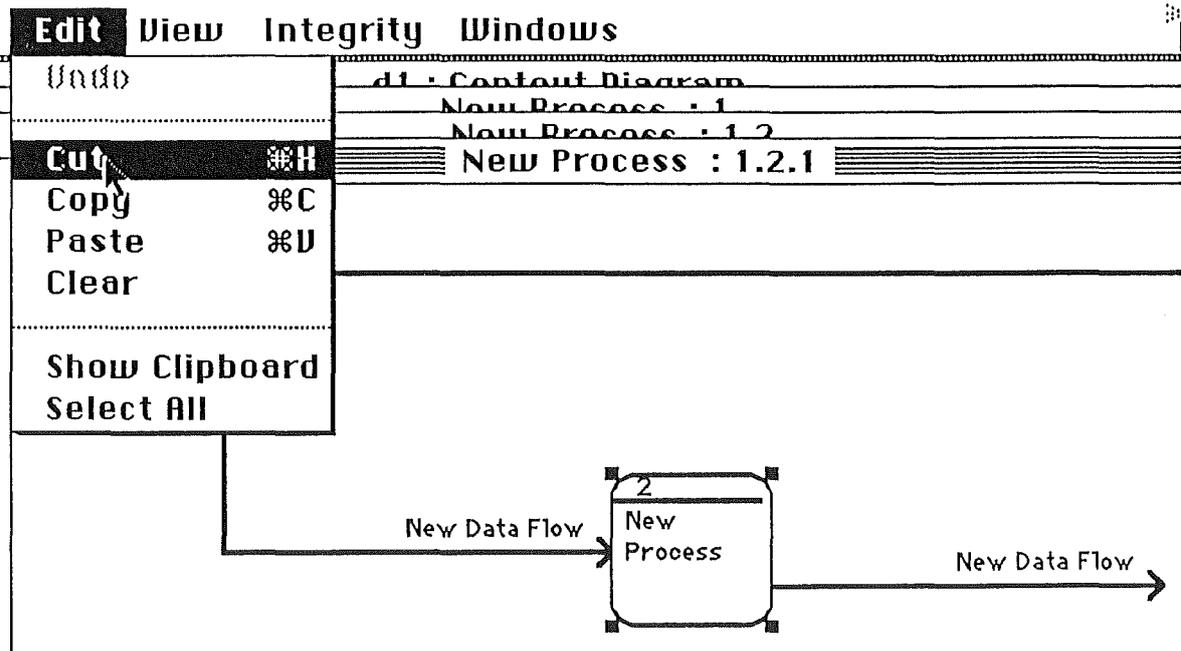


# AFTER COLLAPSE

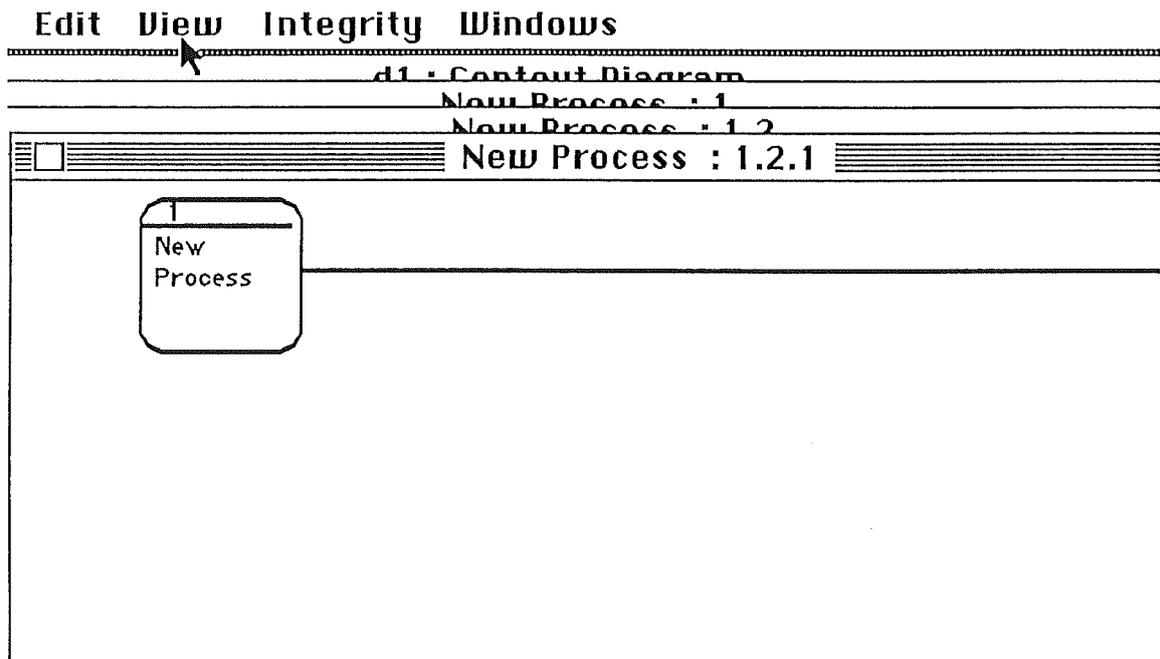
File Edit View Integrity Windows



BEFORE CUT



AFTER CUT



# MacDaffy User Manual



MacDaffy

---

## Introduction

MacDaffy (Mac dataflow diagrammer) is an interactive graphics application, developed on a Macintosh SE computer, written in Lightspeed Pascal using the Extenders, and intended for use on a Macintosh computer.

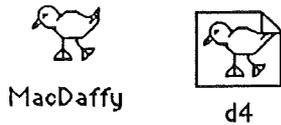
MacDaffy was designed primarily as an aid to Systems Analysis, by allowing the user to create and manipulate dataflow diagrams. (A major analytical tool)

This manual is intended for people who are already familiar with both dataflow diagrams and the Macintosh user interface, and wish to learn how to use this application. For information about data flow diagrams themselves, see the report which accompanies this manual or any relevant textbook. (The report contains some good references.)

## Notation

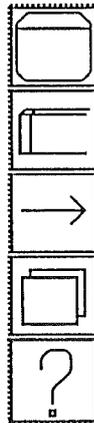
In order to dispel ambiguity, the following notation convention will be adhered to. *Italics* will be used to indicate menu commands. The term **level** will be used to mean a node in the tree structure resulting from top down decomposition and is largely interchangeable with the terms window and diagram. The terms **object** and **entity** will both collectively refer to processes, data stores and external entities.

## Starting up



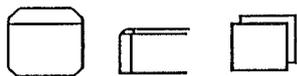
Double click on MacDaffy application, or on any file created from MacDaffy, to start it up. An empty Context diagram will appear ready for the user. Select *close* followed by *new* to create a new data flow diagram or select *close* followed by *open* to edit an existing diagram.

## Using the Icons



Each of the first four icons in the palette to the left of the windows represents a different symbol used in dataflow diagrams. The fifth icon is an information icon. Choose this to click on objects and data flows to bring up an information dialog box.

## Processes , Data Stores , External Entities



To create an object in your diagram, click on the appropriate icon. and then click the mouse at the position in the window where you want the object to appear. A dialog will appear asking the user for information relating to the object you have just created.

**NEW ENTITY**      **OK**      **CANCEL**

**Id**      1      

**Type**      **Process**

**Title**     

**Job Description**     

## Data Flows



The arrow icon is used to create data flows. Click the mouse in the window where you want the data flow to start from and then drag it to where you want it to end. Starting or ending your drag at an object will automatically connect that flow to that object. At any level other than the context diagram, data flows may start or end in empty space but not both. An unconnected end indicates a connection to an object at a different level.

## Information



The question mark icon brings up an information dialog box about the object clicked on, which you can then edit.

## General Operation

Objects may be selected in the traditional Macintosh manner by clicking on them. Data flows can be selected in the same way, by clicking anywhere along them. To select a number of objects, shift-click on the objects.

A highlighted dataflow is indicated by turning it gray, while a highlighted object is indicated by the little black boxes at its corner in the standard Macintosh manner.



Process 1 and the data flow are highlighted, but process 2 is not.

Processes may be levelled by double clicking on them, this opens up a new child window where the parent process may be further detailed. This is an aid to topdown decomposition and readability. A general rule that is not enforced by the program but should be adhered to by the user, is to restrict the number of processes in a level to  $7 \pm 2$ . This is a non-trivial number of processes, but at the same time provides an understandable amount of information in one diagram.

Once a flow has been created, moving the object it is connected with will automatically move the flow as well. Flows that do not start or end at an object can also be moved by selecting a point just past the end of the flow.

## Menu Commands

🍏 File Edit View Integrity Windows

Note that many of the menu commands have corresponding keyboard commands. Pressing the apple key followed by a letter or number has the same effect as selecting the menu option from the menu bar.

| File          |    |
|---------------|----|
| New           |    |
| Open...       | ⌘O |
| -----         |    |
| Close         |    |
| Save          | ⌘S |
| Save as...    |    |
| Save Paint    |    |
| -----         |    |
| Page Setup... |    |
| Print Window  | ⌘P |
| Print Diagram |    |
| -----         |    |
| Quit          | ⌘Q |

The **file menu** commands are standard to the Macintosh and will not be elaborated upon here. The only point to note is that the *new*, *open*, and *close* commands all refer to the context diagram. All other windows are created and opened by double clicking their parent process and closed by clicking in their goaway box.

| Integrity     |    |
|---------------|----|
| Check Window  | ⌘W |
| Check Diagram | ⌘D |
| -----         |    |
| Show Errors   |    |

The **integrity menu** has two commands (*checkdiagram* and *checkwindow*) designed to error check all the windows or just the window you are currently working on. The errors will appear in the errorwindow. If there are no errors then nothing will happen. More information about integrity checks can be found in the report.

The *showerrors* command brings up the *errorwindow* with the errors from the last error check you made so you can correct the errors one by one.

| View            |    |
|-----------------|----|
| Reduced View    | ⌘R |
| Normal Size     | ⌘N |
| -----           |    |
| Collapse        | ⌘K |
| Expand          |    |
| Remove Children |    |
| -----           |    |
| Find...         | ⌘F |
| Hide Dialogs    | ⌘1 |
| Hide Flow names | ⌘2 |

The **view menu** has eight commands.

The first two (*reduced view* and *normal size* ) alter the size of the diagram within the window.

*Reduced View* maps all the objects in the currently active level into the visible window so the user can see and manipulate everything at once. While in reduced view mode window resizing is disabled so that the entire diagram remains in view. Since no automatic scrolling is performed when you are dragging data flows, you must use reduced view mode to connect objects that are a long distance apart.

*Normal size* performs the opposite operation and maps all the objects and their data flows back to their original position and size.

*Collapse* collapses the current selection into a child diagram. All flows that are completely self contained within the selection will be collapsed. This includes flows that are only connected to an object at this level at one end. The parents data flows are reconnected properly.

The *expand* command does the opposite and incorporates the child diagram back into its parent. At the time of writing version 0.8 does not reconnect the child data flows with its parent.

*Remove Children* deletes and irreversibly deletes any children associated with a parent.

The *Find* command brings up a dialog box and searches for an entity by id number.

The *Hide Dialogs* command turns off the dialogs which come up

| Edit                  |    |
|-----------------------|----|
| Undo                  |    |
| -----                 |    |
| Cut                   | ⌘K |
| Copy                  | ⌘C |
| Paste                 | ⌘V |
| Clear                 |    |
| -----                 |    |
| <b>Show Clipboard</b> |    |
| Select All            |    |

The **edit menu** implements the standard Macintosh editing functions *cut*, *copy*, *paste* and *clear*. (undo is not implemented) The *cut* and *copy* commands will cut and copy the selected objects/flows from the current window into the clipboard. The effect cut and copy have on the flows connected with selected objects is complicated and require elaboration. When an object (process, data store or external entity) is cut, one of two things will happen to its associated flows. If the flow is connected to another (unselected) object then this flow will be deleted, if the flow ends in space (ie. not connected to an object) then the flow will be cut along with the object. A rationale behind this is documented in the report. Cutting flows by themselves will simply delete the flow; they cannot be copied and pasted.

*Show Clipboard* activates the clipboard, where you can see what has currently been cut or copied. Note that objects in the clipboard itself can also be manipulated. You can put new objects into the clipboard and move them around but you cannot level processes.

*Hide Clipboard* is the converse and sends the window away.

*Select All* selects all the objects in the current window.

| Windows             |    |
|---------------------|----|
| <b>Show palette</b> |    |
| -----               |    |
| Help                | ⌘H |
| Context Diagram     | ⌘T |

The **windows menu** contains three commands. The first, *show palette*, redraws the palette of icons. This is necessary if something happens to erase part or all of the palette. (Pyro for instance will do this).

The second menu option is the *Help* option. This will bring up a window with text on it describing the features of MacDaffy. *Context Diagram* will bring the context diagram to the front and activate it. Especially useful, when a large number of windows are opened, and the context diagram becomes lost.

## Hints

Begin your data flow diagram by drawing a context diagram. A context diagram is a very general diagram which shows the system as a single process and its interactions with external entities. You can then level the context diagram into children diagrams, each of which will show various functions of the system in greater detail.

Try to keep the number of processes in a single level to  $7 \pm 2$ . This is a manageable number.

Duplicate data stores and external entities (not processes) in order to minimise the number of line crossings.

Develop the diagram with the data flow names turned off (drawing is much quicker in this mode) and then turn flow names back on when you have finished or when you want to print the diagram out.

Use the information dialog box to store low level information. For data flows, this may include a description of the data elements that make up the flow, and for objects a short description of its function.

## Trouble Shooting

### **An error occurs on startup.**

----- Check that the file 'Daffy.help' is in the same folder as the application.

### **Printing causes an error.**

----- This is an error in the application, specifically the printing command does no work properly. So save your diagram before attempting to print.

**Printing in 'best' and 'fast' mode does nothing.**

----- Check that your startup disk has sufficient room for writing to while printing.

**The application won't let you connect two objects.**

----- Chances are you are trying to make an illegal connection. Read up on maintaining the integrity of data flow diagrams.