The Analysis and Design of an Automated Tool to Support Structured Systems Analysis

A thesis presented in partial fulfilment of the
requirements for
the degree of Master of Arts in Computer Science at
Massey University

Joanne Tucker
1988

Volume I

Abstract

Systems analysis is an inherently difficult task. Errors that are introduced in the analysis and design phases become progressively more expensive to fix in the later stages of the system life cycle. Systems analysis and design methodologies attempt to reduce the number of errors introduced into a system model and to detect (and correct) those errors that do occur as early as possible in the system development lifecycle. One such methodology that is widely used in New Zealand is Structured Systems Analysis. Users of Structured Systems Analysis tend to find that the documentation produced using the methodology is easier to read and understand than documentation produced by other currently used methodologies.

This thesis presents the functional specification of MUSSAT, a tool to provide automated support for the Structured Systems Analysis methodology. MUSSAT was designed for a specific group of users. The needs of these users are discussed, together with an introduction to the tools and techniques of Structured Systems Analysis. Existing versions of Structured Systems Analysis are reviewed and a modified form of the methodology, incorporated in MUSSAT, is presented.

A discussion of the tools and techniques used to specify the MUSSAT model are discussed. This is followed by an introduction to the MUSSAT system model. Details of the MUSSAT model are included as a series of technical appendices.

Finally, an overview of the extent to which Structured Systems Analysis is supported by existing Computer Aided Software Engineering (CASE) tools is presented together with a discussion of where MUSSAT fits with these CASE tools.

Acknowledgments

During the past two years there have been many people who have influenced the course and content of the research presented in this thesis. I am grateful to them all for their help.

In particular, I would like to thank my supervisor, June Verner, for painstakingly reviewing each part of this thesis and for her invaluable enthusiasm, encouragement and ideas.

I would also like to thank Professor Graham Tate and Richard Hayward for their constructive criticisms of earlier drafts of the first four chapters of this thesis.

Finally, I would like to thank Stephen Quinn and Jason Cruickshanks for their help in using the Department's Vax 11/750, Unix and Emacs and for helping me understand some of the basics of interactive graphics.

Table of Contents

List of Figures

CHAPTER 1

Introduction

The evolution of a computerised information system begins when the need for the system is recognised. If management is willing to commit resources to the development of the system, various design and construction phases will take place transforming the original idea into an implemented system. Once the system is operational continued financial commitment for maintenance will be required, until eventually, the system reaches the end of its productive life and is phased out.

This is, of course, a simplified picture of the development of a computer system. The feasibility of a proposed system will be reviewed periodically during development and if the perceived benefits of the system are outweighed by the perceived costs the project may be abandoned, even after significant expenditure.

The term 'system life cycle' is commonly used to describe the various phases through which a computerised system evolves. For the development of a system to proceed in a controlled fashion certain activities must take place and specific documentation should be produced during each phase of the life cycle. There have been many definitions of the

system life cycle including those in [KAN84], [DAV83], [WEI84], [POW84] and [DEM78]. Most systems development textbooks present a version of the traditional system life cycle, also called the project or software life cycle. Nevertheless, these life cycle models are similar even though the names and boundaries of the phases may differ.

Some system life cycle models, such as that presented by Boehm [BOE81], incorporate techniques such as prototyping, incremental development and advancemanship. However, these alternate life cycle models still include some of the earlier activities of the traditional system life cycle models. In this thesis, discussion of the system life cycle will refer to the traditional life cycle model in which systems analysis precedes systems design.

One life cycle model [POW84] divides system development into the following five phases:

Phase	Successful completion documented by
INVESTIGATION	
Initial Investigation	
	Initial Investigation Report which, among other activities, identifies the major options and recommends one or two of the options for further consideration in the next phase.
	Oral Report to Management.
Feasibility Study	Feasibility Report which includes: a description of the existing system; and rough cost/benefit analysis estimates and project schedules for each alternative.
	Project Plan
	Working Papers Which serve as an input to the next phase and include: a review of the existing and proposed system; and alternative implementations of the proposed system.

Phase	Successful completion documented by
ANALYSIS AND GENERAL DESIGN	
Systems Analysis	User Specification which includes: a model of the proposed system; and specification of performance, security and control requirements etc.
General Design	New System Design Specification which includes: internal software design, file or data base design, hardware specifications, internal controls, etc.
DETAILED DESIGN AND IMPLEMENTATION	A fully tested and documented system.
INSTALLATION	No new products. Activities include file conversion and personnel transitions.
REVIEW	System Development Recap Report and Post Implementation Review Report.

Weiner and Sincovec [WEI84] provide an alternate description of the outputs that should be produced as a result of the Systems Analysis phase. These components comprise a Software Requirements Document and consists of the following ([WEI84] pp. 17-18):

(1) A description of the functional requirements, with an emphasis on describing the system as it will appear to the user, including a description of the user interface.

(2) A description of the non functional requirements of the new system.

(3) A project plan.

(4) Maintenance information.

(5) An initial user manual.

(6) A glossary of all technical terms.

The outputs of one phase of the lifecycle are used as inputs to the next. Therefore, the successful completion of a phase in the development cycle is, in part, dependent on the quality of the outputs of the previous phase: if the outputs of one phase are deficient or contain errors, then the next phase will not have an accurate description of the current state of the system on which to base successive work. If an error is detected in the system model then the outputs of each phase, from the phase in which the error was introduced to the phase in which the error was detected, must be amended to present a more accurate model of the system. In the worst case deficiencies in the Investigation phase will not be detected until acceptance testing when the users discover that the system developed is not the system that they

wanted.

The earlier that deficiencies are detected in the development of a system the less costly they will be to correct. Boehm ([BOE81], p. 40) states that errors in large system development projects not detected until the maintenance phase are likely to be at least 100 times more expensive to correct than if detected during the requirements phase. To compound the problem, another author [CON82] suggests that during the development of a system a large proportion of all errors (that are later detected) are introduced during the analysis and design phases ([CON82] p. 215).

Systems analysis is an inherently difficult task. Analysts are required to produce complete and unambiguous documentation of a system by working with users who are often unsure exactly what they want the system to do and, in some cases, may not even want a new system. The analysis phase needs to be carried out by experienced computer professionals with expertise in both computer systems and the application area. The specification of the new system produced should fulfill all requirements specified by the users and be detailed enough to serve as an input into the Detailed Design and Implementation phase.

Various systems analysis and design methodologies have been specified in an attempt to reduce the number of errors introduced into a system model and to detect (and correct)

those errors that do occur as early as possible in the system development lifecycle. One such methodology is Structured Systems Analysis (SSA). SSA became popular in the late 1970's and is now widely used, particularly in North America and New Zealand.

1.1. Structured Systems Analysis

The SSA methodology incorporates a set of tools and techniques to describe a logical model of an existing or proposed system. The main components of SSA are:

data flow diagrams (DFDs); and an associated
data dictionary (DD).

The logical system model produced during the analysis phase, using SSA, corresponds to the Software System Model component in Weiner and Sincovec's list of systems analysis products given above. Note that the Software System Model is only one component of several required to fully specify a proposed computer system.

There are two major variants of the SSA methodology: the DeMarco [DEM78] and Gane & Sarson [GAN80][1] SSA methodologies. Although both of these versions of SSA are each based

[1]The text reference throughout this thesis is a republication of: Gane, Chris and Sarson, Trish, Structured Systems Analysis: Tools and Techniques (New York: Improved Systems Technologies Inc., 1977).

on the same principles, they differ in:

physical representation of DFDs;

DFD construction rules;

contents of the DD; and

representation of complex file structures.

The discussion in this thesis assumes a familiarity with the basic SSA tools and their application (as described in [DEM78] and [GAN80]), however, a brief overview of SSA is given in Chapter 2.

Neither the Gane & Sarson nor the DeMarco SSA methodology have proven to be the ideal analysis tool for all situations. SSA, in general, has a number of weaknesses, including:

reliance upon the skill of the analyst to produce a good system model;

lack of consistency and completeness checks;

difficulty in modelling interactive systems, and related to this, the inability to model different processing or response constraints such as monthly report runs as opposed to the response times required for interactive query facilities; and

lack of support in translating the new logical model produced during systems analysis into the new physical model required as the output of systems design.

When considering criticism number three in the above list it is possible to argue that using SSA the analyst should not be concerned with, nor have decided the type of interface

that the proposed system should incorporate at the analysis phase. Hence the inability to show any one particular type of user interface in the logical new model should not be a concern

Nevertheless, in some cases the analyst knows even before the analysis phase begins that the system to be produced must be interactive. In such cases it would be naive and counterproductive to ignore the requirements of an interactive user interface, especially when the incorporation of such an interface is likely to have a significant impact on cost and size estimates (made both at the feasibility and later stages) and the system design itself. Size and cost estimates should be as accurate as possible so that the system alternative selected or even the decision to continue system development is made with the best possible information.

Even in view of the above weaknesses of the methodology, SSA has one main advantage over other design methodologies such as ISAC [LUN81], JSD [JAC83] and Information Engineering [MAR81]: namely, that the specification produced using SSA serves as a good communication tool, especially between the analyst(s) and users. Data flow diagrams are conceptually simple and are similar to other diagrams familiar to many users in the business world. Unlike ISAC, a SSA system specification is relatively concise which means that management of the documentation may be simplified and intended

users of the system are not given copious and daunting amounts of documentation to approve.

Data flow diagrams also highlight the transformation of data, that is, the processes within the system. Users are process oriented: they think in terms of objects and the operations performed on objects, such as editing a document or updating a master file, not in terms of entities, attributes and relationships [TAT86]. Information Engineering takes the latter approach. SSA helps users to identify errors and omissions in systems modelled using DFDs since the model more closely represents the way they view the real world.

Individuals and organisations who use SSA are likely to tailor one of the SSA methodologies to suit the peculiarities of their working environments. Such adaptations can be considered to produce hybrid versions of the SSA methodology.

The research documented in this thesis is concerned with the development of an automated SSA tool. Such software is required to provide computerised assistance in developing and maintaining SSA system models.

A tailored form of the SSA methodology will be required for the proposed automated SSA tool. The user groups of an automated SSA tool are expected to have specific needs that are not explicitly catered for in either of the two standard

SSA methodologies, therefore, a hybrid SSA needed to be defined to suit the requirements of the intended users of the system.

Investigation of automated support for SSA was considered a suitable topic for this thesis because:

> Most of the automated SSA tools currently available are expensive;
>
> Automated SSA tools may be able to offer assistance in other areas of system development in addition to systems analysis; and
>
> An automated SSA tool may be able to provide a means of overcoming some of the deficiencies in the SSA methodology.

1.2. Users of Structured Systems Analysis

Although SSA has been in use for over a decade, automated tools supporting the methodology have only recently become commercially available. It is not known how useful systems analysts find such automated tools: whether they are useful aids in developing the logical model of a system, or only useful in documenting and checking a system model developed using pencil and paper.

In either case, even an experienced analyst usually requires several iterations to produce the final version of a DFD. Drawing and updating DFDs manually is tedious and automated tools are expected to, at least, help reduce the time consuming nature of DFD maintenance and development. Time sav-

ings may not be as readily apparent in the initial specification of a DFD, where the time taken to specify a DFD using an automated SSA tool and drawing a DFD by hand may be similar. The real time savings are expected to be most evident in DFD maintenance where, using an automated SSA tool, the user does not need to redraw the entire DFD in order to produce an updated copy.

Post-implementation changes to a system should be reflected in the system documentation. The maintenance of system documentation can be error prone and time consuming and is often poorly done. An automated SSA tool may be a useful aid in maintaining system models developed or simply documented using SSA.

Automated SSA tools may also encourage more creative experimentation with system designs if changes are easy to make and control. Hence, alternative system designs may be able to be produced quicker and at less cost. In addition, alternate system designs may be more complete using an automated SSA tool since less effort may be required to develop these alternate designs. Designs that are more complete or thorough (even at a very high level) may help users to understand their needs and the proposed system better and hence help create a system meeting their requirements as closely as possible.

Existing automated SSA tools may possibly incorporate version(s) of SSA that overcome some of the deficiencies of the original methodology.

An automated SSA tool is required or would be useful in a number of fields in which research is currently being done. Within the Massey University Department of Computer Science an automated SSA tool may be useful in the following areas:

(1) The Department offers some courses that require students to produce SSA system models. An automated SSA tool would be a useful teaching aid for such courses.

(2) Two groups of researchers within the Department require software that allows users to specify SSA system models using a graphical interface.

Many of the existing automated SSA tools are not available in New Zealand, and those that are available are expensive and often unable to be evaluated thoroughly enough, either by hands-on experience or detailed documentation. Without the tools being available for evaluation it is difficult to decide whether they are cost-effective. In addition, unless potential users are able to gain hands-on experience of available tools, it is difficult to specify exactly what constitutes a useful automated SSA tool. Without a thorough analysis of what is required of an automated SSA tool by potential users, short-term exposure to such tools would be an inadequate foundation on which to base any decision about

the usefulness or otherwise of an automated SSA tool.

There are a number of automated SSA tools and aids commercially available.[2]

These include:

Excelerator	(Index Technologies Corp.)
Teamwork/SA	(Cadre Technologies Inc.)
PCSA	(StructSoft Inc.)
DesignAid	(Nastec Corp.)
Anatool	(Abvent)
ProkitAnalyst,	(McDonnell Douglas Automation Co.)
Blues	(De Landgraff Consultancy).

Potential users of an automated SSA tool are expected to require specific features of such software. This thesis identifies the features that one group of users would require in such an automated SSA tool so that the tool will be useful to them. These needs can also be used as criteria to evaluate existing automated SSA tools and in determining whether the ideal automated SSA tool exists. Given the current technology, it may be that an ideal tool at a reasonable price is not commercially available.

---

[2]The names and addresses of the suppliers of the named automated SSA tools are listed in Appendix I.

1.3. Research Goals and Methodology

It was the aim of the research documented in this thesis to design an automated SSA tool that allows systems analysts to develop and maintain system models using SSA. The design goal was broken into the following sub-steps:

(1) Identify the needs of potential users of an automated SSA tool.

(2) Investigate the SSA methodology in light of the requirements of the users.

(3) Present the design of an automated SSA tool that satisfies the needs of the selected user(s) and incorporates an appropriate form of the SSA methodology.

(4) Compare features of commercially available automated SSA tools with the system designed presented in this thesis.

It may have been more appropriate to examine existing automated SSA tools before steps 1 to 3, however, at the time this research was undertaken it was not known which, if any, automated tools would be available for evaluation. Permission was granted by the Reserve Bank of New Zealand (after steps 1 and 2 were under way) to use their copy of Excelerator for evaluation. Later in the year a demonstration version of PCSA was ordered from the U.S.A. but was not received until December, 1986. In addition, the proposed

automated SSA tool was designed before using any existing tools so that the design was not biased by the characteristics (and perhaps limitations) of existing tools. Hence, an evaluation of existing SSA tools was not carried out until steps 1 to 3 were completed.

1.4. Thesis Organisation

The body of this thesis is divided into six chapters and an nine appendices:

Chapter 2 presents an overview of the SSA features common to both the Gane & Sarson and DeMarco versions of the methodology.

Chapter 3 describes the potential user groups of an automated SSA tool and outlines their general needs. A discussion of the reasons for selecting a subset of all identified users is also presented in Chapter 3.

Chapter 4 includes a discussion of the differences between existing SSA methodologies and introduces the characteristics of the hybrid SSA methodology suggested in section 1.1.

A discussion of the major design decisions and techniques used in the development of MUSSAT: Massey University Structured Systems Analysis Tool, is given in Chapter 5. Chapter 5 also includes an overview of the user interface of MUSSAT. Details of the MUSSAT design are included in a series of technical appendices, each of which is introduced in Chapter

5.

Chapter 6 gives an overview of CASE Tools and compares the goals of MUSSAT with the goals of existing CASE tools. The user interfaces of two automated SSA tools are discussed and compared with the user interface of MUSSAT.

CHAPTER 2

SSA: A Brief Overview

Although the Gane & Sarson and DeMarco SSA methodologies differ in a several ways, the basic SSA concepts are embodied in both approaches. The SSA characteristics discussed in this chapter represent a general form of SSA, that is, the aspects of SSA common to both the DeMarco and Gane & Sarson versions of the methodology.

Figures 2.1 to 2.7 model part of a simple order processing system in which a company receives orders for parts from its customers and then sends the customer the parts ordered, an invoice and an out of stock notice if some parts on the order were unable to be supplied. This example is based on a model similar to that presented in [DOC86] and is used in this chapter to describe the tools and techniques of SSA including: data flow diagrams, the data dictionary, and the logical system model.

2.1. Data Flow Diagrams

Data flow diagrams (DFDs) show all data that may exist during the life of a system and all data transformations that may occur. Figures 2.1, 2.2 and 2.3 are DFDs describing parts of the order processing example introduced above.

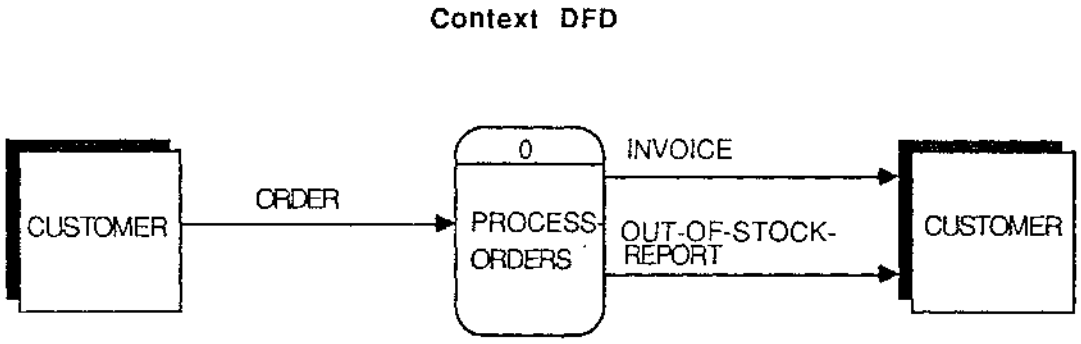


Figure 2.1 - Context DFD for the order processing example.

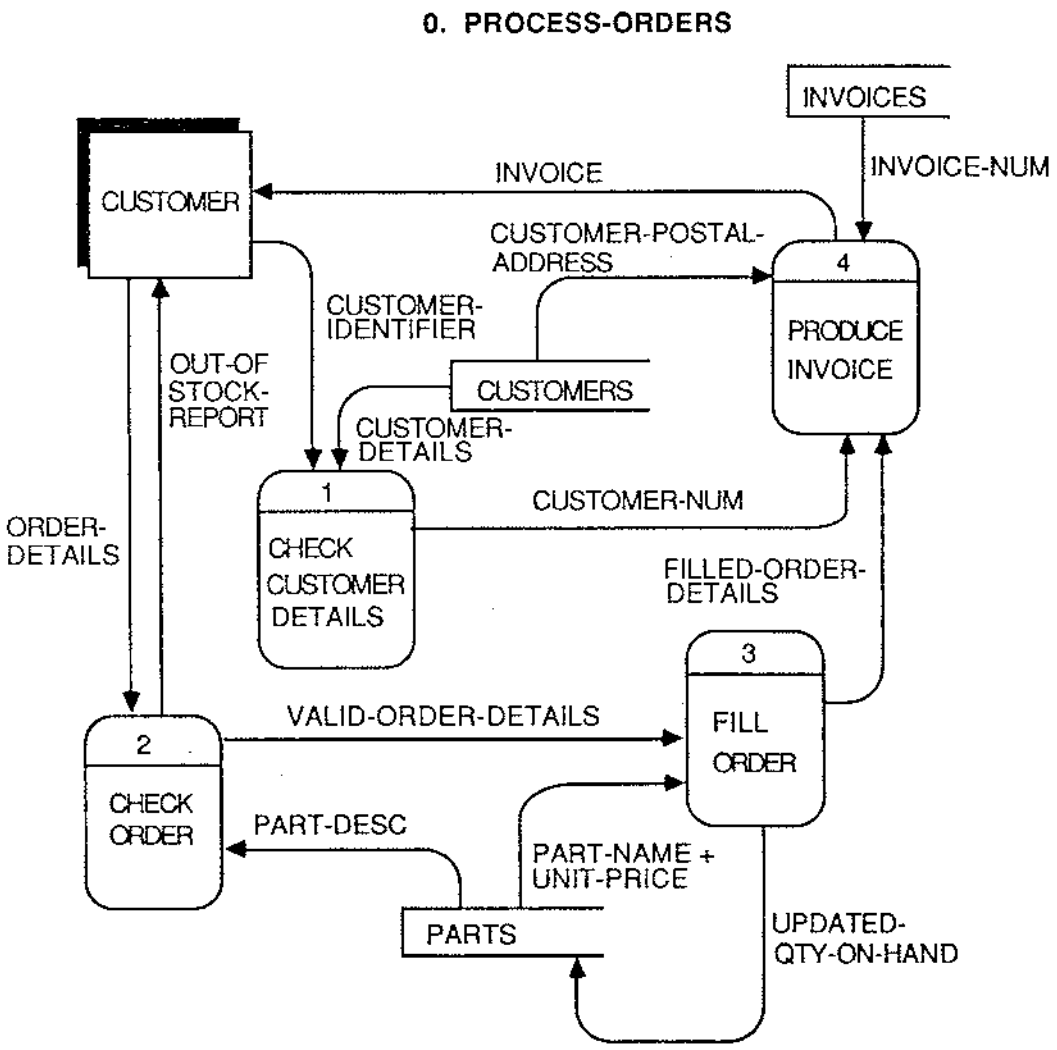


Figure 2.2 - Level 0 DFD for the order processing example.

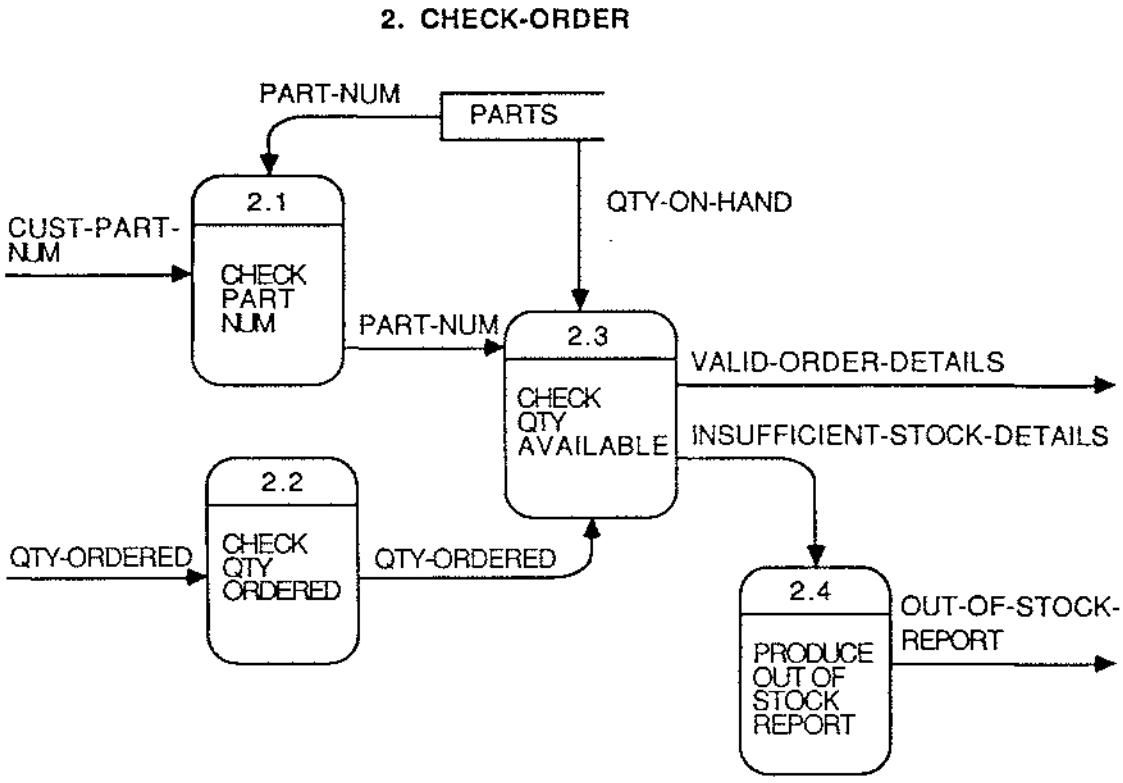


Figure 2.3 - DFD of Process 2 CHECK-ORDER.

Individual DFDs are (ideally) developed from a data precedence viewpoint: by specifying the data inputs required by a process to produce the desired data outputs and working backwards until the net data inputs to the DFD are identified.

DFDs are composed of the following four types of DFD elements:

data flows;

processes;

data stores (files or data bases) [1] ; and

external entities (data source and sinks).

Each type of DFD element is discussed further in the following sub-sections.

2.1.1. Data Flows

In figures 2.1, 2.2 and 2.3 each of the lines ending in an arrow head is a data flow.

Data flows act as pipelines for data flowing through the system. Each data flow should be named with a short descriptive identifier and should only contain the data required by the destination process, external entity or data store.

2.1.2. Processes

Each of the upright, rounded rectangle shapes in figures 2.1, 2.2 and 2.3 is a process.

Processes use the contents of the incoming data flows to produce outward data flows. Not all data flows shown as entering a process are necessarily required for the activation of the process, nor are all outward data flows necessarily produced as a result of activating the process, a DFD simply illustrates all potential inputs and outputs to and from a process that may exist during the life of the system.

---

[1]The terms in parenthesis are the DeMarco equivalent of terms where Gane & Sarson and DeMarco use different names for the same SSA component. In this thesis the Gane & Sarson terminology will be used.

Processes are named and numbered and may be described further using another DFD. For example, process 2 (CHECK-ORDER) in the DFD of figure 2.2 has been expanded into the DFD in figure 2.3.

If a process is exploded into a lower level DFD (also called a child DFD), then net data flows into and out of the child DFD must balance with the parent process. Even so, child DFDs may contain local data stores and show parallel decomposition of data flows.

Processes within a DFD are numbered left to right across the page with each process bearing the number of the DFD it resides in as well as a unique suffix number. A child DFD inherits the same number as its parent process.

Figure 2.1 is the Context Diagram for the order processing system introduced earlier. A context diagram delimits the scope of the system being modelled and shows only the net data flows in to and out of the system. Figure 2.2, the level 0 DFD, is the expansion of the Context Diagram. Note that in the level 0 DFD processes are not prefixed with 0 but are numbered from 1 to n (n being the number of processes in the DFD).

Low level processes, also called functional primitives or leaf-level processes, should be described in the data dictionary in one of three textual formats: Structured English, Decision Tables or Decision Trees.

2.1.3. Data Stores

Data stores represent logical collections of data at rest. A data store can represent anything from a wastepaper basket to a large database. The open ended rectangles in figures 2.1, 2.2 and 2.3 labelled CUSTOMERS, PARTS and INVOICES are the data stores used in the order processing example.

Data is retrieved from and placed in to a data store via data flows. Data flows pointing into a data store represent a net flow of data into the data store; data flows leaving a data store represent a net flow out of the data store. Double headed data flows connected to a data store represent a net flow in to and out of the data store, that is, data is updated and retrieved from the data store.

All net flows to and from a data flow are shown in the first DFD in which the data store appears.

2.1.4. External Entities

The box labelled CUSTOMER in figures 2.1 and 2.2 is an external entity: a group of people, an organisation or another system that interacts with, but whose detailed functioning is outside the scope of the system being modelled.

2.2. Data Dictionary

A data dictionary contains a definition of each data flow, data store, external entity, leaf-level process and data element appearing in a DFD or set of DFDs. All definitions in the DD are filed alphabetically using the names of the defined items as the key. Each definition stored in the DD is called a DD entry.

2.2.1. Data Elements

In addition to the DFD elements shown in DFDs, data elements should also be defined in the DD. A data element is a piece of data that is logically indivisible. All data flows and data stores are ultimately composed of data elements. For example, the data flow CUSTOMER-POSTAL-ADDRESS in figure 2.2 could be described (simplisticly) as consisting of:

CUSTOMER-NAME;
POST-OFFICE-BOX (optional);
STREET-ADDRESS;
CITY; and
COUNTRY.

It is unlikely that CITY could usefully be broken down any further (ignoring suburbs, postal codes, and other anomalies). Therefore, CITY is a data element since it cannot be logically decomposed into smaller elements. On the other hand, STREET-ADDRESS could be further described as

consisting of STREET-NUMBER and STREET-NAME, each of which may be described as a data element (if it is not logical to decompose either item any further).

2.2.2. Data Items

When discussing the use of data elements in SSA it is useful to use one term to describe all types of structures that may be built using data elements. In this thesis the term 'data item' will be used to refer to any named grouping of data that forms a logical unit, that is, a data element, data flow or data store. Data flows and data stores consist of individual data elements which may be grouped into data items.

2.2.3. Data Dictionary Languages

All DD entries contain a description of the contents of a processes or data item in a specific definition language: a process description language for processes or a BNF-type language for a data item.

Process description languages are discussed in the following sub-section to complete the overview of the SSA methodology, however, the DD of the proposed SSA tool will not directly support or enforce any of the process description languages discussed below. It is beyond the scope of this thesis to develop a DD capable of supporting these languages.

2.2.3.1. Process Description Languages

SSA supports three languages for defining processes:

Structured English

Decision Tables

Decision Trees.

The choice of which language to use for a particular process lies with the analyst.

Structured English

Structured English is a pseudo-code type language that uses three logical constructs: sequence, decision and repetition, as well as the names of data items to describe the logic of leaf-level processes. Indenting is used extensively to clarify the structure of the definition. Structured English is probably the most flexible of the three languages as it can be made more natural-language-like to suit users who are uncomfortable with the terseness of Structured English in its purest form. Figure 2.4 is a Structured English definition of process 3 (FILL ORDER) in the DFD of figure 2.2.

1. For each VALID-ORDER-LINE:
    1.1 Access PARTS, using PART-NUM as key.
    1.2 Decrement QTY-ON-HAND by QTY-ORDERED.
    1.3 Set SUPPLIED-PART-NAME = PART-NAME.
    1.4 Set SUPPLIED-UNIT-PRICE = UNIT-PRICE.

Figure 2.4 - Structured English definition of Process 3 FILL ORDER.

Decision Tables

Decision Tables express the logic of a process in a matrix showing the conditions and rules that a particular action depends upon. DeMarco ([DEM78] p. 216) suggests that Decision Tables are the appropriate process description language to use when the action(s) selected depends on several of conditions.

Decision Trees

Decision Trees are the third method of describing leaf-level processes. Decision Tree are simply graphical representations of a Decision Tables and are appropriate for describing the same types of processes for which Decision Trees are used.

In the order processing example introduced earlier in this chapter, a process suitable for description using a Decision Table or Decision Tree would be a process that calculates how to send a filled order back to the customer: either

airmail or standard post (A or S), express or normal delivery (E or N). Assume that such a process is a sub-process of process 4 (PRODUCE INVOICE) in the DFD of figure 2.2.

The delivery service chosen is based on a combination of the following conditions:

(1) the value of the order:

under $200 (U)
$200 and over (T);

(2) the destination of the parcel:

inland (I)
overseas (O); and

(3) the weight of the parcel:

less that 5 kilogrammes (L)
5 kilogrammes or greater (G).

Figures 2.5 and 2.6 show equivalent Decision Table and Decision Tree representations (respectively) of the various ways a parcel may be sent and the conditions under which each method of delivery is selected.

Value of order	U	U	U	U	T	T	T	T
Destination	I	O	I	O	I	O	I	O
Weight	L	L	G	G	L	L	G	G
Delivery method	A	S	S	S	S	A	S	A
Service	N	E	E	N	E	E	E	N

Figure 2.5 - Decision Table representation of freight method policy.

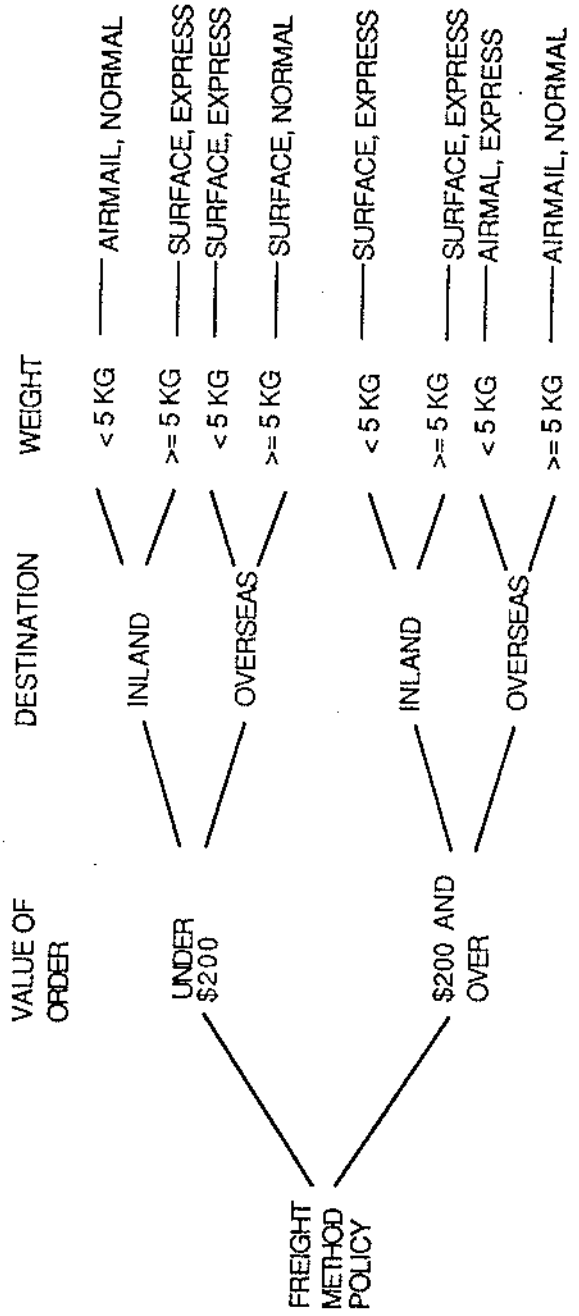


Figure 2.6 - Decision Tree representation of freight method policy.

For comparison, figure 2.7 shows the same logic expressed in Structured English.

```
IF VALUE-OF-ORDER < $200
    THEN
        IF DESTINATION = INLAND
            THEN
                IF WEIGHT < 5 KG
                    THEN
                        SEND AIRMAIL, NORMAL
                ELSE (* WEIGHT ≥ 5 KG *)
                    SEND SURFACE, EXPRESS
        ELSE (* DESTINATION = OVERSEAS *)
            IF WEIGHT < 5 KG
                THEN
                    SEND SURFACE, EXPRESS
            ELSE (* WEIGHT ≥ 5 KG *)
                SEND SURFACE, NORMAL
    ELSE (* VALUE-OF-ORDER ≥ $200 *)
        IF DESTINATION = INLAND
            THEN
                SEND SURFACE, EXPRESS
        ELSE (* DESTINATION = OVERSEAS *)
            IF WEIGHT < 5 KG
                THEN
                    SEND AIRMAIL, EXPRESS
            ELSE (* WEIGHT ≥ 5 KG *)
                SEND AIRMAIL, NORMAL.
```

Figure 2.7 - Structured English representation of freight method policy.

2.2.3.2. Data Item Description Languages

The composition of data flows and data stores are defined using a language consisting of several relational operators and the DD names of data items. The symbols used for the relational operators are different in the Gane & Sarson and DeMarco versions of SSA. Both versions are discussed in

Chapter 3.

Data elements are not described using the relational operators. Instead, discrete or continuous ranges of values and their associated meanings are listed in the DD entry.

Definition of a data item within the DD stops when the analyst believes that the meaning is self-evident; a data item need not necessarily be defined down to data element level if the analyst believes that the meaning of the data item is apparent.

For completeness, the data items used in the order processing example are defined below (using DeMarco's notation):

```
VALID-ORDER-LINE = PART-NUM + QTY-ORDERED

VALID-ORDER-DETAILS = 1{VALID-ORDER-LINE}40

SUPPLIED-ITEM = PART-NAME +
                QTY-SUPPLIED +
                SUPPLIED-UNIT-PRICE

FILLED-ORDER-DETAILS = 1{SUPPLIED-ITEM}40

ORDER-LINE = CUSTOMER-PART-NUM + QTY-ORDERED

ORDER-DETAILS = 1{ORDER-LINE}40

CUSTOMER-IDENTIFIER = CUSTOMER-NAME +
                      (CUSTOMER-NUM)

ORDER = CUSTOMER-IDENTIFIER +
        ORDER-DETAILS

PART-DESC = PART-NUM + QTY-ON-HAND

PART-DETAILS = PART-DESC + UNIT-PRICE

PARTS = 1{PART-DETAILS}N

STREET-ADDRESS = STREET-NUMBER +
```

```
                    STREET-NAME

CUSTOMER-ADDRESS = (POST-OFFICE-BOX) +
                    STREET-ADDRESS +
                    CITY +
                    COUNTRY

CUSTOMER-POSTAL-ADDRESS = CUSTOMER-NAME +
                          CUSTOMER-ADDRESS

CUSTOMER-DETAILS = CUSTOMER-NUM +
                   CUSTOMER-NAME +
                   CUSTOMER-POSTAL-ADDRESS

CUSTOMERS = 1{CUSTOMER-DETAILS}200
```

The data item description language used above is described in greater detail in Chapter 4.

2.3. System Models

A logical system model developed using SSA is documented in a set of DFDs and an associated DD. The set of DFDs in a SSA system model will be referred to in this thesis as a hierarchy of DFDs.

A hierarchy of DFDs is a set of DFDs with one top level DFD; every other DFD in the set is a child DFD of a process within another DFD in the set. A hierarchy of DFDs is normally developed by functional decomposition: processes are expanded into lower level DFDs to show the system in more detail. For example, the DFDs in figures 2.1, 2.2 and 2.3 form part of a hierarchy of DFDs: the DFD in figure 2.1 is the top level DFD, figure 2.2 is an explosion of PROCESS ORDERS in figure 2.1, and figure 2.3 is an explosion of pro-

cess 2, CHECK ORDER, in figure 2.2.

In this thesis each logical system model, that is, a hierarchy of DFDs and the associated DD will be referred to as a SSA project.

CHAPTER 3

The Users and Their Needs

To design an automated SSA tool that will provide useful functions for its users, who the users are, and exactly what their needs will be, must be clearly defined. This section includes a discussion of:

(1) potential user groups of an automated SSA tool;

(2) the activities carried out by each potential user group;

(3) the activities that an automated SSA tool should support for each user group;

(4) possible conflicting needs that may be evident between the user groups;

(5) selection of the user group(s) for which the automated tool is designed; and

(6) the general features required in an automated SSA tool.

3.1. The Users

Within the Department of Computer Science at Massey University three groups of people have been identified as potential users of an automated SSA tool. For the purposes of

this thesis these user groups will be referred to as:

(1) Systems Analysts (Analysts);

(2) Software Sizing Personnel (SSP); and

(3) Executable DFD (EDFD) users.

Systems Analysts (SSA)

Systems analysts are responsible for developing logical models of existing and proposed systems. An analyst produces these models by working closely with the end-users of the existing and proposed systems. The logical model of the proposed system is used by systems designers as a blueprint for systems design activities. Hence, a systems analyst acts as an intermediary between users and systems development personnel.

Within the Computer Science Department at Massey University, this group of users is represented by the students enrolled in courses that teach and use SSA. These students are expected to use the proposed system to learn about SSA while they develop system designs. An automated SSA tool will primarily be a teaching aid for this user group, but teaching staff and analyst/programmers employed by the Department are also expected to find the tool useful when undertaking analysis work.

1. For each VALID-ORDER-LINE:
    1.1 Access PARTS, using PART-NUM as key.
    1.2 Decrement QTY-ON-HAND by QTY-ORDERED.
    1.3 Set SUPPLIED-PART-NAME = PART-NAME.
    1.4 Set SUPPLIED-UNIT-PRICE = UNIT-PRICE.

Figure 2.4 - Structured English definition of Process 3 FILL ORDER.

Decision Tables

Decision Tables express the logic of a process in a matrix showing the conditions and rules that a particular action depends upon. DeMarco ([DEM78] p. 216) suggests that Decision Tables are the appropriate process description language to use when the action(s) selected depends on several of conditions.

Decision Trees

Decision Trees are the third method of describing leaf-level processes. Decision Tree are simply graphical representations of a Decision Tables and are appropriate for describing the same types of processes for which Decision Trees are used.

In the order processing example introduced earlier in this chapter, a process suitable for description using a Decision Table or Decision Tree would be a process that calculates how to send a filled order back to the customer: either

airmail or standard post (A or S), express or normal delivery (E or N). Assume that such a process is a sub-process of process 4 (PRODUCE INVOICE) in the DFD of figure 2.2.

The delivery service chosen is based on a combination of the following conditions:

(1) the value of the order:

under $200 (U)
$200 and over (T);

(2) the destination of the parcel:

inland (I)
overseas (O); and

(3) the weight of the parcel:

less that 5 kilogrammes (L)
5 kilogrammes or greater (G).

Figures 2.5 and 2.6 show equivalent Decision Table and Decision Tree representations (respectively) of the various ways a parcel may be sent and the conditions under which each method of delivery is selected.

Value of order	U	U	U	U	T	T	T	T
Destination	I	O	I	O	I	O	I	O
Weight	L	L	G	G	L	L	G	G
Delivery method	A	S	S	S	S	A	S	A
Service	N	E	E	N	E	E	E	N

Figure 2.5 - Decision Table representation of freight method policy.

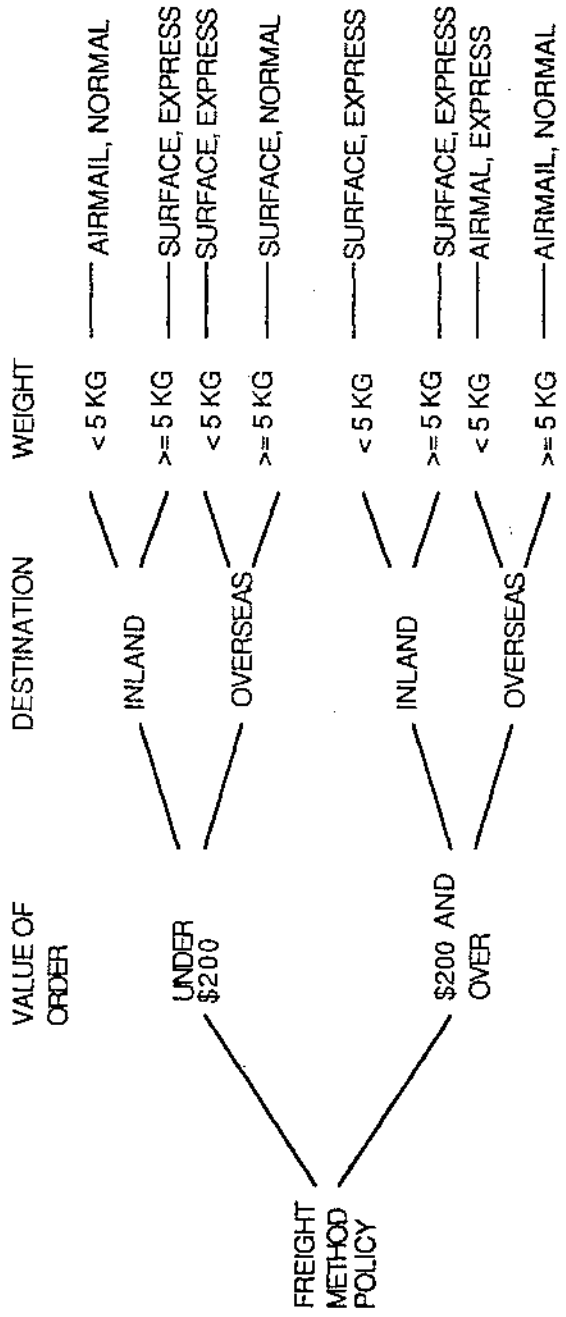


Figure 2.6 - Decision Tree representation of freight method policy.

For comparison, figure 2.7 shows the same logic expressed in Structured English.

```
IF VALUE-OF-ORDER < $200
    THEN
        IF DESTINATION = INLAND
            THEN
                IF WEIGHT < 5 KG
                    THEN
                        SEND AIRMAIL, NORMAL
                ELSE (* WEIGHT ≥ 5 KG *)
                    SEND SURFACE, EXPRESS
        ELSE (* DESTINATION = OVERSEAS *)
            IF WEIGHT < 5 KG
                THEN
                    SEND SURFACE, EXPRESS
            ELSE (* WEIGHT ≥ 5 KG *)
                SEND SURFACE, NORMAL
    ELSE (* VALUE-OF-ORDER ≥ $200 *)
        IF DESTINATION = INLAND
            THEN
                SEND SURFACE, EXPRESS
        ELSE (* DESTINATION = OVERSEAS *)
            IF WEIGHT < 5 KG
                THEN
                    SEND AIRMAIL, EXPRESS
            ELSE (* WEIGHT ≥ 5 KG *)
                SEND AIRMAIL, NORMAL.
```

Figure 2.7 - Structured English representation of freight method policy.

2.2.3.2. Data Item Description Languages

The composition of data flows and data stores are defined using a language consisting of several relational operators and the DD names of data items. The symbols used for the relational operators are different in the Gane & Sarson and DeMarco versions of SSA. Both versions are discussed in

Chapter 3.

Data elements are not described using the relational operators. Instead, discrete or continuous ranges of values and their associated meanings are listed in the DD entry.

Definition of a data item within the DD stops when the analyst believes that the meaning is self-evident; a data item need not necessarily be defined down to data element level if the analyst believes that the meaning of the data item is apparent.

For completeness, the data items used in the order processing example are defined below (using DeMarco's notation):

```
VALID-ORDER-LINE = PART-NUM + QTY-ORDERED

VALID-ORDER-DETAILS = 1{VALID-ORDER-LINE}40

SUPPLIED-ITEM = PART-NAME +
                QTY-SUPPLIED +
                SUPPLIED-UNIT-PRICE

FILLED-ORDER-DETAILS = 1{SUPPLIED-ITEM}40

ORDER-LINE = CUSTOMER-PART-NUM + QTY-ORDERED

ORDER-DETAILS = 1{ORDER-LINE}40

CUSTOMER-IDENTIFIER = CUSTOMER-NAME +
                      (CUSTOMER-NUM)

ORDER = CUSTOMER-IDENTIFIER +
        ORDER-DETAILS

PART-DESC = PART-NUM + QTY-ON-HAND

PART-DETAILS = PART-DESC + UNIT-PRICE

PARTS = 1{PART-DETAILS}N

STREET-ADDRESS = STREET-NUMBER +
```

```
                    STREET-NAME

CUSTOMER-ADDRESS = (POST-OFFICE-BOX) +
                    STREET-ADDRESS +
                    CITY +
                    COUNTRY

CUSTOMER-POSTAL-ADDRESS = CUSTOMER-NAME +
                          CUSTOMER-ADDRESS

CUSTOMER-DETAILS = CUSTOMER-NUM +
                   CUSTOMER-NAME +
                   CUSTOMER-POSTAL-ADDRESS

CUSTOMERS = 1{CUSTOMER-DETAILS}200
```

The data item description language used above is described in greater detail in Chapter 4.

2.3. System Models

A logical system model developed using SSA is documented in a set of DFDs and an associated DD. The set of DFDs in a SSA system model will be referred to in this thesis as a hierarchy of DFDs.

A hierarchy of DFDs is a set of DFDs with one top level DFD; every other DFD in the set is a child DFD of a process within another DFD in the set. A hierarchy of DFDs is normally developed by functional decomposition: processes are expanded into lower level DFDs to show the system in more detail. For example, the DFDs in figures 2.1, 2.2 and 2.3 form part of a hierarchy of DFDs: the DFD in figure 2.1 is the top level DFD, figure 2.2 is an explosion of PROCESS ORDERS in figure 2.1, and figure 2.3 is an explosion of pro-

cess 2, CHECK ORDER, in figure 2.2.

In this thesis each logical system model, that is, a hierarchy of DFDs and the associated DD will be referred to as a SSA project.

CHAPTER 3

The Users and Their Needs

To design an automated SSA tool that will provide useful functions for its users, who the users are, and exactly what their needs will be, must be clearly defined. This section includes a discussion of:

(1) potential user groups of an automated SSA tool;

(2) the activities carried out by each potential user group;

(3) the activities that an automated SSA tool should support for each user group;

(4) possible conflicting needs that may be evident between the user groups;

(5) selection of the user group(s) for which the automated tool is designed; and

(6) the general features required in an automated SSA tool.

3.1. The Users

Within the Department of Computer Science at Massey University three groups of people have been identified as potential users of an automated SSA tool. For the purposes of

this thesis these user groups will be referred to as:

(1) Systems Analysts (Analysts);

(2) Software Sizing Personnel (SSP); and

(3) Executable DFD (EDFD) users.

Systems Analysts (SSA)

Systems analysts are responsible for developing logical models of existing and proposed systems. An analyst produces these models by working closely with the end-users of the existing and proposed systems. The logical model of the proposed system is used by systems designers as a blueprint for systems design activities. Hence, a systems analyst acts as an intermediary between users and systems development personnel.

Within the Computer Science Department at Massey University, this group of users is represented by the students enrolled in courses that teach and use SSA. These students are expected to use the proposed system to learn about SSA while they develop system designs. An automated SSA tool will primarily be a teaching aid for this user group, but teaching staff and analyst/programmers employed by the Department are also expected to find the tool useful when undertaking analysis work.

SSA is currently a pencil and paper methodology involving iterative refinement of text and diagrams, hence, maintenance and consistency checking of the SSA documentation can be tedious and error prone. Therefore, one of the aims of an automated SSA tool for Analysts is to incorporate as many of the clerical aspects of the SSA methodology as possible in the proposed system. This automation should include the development of DFDs and the DD. The automated SSA tool must provide assistance in maintaining (and checking) the consistency and completeness of DFDs and DD entries.

The DD should support Structured English for process definitions and a simple BNF-type language for all other types of DD entries, except for external entity definitions.

As well as the ability to develop hierarchical sets of DFDs and a DD, the Analysts also require the proposed system to be able to check the consistency and completeness of individual DFDs, sets of DFDs, and the DD, all under control of the user. These checks will help the user of the system to identify DFDs or DD entries that require further refinement. The user must be allowed to leave DFDs and DD entries in an inconsistent or incomplete state until that portion of the system is defined further. A minimum amount of compulsory information should be required for the specification of DD

entries and DFDs.

Software Sizing Personnel (SSP)

During the development of a system, size and cost estimations need to be made by SSP to manage the project and to provide empirical data for future use. People involved in software sizing and costing activities may, of course, also assume other roles during the development of a system. Even so, when acting in the role of estimators the major use of an automated SSA tool would be to provide information necessary to produce sizing estimations for any set of DFDs determined to be both consistent and complete.

SSP work closely with systems analysts; the two functions of software size estimation and system specification interlace. If a cost estimate made for a system model is considered by the users to be too high then the Analysts may have to redesign the system. Iterations involving modification of the system model and subsequent re-estimation of the size and cost of the system may be required before a system acceptable to the users (in that the perceived benefits outweigh perceived costs) is specified. Iterations of estimating and modifying a system model may take place before an acceptable system is specified. In a sense, the role of SSP can be seen as a specialised part of the design process, although size/cost estimations will continue

to be made in later stages of system development to monitor the project.

SSP need to be able to retrieve information derived from the set of DFDs (and the DD) that represents the current state of the system being developed. This information must have been previously specified by systems analysts. Therefore, it would be useful to combine the functions required by the Analyst (to specify systems) with those of the SSP (to retrieve sizing information) in the same tool since the two functions are so closely related.

The model to predict system size from specifications is currently the subject of research. The addition of a small number of extra fields in some DD entries together with the information from the DFDs and the standard DD, should provide most of the information required for the sizing model.

Executable DFD (EDFD) users

The high level data flow environment currently under development at Massey University requires a means of allowing DFDs and DD entries to be defined via a DFD definition interface [DOC86]. Unlike Analysts and SSP, the processes of interest in an EDFD environment are those at the leaf level; higher level process descriptions are seen mainly as documentation and development

aids [TAT85].

EDFD users may not necessarily use conventional leveled DFDs to develop system definition(s).

The role of an automated SSA tool in an EDFD environment is less of a development tool (as for the Analyst/SSP users) and more as a specification tool. Therefore, the activities of the EDFD Users can be seen as occurring after the systems analysis phase; the main function of an automated SSA tool would be to allow the specification of a fully developed system model. Nevertheless, the type of consistency and completeness checks required by Analysts are still likely to be useful to EDFD users, although the definition of what a complete or consistent set of DFDs is may differ. In principle, planning and implementing several consistency criteria in the proposed system is possible. Even so, the expected complexity of an automated SSA tool with more than one set of consistency criterion is considered to be beyond the scope of this project.

The executable DD required in an EDFD environment is considerably more structured than the standard SSA DD. The dictionary language used in an earlier prototype of the EDFD environment allowed processes to be defined using an extended BNF language under the direction of a syntax-directed editor. Research currently in progress

includes the development of an algebra for the definition of DFDs [DOC86].

Execution of the leaf level DFDs is expected to be illustrated by animating defined DFDs. Therefore, the major aims of a DFD development tool for EDFD users are:

(a) To allow the specification of the lowest level of a set of DFDs in which DD entries and DFDs define a complete and consistent system capable of being executed by the EDFD environment. These specifications should be in a structured language which, at the time of writing, is not fully defined.

(b) To provide a vehicle by which the execution of any given DFD(s) may be illustrated.

Summary

All three user groups require a tool that allows a system to be described as a hierarchy of DFDs and a DD. There are several conflicting user requirements:

(1) Some of the DFD construction rules necessary for the specification of an EDFD environment are expected to conflict with the flexibility needs of the Analysts.

As a design and analysis tool, SSA offers considerable freedom in what the user can or cannot represent in a

DFD; many of the so-called rules of the methodology are more guidelines which can be tailored to suit the installation or individual. The Analysts require a version of SSA that allows as much design freedom as possible while ensuring that the SSA rules are being correctly applied. In contrast, a SSA methodology for the EDFD environment would need to be tailored to ensure that the resulting leaf level DFDs and associated DD entries defined by a user are precise enough so that the specified system may be executed. It appears likely that a SSA methodology suited to Analysts may not be constrained enough for EDFD users. Conversely, a SSA methodology tailored to the needs of EDFD users will be too restrictive for the Analysts.

(2) The type of grammar required in the executable DD required by EDFD users is not suited to the Analysts. Part of the advantage of using Structured English for process definitions is the ability to extend the basic vocabulary so that the process definitions can be made using a language familiar to the users. This is particularly important for users who are uncomfortable with computer-like languages ([DEM78] p. 204). A highly structured grammar would not allow the flexibility that the Analysts require in a DD.

(3) The development of an executable DD grammar for use in the EDFD environment is outside the scope of this

project. Therefore, some intermediate form of DD would be required for use in the automated SSA tool until the executable DD was available for use. Difficulties in designing an automated SSA tool and later adapting the tool to interface with an executable DD may arise.

(4) The ability to animate DFDs may force a system design not suited to the other user groups.

These conflicts could be resolved by developing a system that allowed different modes of DFD development, for example, one mode for EDFD users where DFD construction rules are quite constrained and another mode for Analyst/SSP users where DFD construction rules are more flexible. Even if the animation of DFDs (which is considered to be a significant exercise in itself) is not included in this project, the development of one automated SSA tool to satisfy all the needs of all the intended user groups is still beyond the scope of this project because of the complexity of such a system. Therefore, an automated SSA tool for Analysts and SSP has been chosen as the subject of this project. The reasons for this decision are that:

(1) Meeting the needs of the Analyst user group satisfies the common needs of all three user groups.

(2) Such a tool will be potentially useful to a greater number of users.

(3) An automated SSA tool for Analysts and SSP users has a more clearly defined interface than an automated SSA tool for a EDFD environment.

In the rest of this thesis, unless stated otherwise, any reference to the users or target user group refers to SSP and Analyst users.

3.2. General Requirements of an Automated SSA Tool

This chapter outlines the general features that the Analyst and SSP user groups require in an automated SSA tool. These requirements form the criteria that will be used to evaluate existing automated SSA tools and form the basis on which the proposed automated SSA tool will be designed.

Major decisions regarding the more detailed aspects of the proposed system will need to be made, either by:

(1) consultation with the users, or when the decision is one the users are unlikely to be able to make (or should not have to make), then

(2) the system designer will make the decision.

In the latter case, the designer should consider all options available and choose the option that not only enhances the usefulness of the product but at the same time complies with the SSA rules. The decision (and all options) must be documented and passed to a user representative for approval.

The main goals of the proposed system are:

> To provide automated assistance in creating and manipulating SSA system designs. The system should allow a user to interactively draw DFDs and maintain an associated DD using a graphical interface. Printed copies of DFDs and DD entries

should be available to the user.

The system should be able to carry out checks for:

unbalanced DFDs;

DFD elements that do not have corresponding DD entries; and

DD entry components that may need further definition.

3.2.1. System Features

(1) The specific hardware configuration on which the system should be implemented on is not known at this stage. Nevertheless, in order to support the activities required, the following generic hardware configuration will be required:

a machine with graphics capabilities;

ideally an A4 size high resolution graphics terminal (probably monochrome);

a keyboard;

an alternative cursor manipulation device (a mouse or tablet and pen); and

a graphics printer.

(2) A file system capable of storing and quickly retrieving individual DFDs.

(3) The system should help the user accomplish tasks as easily and conveniently as possible.

(4) The system should allow a user to create and maintain several different SSA projects concurrently. The maximum number of projects the system should be able to support is not known but will depend on the storage capacity of the hardware.

3.2.2. Project Features

(1) Each project should be able to be:

created;
named;
renamed;
amended;
deleted;
viewed; and
printed.

(2) The user should be able to define a hierarchical set of DFDs. It is not known how many levels of DFDs a project will require.

(3) The user should be able to keep definitions of the data and processes within a project in a DD. There should be one (logical) DD for each project in the system.

(4) A process should be named before it is allowed to be expanded into a lower level DFD.

(5) The user should be able to direct the system to test whether a specific hierarchy of DFDs is balanced (so that the child DFD has the same net data flows as the parent process, [DEM78] p. 78).

3.2.3. DFD Features

(1) The elements that should be represented in a DFD are:

data stores;
data flows;
external entities; and
processes.

(2) The user should be able to add new and delete existing DFD components to and from a DFD.

(3) The user should be able to name, rename and (where appropriate) number and renumber DFD elements.

(4) The user should be able to move any existing DFD element to any unused space within the current DFD (except for data flows which must be connected to a source and destination).

(5) The system should provide an automatic numbering facility for those DFD elements that may be numbered. The user should be able to turn this facility off to allow a non-standard numbering system to be used.

(6) When a user is specifying a DFD the system should be able to detect errors arising from:

> Attempts to draw data flows between incompatible DFD elements, for example, drawing a data flow from a data store to another data store without an intervening process.
>
> Attempts to create a DD entry for an un-named DFD element.
>
> Attempts to draw a DFD element atop another.
>
> Attempts to number a DFD element with an already allocated number.

(7) The user should be able to print out all or a subset of DFDs in a project.

3.2.4. DD Features

(1) One DD entry should be able to be defined for each DFD element and for each sub-structure of a DD entry.

(2) The user should be able to ascertain which DFD elements and which components of DD entries are not defined in the DD.

(3) The user should be able to view and edit existing DD entries.

(4) The DD should be able to store six types of DD entry:

> data element
>
> data structure

data flow

data store

process definition

external entity.

(5) The data stored in a DD entry should include, at the least, the name of the element and its number (for numbered elements).

(6) DD entries should have a NOTES section where the user may describe any relevant details in free-form text.

(7) Each type of DD entry (except for external entities) should contain a DEFINITION of the named element. For processes this definition should be in a process description language. A BNF-type language (such as that described in [DEM78]) should be used for data stores, data structures and data flows.

(8) The following information may be required to be stored in the DD:

data element

- name
- description
- definition - values and meanings
- aliases.

data structure

- name
- description
- definition
- volume.

data flows

name
description
definition
aliases
volume
type [1]
source and destination(s).

data stores

name
description
definition
aliases
data flows in and out
arguments to flows
number
organisation.

processes definitions

name
description
definition
data flows in and out
type
number.

[1]'Type' fields are used by SSP.

external entities

name
description
data flows in and out.

(9) When the user is defining a DD entry, as much of the DD entry as possible should be filled in automatically.

(10) The user should be able to request (and receive) the following DD reports:

Listing of entire DD contents in alphabetic order.

Partial listing of DD contents in alphabetic order, with range specified by start and end DD entry names.

Listing of all DD entries, in alphabetic order, corresponding to the DFD elements of one or more DFDs.

Listing of all DFD elements in a specified set of DFDs that are not defined in the DD, ordered alphabetically within DFD.

CHAPTER 4

A Comparison of SSA Methodologies

Although the two major proponents of SSA, DeMarco and Gane & Sarson, have developed methodologies which incorporate the techniques outlined in Chapter 2, there are noteworthy differences between the two methodologies. The more significant of these differences are discussed in this chapter and an amended form of SSA tailored to the needs of the identified users is outlined. This amended form of SSA has been incorporated in an automated tool: MUSSAT (Massey University Structured Systems Analysis Tool). Specific features of MUSSAT are discussed in subsequent chapters.

The major difference between the DeMarco and Gane & Sarson versions of SSA is the difference in DFD size and the associated levelling conventions as discussed in section 4.1.1. Other differences are also outlined in following subsections. In each case where there are differences in the Gane & Sarson and DeMarco approaches, the approach of each methodology is discussed and then the approach that MUSSAT incorporates is outlined. The new form of SSA used in MUSSAT is summarised as a series of rules in Appendix II.

4.1. Data Flow Diagrams

4.1.1. Levelling Conventions

The most significant difference between the two methodologies lies in their DFD levelling conventions. Gane & Sarson suggest that each process in the top level diagram "...can itself be broken down to a third level of detail...", although this is "...not often necessary." [GAN80] (p. 18). On the other hand, DeMarco suggests that a set of DFDs may consist of "...several middle levels, sometimes as many as eight or nine..." [DEM78] (p. 77).

These two views are reconcilable. Gane & Sarson place no limit on the size of their top level DFDs: they are expected to be large with minimal partitioning. A simple example given in the text [GAN80] (pp. 64-65) consists of 22 processes, 12 data stores and numerous data flows. This does not appear to be an atypically large DFD using the Gane & Sarson approach. Considering the complexity of such a top level diagram, it is reasonable to expect a logical system model to be defined in three (or fewer) additional levels of DFDs.

DeMarco, however, says that "...we cannot just expand the size of our diagrams to cope with larger and larger requirements." [DEM79] (p. 71). His approach is to apply strict partitioning conventions, producing a greater number of small, comprehensible DFDs. DeMarco suggests that each DFD

should fit on one A4 size page with seven plus or minus two processes on a page, although this guideline may be relaxed so long as resulting DFDs are easy to understand. Top level diagrams may be larger than optimal, however, these diagrams are working diagrams and need not be readable at a glance. Nevertheless, no example DFD in DeMarco's book is anywhere near the size and complexity of the Gane & Sarson example discussed above.

The DeMarco emphasis on limiting the number of processes in one DFD and allowing more levels in a hierarchy of DFDs can be seen as the opposite approach to the Gane & Sarson methodology of limiting the depth of the hierarchy and not restricting the size of any one DFD.

Seven plus or minus two as a guideline for the number of processes on one DFD is a useful measure of DFD readability, however, even DeMarco views this range as a guideline, not a hard and fast rule. He emphasises the need for comprehensible DFDs and acknowledges that sometimes larger DFDs are necessary and present a more natural portrayal of the portion of the system in question. DeMarco's major concern is DFD readability; the seven plus or minus two guideline is secondary to DFD readability and "...the final standard has got to be your own judgement on resulting readability." ([DEM78] p. 82).

Ensuring that a DFD has no less than five and no more than nine processes does not ensure that the DFD is conceptually simple and easy to read. Some authors (including [TEA85] and [DAV83]) have interpreted DeMarco's guideline literally and limit the number of processes in a DFD without emphasising the need (and the original application of the seven plus or minus two guideline) to produce DFDs that are easy for users to read.

Although the seven plus or minus two guideline is a useful psychological principle, there is no inherent characteristic of an information system that ensures that its system model can be meaningfully decomposed into any particular number of subsystems or functions [TAT86b].

In addition, a SSA requirements specification is also used by systems designers during the detailed design phases of the project. It may be that these users of DFDs are more concerned with the level of detail and completeness of the DFDs rather than size and readability.

The underlying philosophy regarding DFD levelling conventions in MUSSAT is that DeMarco's smaller DFDs serve as better communication tools than do the less constrained Gane & Sarson DFDs. Even so, insisting that all DFDs are drawn with seven plus or minus two processes may force artificial partitioning within a system model.

Hence, MUSSAT encourages the user to draw small DFDs, however, because of the flexibility of DeMarco's seven plus or minus two guideline, the number of processes in one DFD is limited only by the physical drawing area of a DFD. MUSSAT can inform the user of those DFDs with more than nine or less than five processes, but this is not necessarily an indication of a poorly partitioned DFD.

Using MUSSAT the user may draw DFDs consisting of any even number of A4 pages, however, most DFDs drawn are expected to fit on one A4 sheet of paper. MUSSAT encourages the use of small DFDs by requiring the user to take specific action(s) to draw a large DFD, that is, the user is required to select certain system options to draw a DFD larger than one A4 page.

4.1.2. DFD Construction Conventions

DFD Completeness

Gane & Sarson suggest that a DFD should be fully specified before any processes are exploded into lower level DFDs ([GAN80] p. 51). DeMarco does not state that a DFD should be fully specified before processes are exploded, but his approach does seem to suggest it ([DEM78] pp. 63-69).

Insisting that a DFD is fully constructed before lower level DFDs can be drawn ensures that:

(1) The analyst understands the portion of the system being analysed well enough to draw a readable DFD for it.

(2) The interfaces between the DFD elements will be correctly specified and hence the inputs and outputs of a child process will be correctly specified.

If, for example, DFD elements need not be named before a child DFD is drawn for a process, then it would be difficult to check the consistency between a child and parent process when the data flows into and out of a parent process are unnamed and flow from unnamed sources or to unnamed sinks. An incomplete DFD also suggests that the analyst either does not understand fully what the DFD should contain or that the partitioning of the DFD is deficient since the DFD cannot be drawn without violating the DFD construction rules.

In MUSSAT a DFD should be syntactically, but not necessarily semantically, correct before any processes are exploded into lower level DFDs. In other words, a DFD must comply with all the DFD construction rules of the SSA methodology. These rules include the basic SSA rules outlined in Chapter 2 and other rules specific to MUSSAT that are introduced in this chapter.

Numbering processes

In MUSSAT a process need not be numbered before it is exploded into a lower level DFD because numbering is a

notational tool and does not add to the meaning of the DFD; a DFD can be syntactically and semantically correct without processes being numbered. Even so, in a SSA project of any non-trivial system, processes should be numbered for the convenience of the analyst and intended audience of the DFDs.

An analyst is expected to number processes once a DFD is reasonably well defined. MUSSAT allows processes to be numbered at any stage of DFD development, however, a hierarchy of DFDs with unnumbered processes is difficult to maintain and hence the analyst is expected to number processes at least once more than a few DFDs have been drawn.

Order of DFD construction

Although SSA is a top-down analysis tool, there may be cases where the analyst is unable to develop a set of DFDs in a strictly top-down manner. DeMarco suggests that when producing a SSA model for the current physical environment, the analyst may need to start analysis at some intermediate level of the system ([DEM78] p. 260). Therefore, MUSSAT allows DFDs to be drawn in any order. The analyst is responsible for deciding when to deviate from a top-down approach.

4.1.3. DFD Element Naming Conventions

Valid DFD element names

Differences exist in the conventions used to name DFD elements. DeMarco suggests that names of data flows (and presumably other DFD elements) should be in upper case with hyphens joining individual words in the name ([DEM78] p. 54).[1] Gane & Sarson, however, propose that data flow names should be written in upper and lower case (the first letter of the name in upper case) until a DD entry is defined for the data flow. Once the data flow is defined, the data flow name should be changed to all upper case ([GAN80] p. 40).

Gane & Sarson do not specify naming conventions for other DFD elements, but from the illustrations in their text, it appears that data store and external entity names are written in upper case and process names in upper and lower case.

MUSSAT allows any combination of printable characters (except the DeMarco relational operators), including blanks, for DFD element names. An analyst may choose to use hyphenated data flow names, as DeMarco does, but this convention is not enforced.

---

[1] Both Gane & Sarson and DeMarco allow certain types of data flows to remain unnamed. These exceptions are discussed in section 4.1.5. The naming conventions discussed in this section (4.1.3) apply to all processes, external entities, data stores and those data flows that are named.

Case sensitive names

DFD element names in MUSSAT are not case sensitive: Customer-Name, CUSTOMER-NAME and customer-name all identify the same DFD element. Changing a data flow name to upper case once a DD entry is defined (as Gane & Sarson do) is not considered useful for the following reasons:

(1) In a manual system, changing the case of a data flow name means that the name has to be erased and rewritten. Maintenance of pencil and paper DFDs is tedious enough without this additional clerical task.

(2) Although a data flow in upper case may be used to show that a data flow has been defined in the DD, it does not show the current state of the DD definition. For example, a data flow may have been defined in the DD and then the DFD changed to make the existing data flow definition incorrect. Unless the DD entry for the data flow is updated, the DFD may give an inaccurate representation of the state of the system by implying that the now invalid data flow has a valid definition in the DD.

If data flow names differing only in case are required to name two data flows uniquely, then one of the following situations is likely to exist:

(1) The data flows should be defined as duplicates of each other, or if the contents of the data flows are not identical then;

(2) One of the data flow names should be changed. If it is difficult to rename either data flow because they are used for similar purposes, then this may suggest poor functional partitioning.

Using MUSSAT, analysts may choose to use upper and lower case data flow names to differentiate between special types of data flows such as error conditions or major data flows.

Compound data flow names

Both Gane & Sarson and DeMarco allow a data flow to be named with a compound name. For example, if a data flow consists of two structures: EMPLOYEE-NAME and EMPLOYEE-SALARY, then, rather than defining a new data flow consisting of these two structures, the data flow could be named EMPLOYEE-NAME, EMPLOYEE-SALARY (using the Gane & Sarson notation) or EMPLOYEE-NAME + EMPLOYEE-SALARY (using the DeMarco notation). This naming convention is useful when a data flow consists of only a few elements.

In an attempt to allow as much design freedom as possible, data flow names that consist of two or more elements are allowed in MUSSAT. Such data flow names should be specified using the DeMarco relational operator, '+', as used

above.[2] When data flow names consisting of multiple elements are used, each element should be defined in a separate data flow DD entry.

Using compound data flows names is only useful for names consisting of a few elements; if too many names are used the DFD becomes cluttered. If a large number of component names are required to name one data flow, then a simple data flow name should be used and the data flow broken into component flows in the data dictionary. The analyst must determine when a data flow should be named with a single name, perhaps consisting of several other data flows, and when to name a data flow with a compound name.

4.1.4. DFD Element Symbols

Figures 4.1a and 4.1b show the DFD symbols used in the DeMarco and Gane & Sarson (Gane & Sarson) versions of SSA.

[2]The DeMarco relational operators are used in the DD of MUSSAT. The reasons for selection the DeMarco operators are discussed in section 4.2.3.

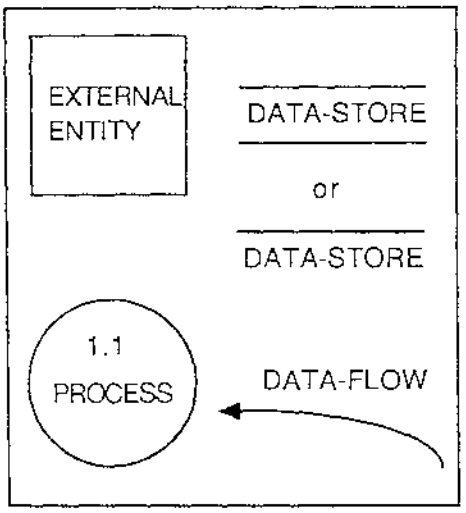


Figure 4.1a - DeMarco DFD elements.

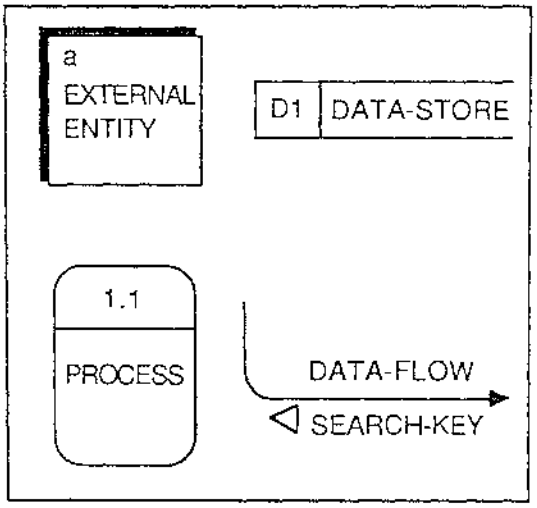


Figure 4.1b - Gane & Sarson DFD elements.

Figure 4.2 shows the DFD symbols used in MUSSAT.

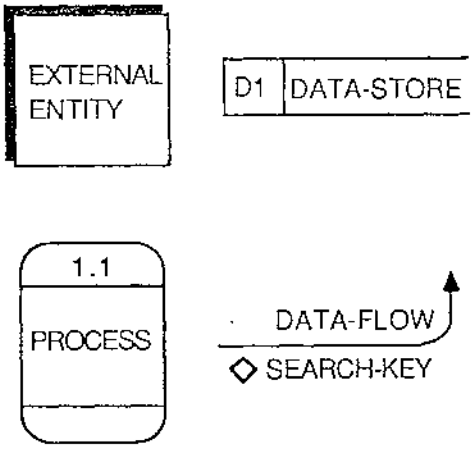


Figure 4.2 - Tailored SSA DFD elements.

Data flows

DeMarco uses curved data flow arrows whereas Gane & Sarson use arrows with rounded corners. Figures 4.3 and 4.4 illustrate these differences.

DIAGRAM 0 : EXAMPLE DFD

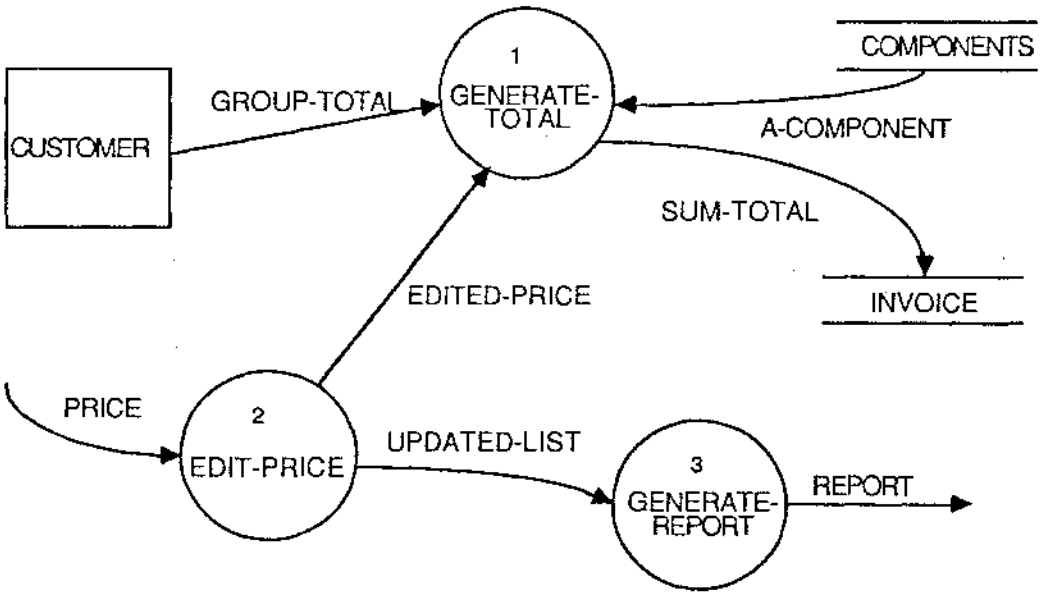


Figure 4.3 - A level 0 DFD, DeMarco style.

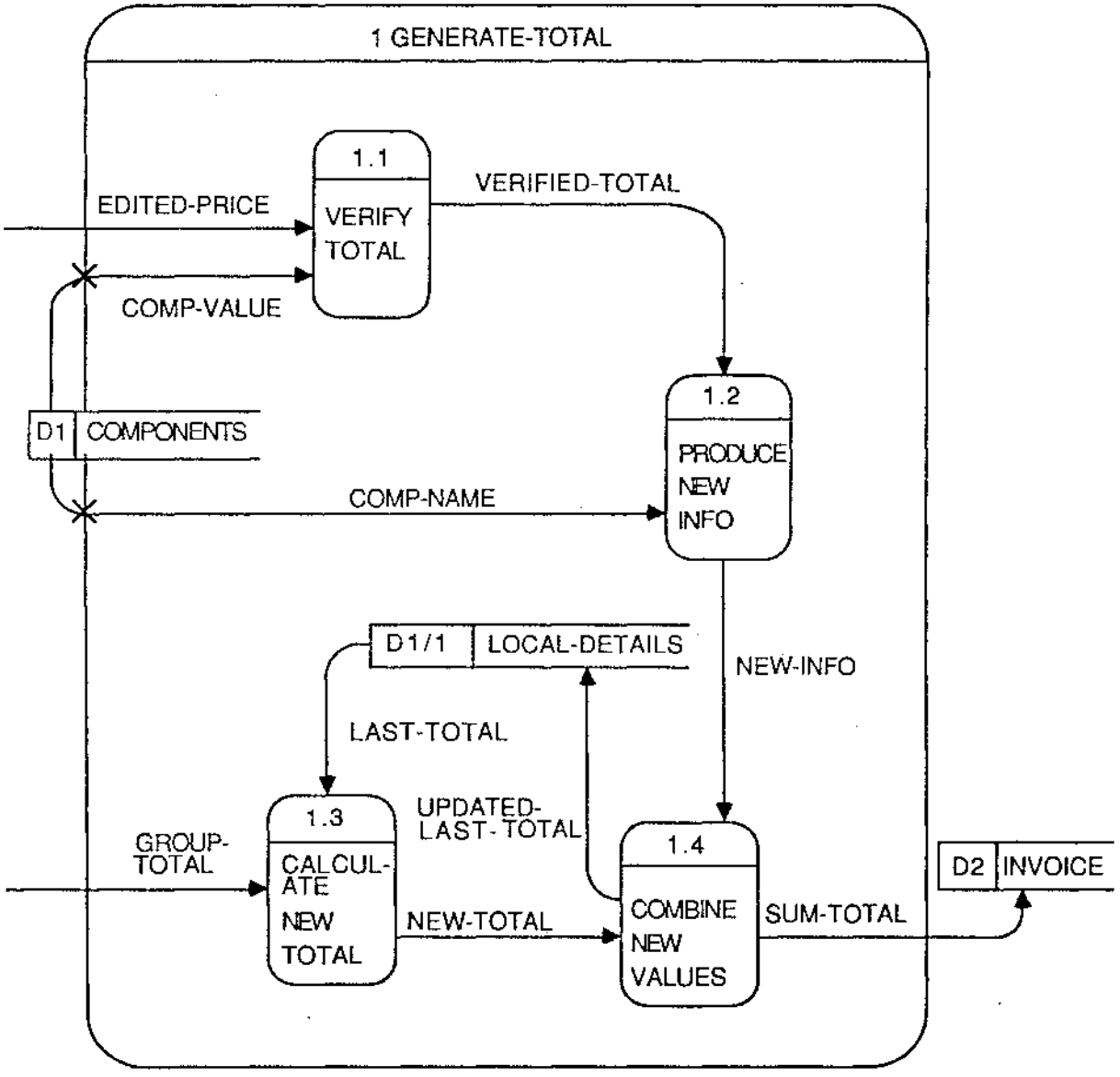


Figure 4.4 - Gane & Sarson exploded process 1.

The Gane & Sarson rounded cornered data flow representation is used in MUSSAT rather than DeMarco's curved data flows because the former are easier to label and draw and because the users prefer the Gane & Sarson representation.

Processes

The same convention is used by both Gane & Sarson and DeMarco for numbering processes, although the shape of the process bubble differs. Gane & Sarson use a rounded rectangular shape for processes as they claim that "...it is hard to get much legible writing inside a circle..." [GAN80] (p. 37). Gane & Sarson process bubbles may also include a physical reference in the lower portion of the bubble. This section is intended to be used during analysis of the existing system to note where the function is carried out, or after system design to note what part of the system performs the function. Process names in both methodologies should consist of an active verb and a simple object clause. Figure 4.2 shows the process bubble shape used in MUSSAT. The process bubble is identical to the Gane & Sarson representation except that the process bubble does not include a physical reference area, although a short comment may be placed in the lower portion of a process.

The Gane & Sarson rectangular process shape was chosen because it is easier to fit a process name and number in a rectangle than a circle. Also, the analyst, SSP and EDFD

users groups prefer the rectangular process bubble.

Data stores and external entities

The DeMarco and Gane & Sarson representation of external entities and data stores are similar. Both use a square to represent external entities; although Gane & Sarson outline the top and left hand side of each square with a solid line. DeMarco shows a data store as a name between two open ended horizontal lines or underneath one line. Gane & Sarson use a narrow rectangle open at the right hand end with a section at the left hand end for a data store number.

The Gane & Sarson and DeMarco versions of SSA allow external entities and data stores to be duplicated on DFDs. In addition to naming external entities, Gane & Sarson also label each external entity with a lower case letter in the upper left hand corner of the box. If external entities need to be duplicated on a DFD, then the original and duplicated box(es) are marked with a number of lines in the lower right hand corner of the box. Duplicated data stores are similarly identified with a number of parallel lines next to the data store number. Gane & Sarson do not make it clear exactly how these parallel lines are to be used. The following two interpretations are possible from their text [GAN80]:

(1) Each time a DFD element is duplicated all occurrences of the element are marked with a number of lines, the number of lines determined by how many other DFD elements have duplicates. For example, all instances of the first duplicated DFD element will have one line, all instances of the second element that is duplicated, two lines, and so forth.

(2) The number of lines shows how many duplicates of the DFD element have been drawn. For example, an external entity with three parallel lines in the lower right hand of the box means that there are four copies of the external entity: one original and three duplicates.

Martin [MAR87] states that the second case is the Gane & Sarson standard ([MAR87] pp. 181 - 183).

DeMarco does not mark duplicate external entities or data stores in any way; all occurrences of a duplicated element are identical.

Gane & Sarson number local data stores differently from other data stores. For example, data stores used only in DFD number 5.1.2 would be numbered D5.1.2/1 to D5.1.2/n, with n being the number of local data stores in the DFD.

In MUSSAT the identifying letter in the upper left hand corner of external entities is not used and neither duplicated data stores nor external entities are marked with

parallel lines.

These conventions were a result of the decision to encourage use of DeMarco's small DFDs. In a large DFD with many DFD elements the additional visual cues provided by a lower case letter in the corner of external entities and parallel lines in duplicate data stores and external entities are useful. In small DFDs with only a few DFD elements, these additional identifiers are not necessary.

Numbering data stores is still useful in small DFDs to quickly identify data stores with similar names, such as CUSTOMER-MAILING-LIST and CUSTOMER-BAD-LIST. In MUSSAT the use of data store numbers (D-numbers) in data stores is not compulsory. The automatic numbering facility incorporated in MUSSAT uses the Gane & Sarson numbering system. The user may alternatively number a data store with a short string of characters if a different numbering convention is required.

In MUSSAT the Gane & Sarson local data store numbering convention is not used. Such a numbering system is potentially confusing since data stores local to, say, DFD 4 would be numbered D4/1..D4/n, which may suggest to the reader of the DFD that these data stores are related to a non-local data store D4, which is not necessarily the case.

4.1.5. Data Flow Details

Duplicate data flows

For the purposes of this thesis a duplicate data flow is a data flow with the same name and definition as another data flow. An implicit form of duplicate data flow is formed when two or more unnamed data flows from the same data store are drawn. These data flows are discussed in the next sub-section.

Neither Gane & Sarson nor DeMarco allow named duplicate data flows, although DeMarco does allow a data flow to diverge, that is, one data flow may flow to more than one end element.

MUSSAT allows the analyst to explicitly define duplicates of named data flows. All instances of a data flow with the same name are defined in one DD entry.

Although the ability to created duplicate data flows is not essential to the development of a SSA system model, duplicate data flows are useful in DFDs where two (or more) copies of a data flow have the same contents and are seen by the analyst and users as identical. Aliases could be used for duplicate data flows, but using different names for such data flows would not show the logical equivalence of the duplicate data flows.

Even though the ability to create duplicate data flows is incorporated in MUSSAT, it is still the responsibility of the analyst to ensure that such data flows are used correctly. In some cases duplicate data flows may suggest that redundant data is included in one (or more) of the duplicate occurrences of a data flow.

Consider the data flows in figure 4.5a.

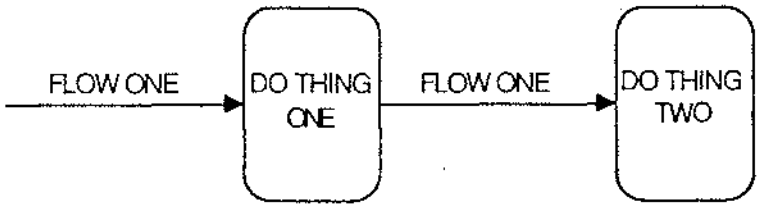


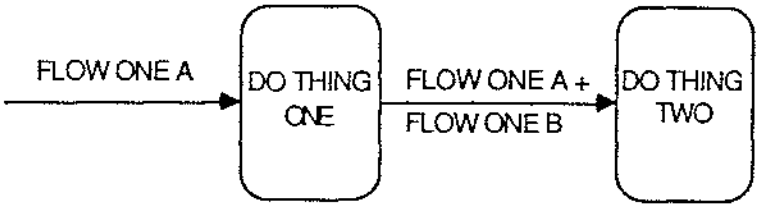
Figure 4.5a - Questionable use of duplicate data flows.

process DO THING A does not appear to transform FLOW ONE since the data flow in is the same as the data flow out of the process. This suggests that either:

(1) one of the processes does not perform any useful data transformation; or

(2) that at least one of the data flows contains redundant data.

Figure 4.5b may be a better representation of the processes and data flows of figure 4.5a.



(Where FLOW ONE = FLOW ONE A + FLOW ONE B)

Figure 4.5b - Removing redundant data in duplicate data flows.

The point is that the ability to show duplicate data flows is useful, however, this facility should be used judiciously. Duplicate data flows should be scrutinised to ensure that they really are duplicates and not just poorly defined data flows containiing redundant data.

Named and unnamed data flows

Both SSA methodologies allow data flows into and out of simple data stores to remain unnamed if their contents are obvious; DeMarco states that all other data flows should be named ([DEM78] p. 54).

Unnamed data flows between processes and data stores serve as useful abstractions in high level DFDs. Single headed, unnamed data flows to or from a data store represent a net update or retrieval (respectively) of data. Unnamed data flows connected to a data store are helpful during early stages of analysis because when the top level DFDs are created the analyst may not be able to precisely define, or

may not be concerned with, the detailed contents of the data store or data flow. In such cases the analyst is only concerned with showing that a net flow exists. Lower level DFDs are expected to define unnamed data flows in more detail.

The assumed contents of a single headed unnamed data flow connected to a data store are the contents of the data store.

If an unnamed data flow is used, unnecessary data could be transported by the data flow since the contents of the data flow are assumed to be the entire data store. Unnamed data flows are expected to be expanded into named data flows in one or more lower level DFDs. In an automated system, checks can be made to see which elements passed in an unnamed data flow are used in lower level child DFDs. As a result, the analyst may choose to name previously unnamed data flows if the contents of the data flow are different from the assumed contents.

In addition to allowing unnamed data flows between processes and data stores, Gane & Sarson also allow data flows between two processes or between a process and an external entitiy to remain unnamed so long as the contents of the data flows are able to be explained by the author of the DFD ([GAN80] p. 40). In other words, these data flow are defined, but only in the mind of the analyst. This is unacceptable for

four reasons:

(1) It assumes that the creator of the DFD will always be available to describe the contents of any unnamed data flow.

(2) Although the meaning of the unnamed data flow may appear obvious to the analyst (and possibly most of the intended readers of the DFD), some users may assume an incorrect definition of the data flow.

(3) The analyst's implicit definition of the data flow may change over time. Unless the contents of a data flow are formally documented any inconsistencies arising from changes in the contents of the data flow may be difficult to detect.

(4) Consistency and completeness checks are difficult to carry out for DFDs with data flows whose contents are undefined. The contents of an unnamed data flow may be inferred from the leaf level DFD of the source process of the undefined data flow (except in the unlikely case that the unnamed data flow occurs in a leaf level DFD) but this means that checks cannot be carried out until the process is defined at a low level. Considerable backtracking may be necessary if inconsistencies in an unnamed and undefined data flow are not detected at a high level.

If the contents of a data flow are obvious, then the analyst should have no trouble in naming and providing an explicit DD entry for the data flow. DeMarco suggests that data flows that are difficult to name are likely to be the result of poor partitioning ([DEM78] p. 66).

In a system model produced using MUSSAT, data flows between processes and data stores may remain unnamed, however, unnamed DFD elements, other than unnamed data flows between data stores and processes are treated as incompletely specified elements and are ignored in system generated reports. Hence, it is in the best interest of the analyst to ensure that as many DFD elements as possible are named before system generated reports are produced.

Double headed data flows

In addition to allowing single headed unnamed data flows between processes and data stores, Gane & Sarson and DeMarco also allow double headed (that is, two-way) unnamed data flows between processes and data stores. Such double headed unnamed data flows are assumed to update and retrieve the entire contents of the data store.

MUSSAT allows the use of double headed named and unnamed data flows between processes and data stores. The contents of the data flow are assumed to be the same in both the flow into and the flow out of the data store.

Data flow search arguments

The Gane & Sarson approach allows search arguments to data stores to be shown on any data flow that serves as a pipeline for data retrieved out of a data store. Such search arguments are shown by a small triangle pointing towards the data store and preceding the name of the data element(s) acting as the key (as in the data flow in figure 4.1b).

In MUSSAT search keys may optionally be specified for single headed data flows originating from a data store, as in the Gane & Sarson SSA.

Data flow end points

In the Gane & Sarson SSA methodology all data flows, except for net data flows out of a process within the boundaries of a child DFD or data flows flowing into a child DFD from a process outside the DFD, should have both end points specified. The DeMarco approach is similar except that all net data flows into or out of a child DFD, regardless of the source or sink of the data flow, need only have one end point specified. If a data flow is entering a child DFD then only the destination of the data flow within the child DFD needs to be shown; a data flow leaving a child DFD needs only to be connected to the source DFD element within the child DFD.

The DeMarco convention is adopted in MUSSAT: all data flows other than data flows representing a net flow of data into or out of a DFD must have both end points specified. For net data flows into or out of a DFD, the data flow need have one end point connected to the process bubble acting as the data source or sink within the DFD. When only one end point of a data flow is specified the identity of the missing end point is derived from the parent DFD. All data flows with a missing end point are assumed to be net flows into out of the DFD and the contents of such data flows are determined from the DD entries of the data flows into and out of the parent process. All other data flows local to the child DFD must have both a source and destination specified.

Intersecting data flows

DeMarco says that data flow lines that intersect are the "...Ultimate Crime Against Nature..." ([DEM78] p. 71) and should not be allowed. If DFDs are drawn according to DeMarco's conventions (that is, DFDs with no more than nine processes)then, usually, there should be no need for data flows to intersect. Gane & Sarson suggest that intersecting data flows should be minimised (not eliminated). In large DFDs intersecting data flows are sometimes inevitable, even when duplicate data stores and external entities are used. In the Gane & Sarson methodology, when data flows cross, one of the intersecting data flows is drawn with a hoop over the point of intersection.

In MUSSAT, intersecting data flows are allowed, and when they occur the Gane & Sarson hoop notation is used. Although small DFDs are encouraged using MUSSAT, larger DFDs are allowed. Hence, it is likely that intersection data flows may occur, so there should be some way of allowing such data flows to be represented as clearly as possible.

4.1.6. Lower Level DFD Details

Representation of lower level DFDs

The representation of exploded, or lower level DFDs is similar in both methodologies. Cosmetic differences exist in representing:

(1) the boundary of a lower level processes;

(2) source and sinks of net data flows into and out of an exploded process,

(3) local data stores; and

(4) data flows shown as parallel decompositions of parent data flows.

Using Gane & Sarson techniques, the contents of an exploded process are enclosed within a larger process bubble labelled with the parent processes name and number (see figure 4.4). Figure 4.3 shows a second level DFD using DeMarco DFD conventions. The explosion of process 1 (GENERATE-TOTAL) using

the Gane & Sarson representation is shown in figure 4.4.

Local data stores are drawn inside the boundary of the parent bubble and are numbered with the parent process number plus a local qualifier (as was discussed in section 4.1.4). Data stores that exist outside the exploded DFD are drawn either totally or partially outside the boundary of the DFD.

Data flows decomposed in parallel with the parent processes are marked with an "X" on the child process boundary. Gane & Sarson also suggest that any processes that are not true decomposition of higher level processes (that is, they are additional processes added at a lower level) should be labelled to show this using names such as X1, X2 etc. An "X" on a DFD element in any child DFD means that the marked element has not appeared in the parent DFD, either because the element is a parallel decomposition of a data flow, a local data store, or an augmented process.

The DeMarco representation of process 1 is shown in figure 4.6.

DIAGRAM 1: GENERATE TOTAL

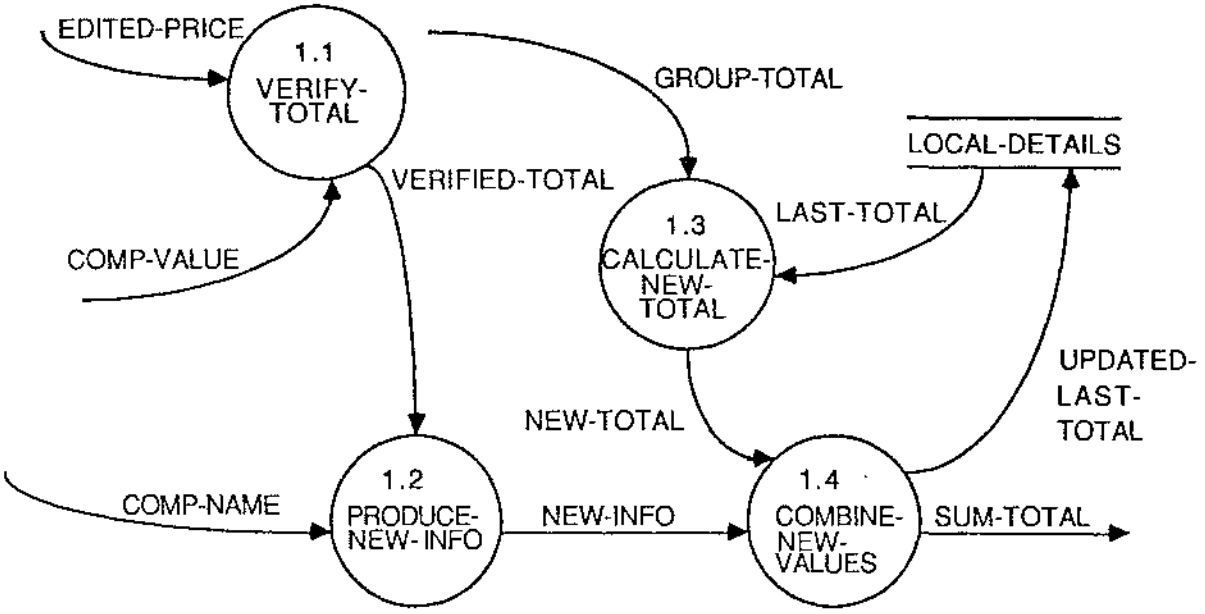


Figure 4.6 - DeMarco exploded process 1.

DeMarco does not enclose an exploded process with any boundaries and the source and destinations of net data flows into and out of an exploded process are not explicitly shown. To see the destination or source of a data flow that leaves or enters a child DFD, one must refer to the parent diagram. Data stores local to a child DFD are not suffixed with a local qualifier as in the Gane & Sarson approach; all data stores are numbered in the same way. No treatment is suggested for processes derived other than by strict functional decomposition.

A representation of child DFDs similar to the Gane & Sarson approach is adopted in MUSSAT. DFDs are drawn within a boundary and data flows may be cross this boundary to

indicate a source or sink outside the DFD.

Nevertheless, data flows crossing the DFD boundary need not be connected to a graphical representation of the source or sink DFD element in the parent DFD as in the Gane & Sarson approach. Using MUSSAT the analyst may choose to duplicate data stores acting as sources and sinks of net data flows to and from a child DFD within the child DFD, especially when a data flow on the parent DFD is decomposed into several in a child DFD. For example, figure 4.4 shows two data flows from the data store COMPONENTS. In the parent DFD (figure 4.3) these flows are shown as one composite flow: A-COMPONENT. In a MUSSAT child DFD the user may duplicate the data store COMPONENTS to show the source of COMP-VALUE and COMP-NAME.

The ability to show data flows crossing the DFD boundary is a notational tool for the analyst and such data flows have the same meaning as data flows that do not cross the boundary. The analyst may choose not to show data flows crossing the DFD boundary when the source or sink of the data flow is outside the DFD. Such DFDs are equally valid.

Some indication of the net flows into and out of a DFD from the parent DFD should be given when drawing a child DFD. This gives the analyst a manual consistency check between the parent and child DFD.

Numbering lower level processes

DeMarco suggests that processes in lower level DFDs need not contain the entire process number. For example, in a DFD numbered 6.2.1.1, processes would normally be numbered from 6.2.1.1.1 to 6.2.1.1.n. Instead of displaying the full number in each process only the last digit would be shown (that is, .1 to .n).

A numbering system similar to DeMarco's is adopted in MUSSAT. Each process bubble will be numbered with no more than two digits. The DFD will always be numbered with the entire DFD number, for example, in a DFD numbered 6.2.1.1 processes within the DFD would be numbered from 1.1 to 1.n.

4.1.7. Error and Exception Handling

Neither methodology shows error or exception handling at the top level(s). Gane & Sarson suggest that error and exception conditions should be shown in the second level of the DFD hierarchy after which they occur, however if the processing involved is a significant business function (especially if finances are involved), the function may be moved to the top level DFD [GAN78] (p. 49).

MUSSAT provides several features that are intended to help the user identify possible errors and inconsistencies in a SSA model. If error data flows or are not shown, in some form, at top levels of the SSA model then the balancing

checks that MUSSAT performs will inlcude the names of these data flows in a report identifying data flows not derived by decomposition of higher level data flows. In some cases the user may choose not to include low level error data flows in high level aggregate data flow definitions. If so, then the unbalanced data flows identified as the result of balancing checks are not really errors.

The choice of where to first show error data flows or exception handling is left up to the analyst, however, MUSSAT will correctly identify all unbalanced data flows in child DFDs.

4.2. Comparing Data Dictionaries

4.2.1. Types of Data Dictionary Entries

Both methodologies allow intermediate data structures to be stored in the DD, for example, a data store may be defined as a number of logically related groups of data rather than as a series of implicitly related data elements. Gane & Sarson use data structure DD entries for defining substructures in a data store. DeMarco does not use a separate type of DD entry for intermediate data store structures. Hence, whereas in a Gane & Sarson DD a data store may be composed of data elements and data structures, in a DeMarco DD a data store would consist of data flows and data elements. Data flows in both methodologies are defined using subordi-

nate data flows (which may appear in lower level DFDs) and data element entries.

Gane & Sarson state that external entities should have entries in the DD documenting any relevant information. Glossary entries that describe terms peculiar to the system that is being analysed are also kept in the DD. DeMarco does not define external entities or include glossary terms in the DD.

The DD in MUSSAT will consist of the following six types of DD entries:

data element
data store
data structure
data flow
process
sub-process
external entity.

Data structure entries allow a data store to be defined as a logical grouping of data elements and other data structures. This is so that the analyst may define the logical contents and structures in a data store independent of the data flows into and out of the data store, whose structures are based on the data required and produced by processes. External entity entries are included so that any relevant

information may be stored in the DD with the rest of the system definitions.

4.2.2. Data Dictionary Redundancy

Redundancy in the DD refers to explicitly stored data that can be recreated using existing data in the DD and DFDs. Examples of redundant data include:

(1) Names of data flows going into and out of a data store in data store DD entries.

(2) Names of data flows going into and out of a process in process DD entries.

(3) Names of any related data element stored in data element entries.

(4) Names of related data flows and data structures in data structure entries.

(5) Names of the source and destination of the data flow in data flow entries.

(6) Names of all aliases of a DD entry in the main DD entry and reference to the main entry from all aliases.

Gane & Sarson store redundant data (such as that listed above) in the DD. Their approach is to store as much potentially useful information as practical in the DD. As well as redundant data, design and implementation details such as

internal representation of data elements, editing information and file organisation are also stored in the DD. A DD produced using Gane & Sarson guidelines is expected to be useful to system designers and implementors as well as systems analysts.

In contrast, DeMarco emphasises the need to keep the DD free of redundant data. His argument is that although certain types of redundant data are useful, the effort required to maintain such data does not warrant its inclusion in the DD ([DEM78] p. 139). The only exception DeMarco makes is in the treatment of aliases: the main DD entry should list all aliases and all aliases should refer to the main DD entry. DeMarco says that although the DD produced during the analysis phase may be useful after the phase is completed, it should not be developed explicitly for any purpose other than as a descriptive analysis tool ([DEM78] p. 126). Hence the DD should include little implementation data; the implementation data included is recorded in the NOTES section rather than in specific fields as in the Gane & Sarson DD.

In the MUSSAT DD analysts are not be required to maintain redundant data. This decision was made for two reasons:

(1) As DeMarco states, considerable time and effort (expended by the analyst) is required to maintain redundant data.

(2) In MUSSAT, all redundant is able to be maintained by the system and made available to the user when requested.

The following list shows the DD fields that the analyst is required to explicitly specify in the DD (since the information is not found on the DFDs) and the fields that can be determined by MUSSAT from the DD and DFD specifications.

ANALYST MAINTAINED FIELDS	SYSTEM MAINTAINED FIELDS
DATA ELEMENT:	
name	names of aliases
definition	
alias for	
notes	
DATA STRUCTURE:	
name	names of aliases
definition	
alias for	
volume	
notes	
DATA FLOW:	
definition	name
alias for	names of aliases
volume	source and destination
type	
notes	
DATA STORE:	
definition	name
alias for	names of aliases
organisation	number
notes	data flows in and out
	retrieval keys of outward data flows
PROCESS:	
definition	name
type	number
notes	data flows in and out
SUB-PROCESS:	
definition	name
type	
notes	
EXTERNAL ENTITY:	
notes	name
	data flows in and out.

4.2.3. Data Dictionary Definition Languages

Relational operators

The contents and structure of data items are described in both methodologies using a set of relational operators and the names of DD entries. The representation and meaning of the operators used in each methodology are shown below:

DeMarco relational operators:

=	Is equivalent to.
+	And.
[ ]	Selection of one option within the brackets (options are separated with a '|').
X{ }Y or { }	An iteration of the enclosed component(s), the optional lower and upper limits of the iteration indicated by the values of X and Y (respectively).
( )	Component(s) within the parenthesis is (are) optional.

Gane & Sarson relational operators:

{ }	Selection of one option within the brackets (options are listed vertically).
DATA* (X-Y)	An iteration of DATA with optional lower and upper limits of the iteration optionally indicated by the values of X and Y (respectively).
[ ]	Component within the square brackets is optional.

The following is a syntactically correct definition of a data structure named EMPLOYEE-DETAILS using each of the sets of the relational operators defined above:

DeMarco definition:

```
EMPLOYEE-DETAILS = EMPLOYEE-NUMBER +
                   EMPLOYEE-NAME +
                   EMPLOYEE-ADDRESS +
                   (EMPLOYEE-PHONE) +
                   0{EMPLOYEE-QUALIFICATION}10 +
                   [SALARY | WAGE-RATE]
```

Gane & Sarson definition:

```
EMPLOYEE-DETAILS
  EMPLOYEE-NUMBER
  EMPLOYEE-NAME
  EMPLOYEE-ADDRESS
  [EMPLOYEE-PHONE]
  EMPLOYEE-QUALIFICATION* (0-10)
   SALARY
   WAGE-RATE
```

In both approaches the elements of EMPLOYEE-DETAILS could be defined further in the DD. For example, EMPLOYEE-NAME could be defined as:

DeMarco definition:

```
EMPLOYEE-NAME = LAST-NAME +
                0{INITIAL}4 +
                FIRST NAME
```

Gane & Sarson definition:

```
EMPLOYEE-NAME
  LAST-NAME
  INITIAL* (0-4)
  FIRST-NAME
```

If such a definition does not serve any purpose outside the context of EMPLOYEE-DETAILS, then EMPLOYEE-NAME could be included in the EMPLOYEE-DETAILS DD entry rather than as separate entry.

In MUSSAT a data item description language is supported for definition of data stores, data flows and data structures. A data item description language is required so that completeness checks can be carried out to determine which components of DD entries have been defined and to what level. The DeMarco relational operators and definitions were chosen for the following reasons:

(1) The Gane & Sarson approach of putting each data item in a definition on a separate line can create definitions that take up an unnecessarily large space; the DeMarco representation allows several elements on one line in a definition.

(2) The DeMarco '+' operator shows the 'and' relationship between data items explicitly while the Gane & Sarson representation relies on indenting to show such relationships.

(3) The Gane & Sarson method appears to be based on COBOL data definitions: using upper case, hyphenated names and indentation of names to show the relationships between component data items. The DeMarco representation is more flexible and is independent of any

programming language standards; data item names are not restricted to upper case and indentation is not required to show component data items.

Process definitions

Both SSA methodologies use Structured English, Decision Tables and Decision Trees to describe the logic of leaf-level processes. Gane & Sarson differentiate between Structured English and Tight English: the former being the use of a small set of keywords and unambiguous English, and the latter being a form of Structured English expanded to make it more acceptable to users. DeMarco makes no such distinction; Tight English is simply a more terse form of Structured English. When defining the logic of a process, DeMarco suggests that a process that is too large to fit on one DD page may be divided into two or more subprocesses, each having its own DD entry. Such subprocesses could be used by more than one parent process. Gane & Sarson say that if a process description is too large to fit in the DD, then a DD entry summarising the process and including a reference to full logic details elsewhere in the functional specification should be included in the DD.

The DD in MUSSAT does not directly support any of the three process description languages. The MUSSAT DD allows textual definitions of processes and data elements. It is up to the analyst to enforce standards for these types of DD

definitions. A future extension to MUSSAT may be to develop one or more of the above process definition languages for use in the DD.

MUSSAT also allows a process to be defined as a number of sub-entries that can be used by more than one process.

4.3. Other SSA Tools

DFDs and a DD form the basis of the SSA methodology, however, both Gane & Sarson and Demarco use additional graphical tools to describe the logical structure and accesses required in complex data stores: Data Structure Diagrams (in the DeMarco methodology) and Data Immediate Access Diagrams (in the G & S methodology). Both methodologies use these additional tools after all data stores in the current physical model have been normalised to third normal form.

The purpose of each of these two types of diagrams is slightly different. DeMarco's DFDs are used by the analyst to present the logical file design structures to the user. DSDs show the primary key and name of a each logical file; the contents of the file are defined separately using the relational operators discussed earlier. DeMarco suggests that DSDs are a useful communication tool for analysts to use with both data base designers and intended users of the proposed system.

Gane & Sarson's DIADs are used by the analyst to document immediate informational access requirements and hence to help users decide which accesses are of the most value to them. DIADs use the logical file structures produced as the result of normalization to show the immediate accesses requested by users, that is, the accesss on each logical file required in a time period less than the time it takes to search a file serially or sort it. DIADs explicitly show all attributes of a logical file, the primary key(s), and the attributes that act as the key(s) for each immediate access.

Figures 4.7 and 4.8 are examples of a DIAD and DSD (respectively). Both figures are based on a DIAD given in [GAN80], p. 225.

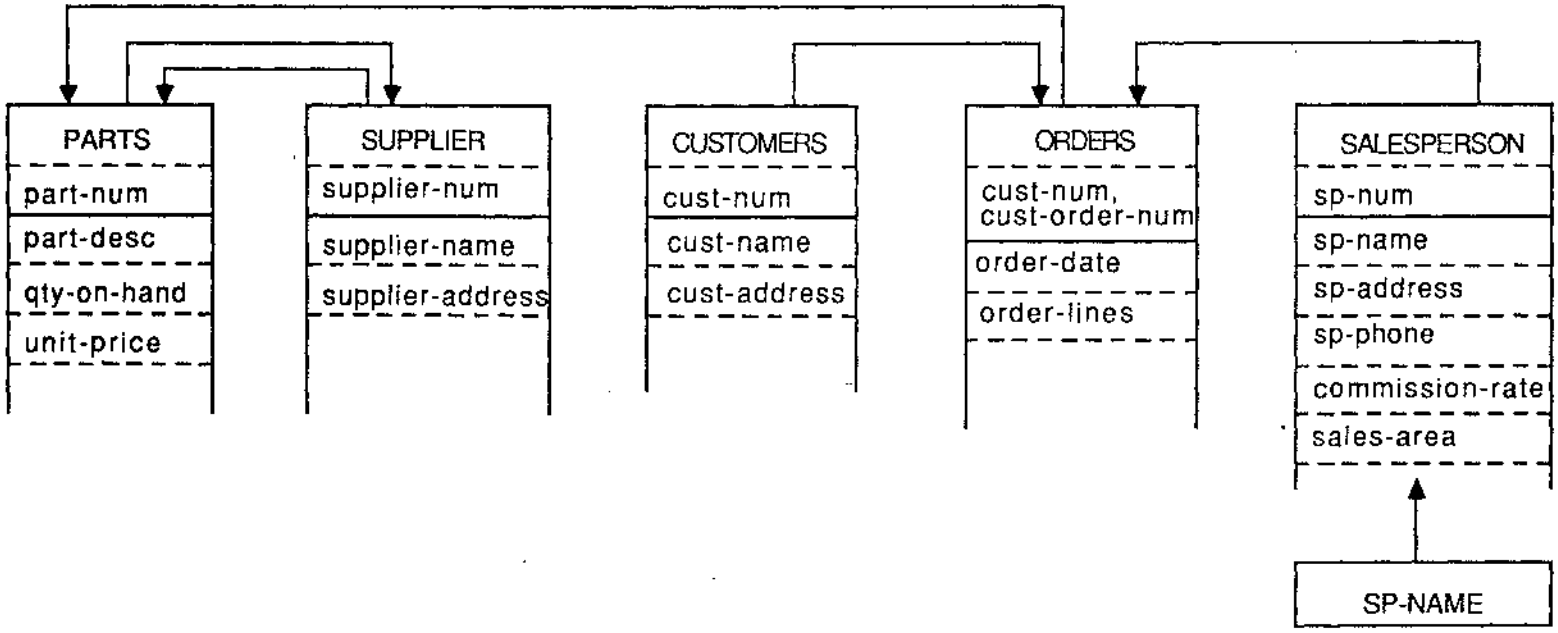
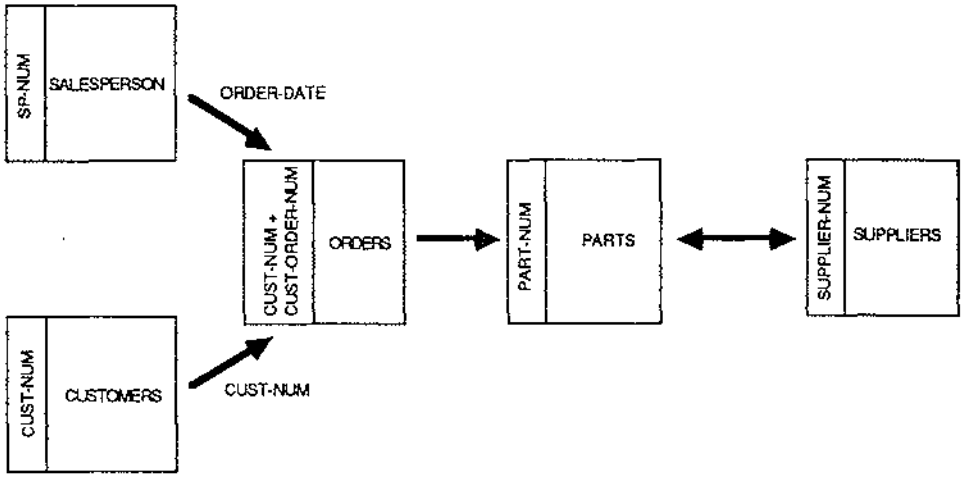


Figure 4.7 - A Gane & Sarson Data Immediate Access Diagram (DIAD).



SALESPERSON = {SP-NUM + SP-NAME + SP-ADDRESS + SP-PHONE + COMMISSION-RATE + {SALES-AREA}}

CUSTOMERS = {CUST-NUM + CUST-NAME + CUST-ADDRESS}

ORDERS = {CUST-NUM + CUST-ORDER-NUM + ORDER-DATE + {ORDER-LINE}}

PARTS = {PART-NUM + PART-DESC + QTY-ON-HAND + UNIT-PRICE}

SUPPLIERS = {SUPPLIER-NUM + SUPPLIER-NAME + SUPPLIER-ADDRESS}

Figure 4.8 - A DeMarco Data Structure Diagram (DSD).

Figure 4.7 is a DIAD showing several hypothetical accesses for five entities: PARTS, SUPPLIER, CUSTOMERS, ORDERS and SALESPERSON. The primary key(s) to each entity is the data item beneath the name of the entity, separated from the other attributes of the entity by a solid line. Each arrow shows that, given an occurrence of the entity that the arrow

starts at, all related occurrences of the entity pointed to should be accessible. The box labelled SP-NAME shows that individual occurrences of SALESPERSON should be able to be retrieved by SP-NAME as well as by the primary key SP-NUM.

Figure 4.8 is a DSD showing the same logical accesses as the DIAD of figure 4.7. The primary key of each entity is shown on the left hand side of each box; the name of the entity is shown in the main part of the box. As in a DIAD, connecting arrows between entitites represent the required logical access paths. The labels on the arrows show key(s), other than the primary key, on which another entities should be accessible.

Unlike a DIAD, in a DSD the attributes of each entity are not shown directly in the diagram, but in separate definitions using the same data description language that is used in the DD. Definitions of the attributes of each entity in figure 4.8 are given in the lower half of the diagram; the key attribute(s) for each entity is underlined. The use of relational operators allows repeating data items, such as ORDER-LINE in ORDERS, to be represented explicitly. In contrast, when using a DIAD, multiple occurrences of an attribute are not shown.

Gane & Sarson suggest that a Materials Flow Diagram should also be included in the functional specification of a system. Although the flow of materials through an organisation

is a physical consideration, Gane & Sarson suggest that it is an important consideration, especially in industries such as manufacturing and distribution. A Materials Flow Diagram shows, not surprisingly, the flow of materials through the logical system and the relationships between the physical materials flow and the logical DFD system model.

Figure 4.9 is a Materials Flow Diagram showing the flow of materials through a simple stock receipt and sales system.

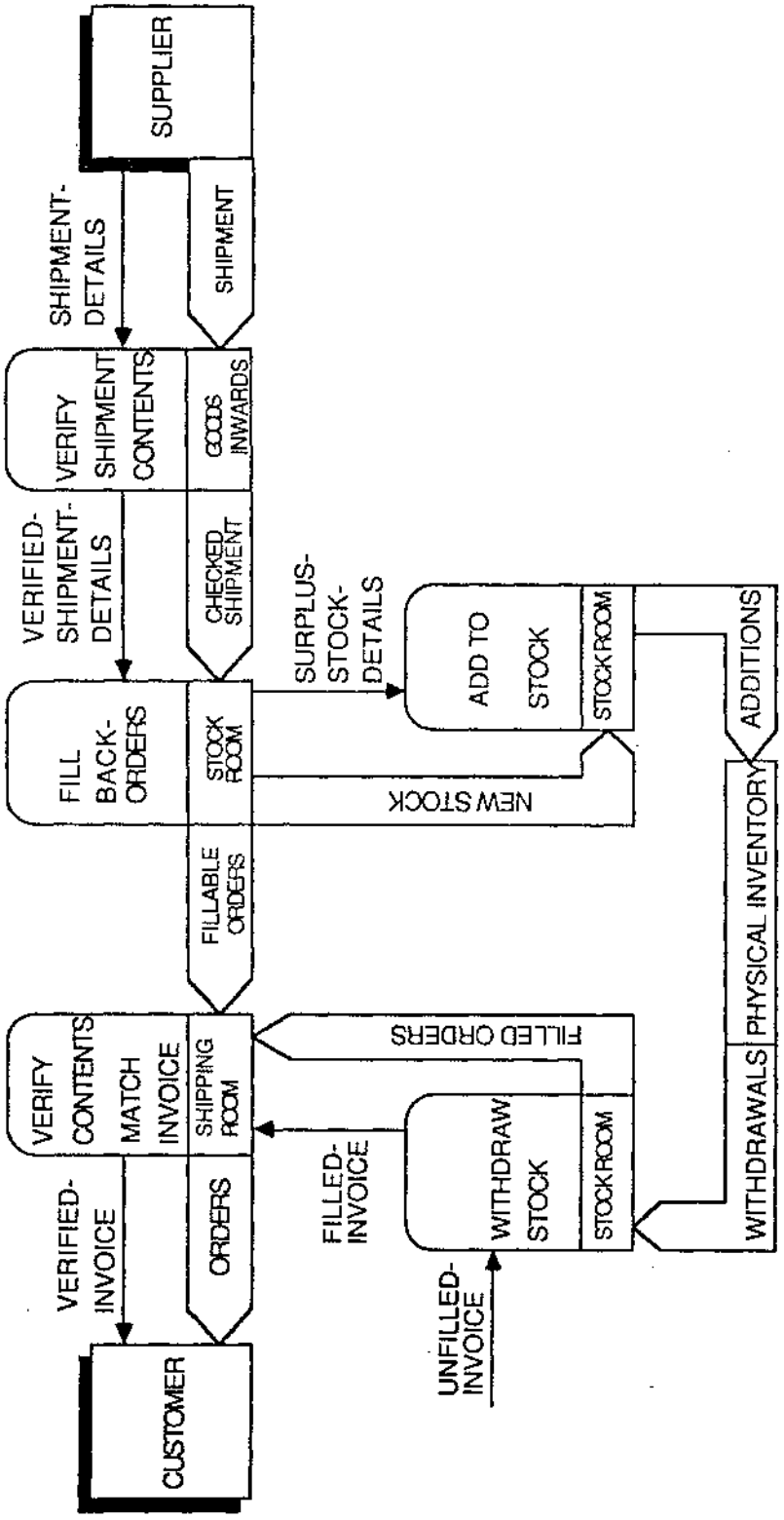


Figure 4.9 - A Gane & Sarson Materials Flow Diagram.

This figure is a simplified form of a Materials Flow Diagram given in ([GAN80] p. 67).

The processes shown in the Materials Flow Diagram may correspond to logical processes in a DFD, in which case the process would be numbered with the same number as the corresponding logical process. Other processes in a Materials Flow Diagram, such as WITHDRAW STOCK and ADD TO STOCK in figure 4.9, are purely concerned with processing the materials flow and do not appear on any logical DFD. Data flows in a Materials Flow Diagram correspond to logical data flows in a DFD and are represented in the same way as in a logical DFD.

MUSSAT does not support the development or maintenance of DSDs, DIADs or materials flow diagrams. None of these diagrams are an integral part of the SSA methodology. In addition, DSDs, and DIADs both described characteristics of the data model which can be shown more concisely, and in greater detail using database design techniques such as the that documented in [TEO82].

Demarco's DFDs are too bulky for anything other than very simple file structures. Only a few simple file boxes can be shown on an A4 page, and within each simple file box there is only a limited amount of space for showing the key; a multiple field key larger than two or three fields would not fit in the file box. Another argument against using DeMarco's DSDs is that a higher level entity-relationship type diagram is required at the analysis stage, with separate diagrams, such as those described by Martin and

McClure ([MAR87] pp. 231-277), showing all attributes of each entity and the relationships between these attribues.

CHAPTER 5

MUSSAT

The general requirements of an automated SSA tool, as outlined in section 3.2, and the tailored SSA methodology of Chapter 4, were used as the basis for designing an automated SSA tool for Analyst and SSP users. The design presented in this chapter describes the functional requirements of MUSSAT. As was discussed in Chapter 1, the functional requirements are only part of the documentation that should be produced during the analysis phase of the software lifecycle.

One of the requirements of the MUSSAT system (as documented in chapter 3.2) is that the user should find the tool easy and convenient to use. This requires two things:

(1) that the characteristics making a system easy to use (for the selected users) are identified; and

(2) that these characteristics are then incorporated in the system design.

Several researchers have identified characteristics that contribute to the ease of use of specific types of systems. In section 5.1 a set of design guidelines used in the development of MUSSAT is outlined. These design goals are most relevant to the design of the user interface, although

they also influence the system design by suggesting what types of operations the users are likely to find easiest to use.

The design presented in this chapter is broken into two major components:

> The user interface, which describes how the user will communicate with the system and vice versa; and
>
> The system model, which describes what the system must be able to do.

Although the design is presented as two separate components, neither the system model nor the user interface could be designed in isolation from one another. The general style of interaction, to be incorporated in the user interface, was suggested by the users at an early stage of the design process.

A decision was made early in the design phase to use an object-action syntax[1] in MUSSAT for the following two reasons:

(1) The users stated a preference for and are very familiar with object-action command languages; and

---

[1] An object-action (also called a postfix) command language, or syntax, requires that the user specify commands in the form object-verb, for example, printing a file is specified by a filename (or filenames) followed by the print command.

(2) No evidence was found to suggest that any one command language style was significantly easier for users to learn and use than any other command language style.

Several researchers (as cited in [CHE86]) have attempted to determine whether postfix or prefix command languages are more effective and easier for users to learn and use. To date, the experimental results are inconclusive.

A recent experiment [CHE86], although concluding that there were no significant differences between using prefix and postfix command languages in a series of text editing exercises, suggests that postfix notation may be chosen more frequently by designers since such a language requires fewer user interactions and fewer keys on the keyboard. Postifx notation may also avoid problems associated with using modes ([CHE86] p. 373).

Sections 5.2 provides an overview of the MUSSAT system model, the details of which are included as Appendices III-VII.

An overview of the MUSSAT user interface is also given in section 5.2, with further detail given in Appendix III. The model of the user interface presented in this chapter and in Appendix III is intended to be a descriptive rather than a definitive specification; the documentation presented gives an overview of the proposed interface without

specifying a formal syntax. An overview description of the interface was considered more appropriate to include in this thesis because:

(1) A less formal interface description was considered to be a more useful communication tool than a formal syntax definition for the intended audience of this thesis.

(2) Some of the components of the interface require further refinement based on research findings outside the scope of this thesis. The specialist areas not considered in detail in the interface design include the following:

(3) the design and use of icons [GIT6], [HEM82], [LOD83], [MCC83];

(4) window size and placement [GAI85], [NOM86]; and

(5) the naming and ordering of options within a menu [LAR82], ROS82].

The interface design presented in section 5.2 assumes the hardware and supporting software configuration outlined in section 3.2.

5.1. Design Goals

Hansen [HAN71] presents a set of design guidelines for interactive systems which he calls user engineering principles. These principles were documented by Hansen as a

result of experiences with several text-editing systems. Hansen's user engineering principles are considered to be relevant to the design of MUSSAT for three reasons:

(1) The general goals of an text-editing system are the same as those of the MUSSAT system: to create, view or modify stored documents. Hansen describes the role of the computer in such applications as "..a tool for the creative worker and the emphasis must be on capturing his thoughts with minimal interference." ([HAN71] p. 218).

(2) The design goals that Hansen proposes encourage the development of a system that fulfills the requirement (from section 3.2.1) that MUSSAT should allow the user to accomplish tasks as easily and conveniently as possible.

(3) The user profile that Hansen presents is also applicable to the intended Analyst and SSP users of MUSSAT.

Hansen claims that his user engineering principles are compatible with, but more comprehensive than, earlier design guidelines ([HAN71] p. 218). Shneiderman [SHN80] summarises several different sets of interactive systems design goals, including Hansen's user engineering principles. Of these different sets of design goals, some appear to be more comprehensive than Hansen's guidelines, although none are as applicable to the general goals of MUSSAT.

Hansen urges the system designer to 'Know the User': to build a profile of the intended users so that their skills, weaknesses and special requirements can be explicitly catered for in the system design. The main points of Hansen's user engineering principles are summarised below (reference is made to [HAN71] for a more detailed explanation of each principle):

(1) MINIMISE MEMORISATION

(i) Selection not entry:

The user should be able to select commands and options from a list or menu rather than having to specify the command in full using keyboard input.

(ii) Names not numbers:

The user should be able to differentiate between commands and options by their names and not simply a code number.

(iii) Predictable behaviour:

The user should be able to predict how specific operations will behave after having formed a general impression of the system.

(iv) Access to system information:

The user should be able to view and modify any variable system parameters.

(2) OPTIMISE OPERATIONS

(i) Rapid execution of common operations:

Frequently used operations should be executed as fast as possible. The user should be provided with feedback for lengthy operations.

(ii) Display inertia:

The display should change as little as possible as the result of executing an operation.

(iii) Muscle memory:

Repetitive operations should be able to be performed by the user with the minimum of conscious effort. This implies that:

> input devices should be as state independent as possible; and
>
> the system should be able to handle bursts of commands, perhaps more than 10 keystrokes per second (where keyboard entry is used).

(iv) Reorganise command parameters:

Observing users as they interact with the system, either through direct observation or system statistics, will identify commands that are not as convenient to use as they could be (for the frequency with which they are executed) and other commands that are used infrequently. The former should be made easier to use and the latter com-

mands made into subcommands.

(3) ENGINEER FOR ERRORS

(i) Provide good error messages:

Error messages should be displayed quickly and should explicitly identify the error with an appropriate error message.

(ii) Engineer out the common errors:

Errors that occur frequently are not the fault of the user and the system should be modified so that the likelihood of such errors is decreased.

(iii) The system must provide reversible actions:

The system should help the user recover from their own mistakes as well as system and software bugs.

(iv) Redundancy:

The system should provide the user with more than one way to achieve a goal: usually providing several simpler operations that can be performed step-by-step as an alternative to a single, powerful command performing the same operations.

(v) Data structure integrity:

Regardless of system or hardware errors, the users must have access to some version, either the current or back-up copies, of their work.

Although these design goals offer useful guidelines to the system designer, they do not provide a means of measuring how well any system design conforms to the guidelines. Nor do the guidelines suggest likely trade-offs between individual design goals. Shneiderman ([SHN80] p. 254) summarises the informality of existing design goals for interactive systems:

> Unfortunately, these lists are only crude guides to the designer. The entries are not independent and sometimes are in conflict. The lists contain contradictory recommendations and are incomplete. Finally, these design goals are largely unmeasurable. Can we assign a numerical value to the simplicity, stability, responsiveness, variety, etc. of a system? How can we compare the simplicity of two design proposals? How do we know what has been left out of the system design?

Hansen's guidelines are informal and may represent no more than what a good designer instinctively knows [EMB81], however, for the purposes of this thesis, these goals are useful design guidelines; a sensible but limited set of guidelines is better than none at all.

Guidelines 2(iv) and 3(ii) suggest that a system design may need to be modified once the intended users have had hands-on experience with some version of the system. This suggests that for some systems, prototyping may be a suitable implementation technique. Users can then use (limited) versions of the target system thereby detecting and allowing correction of deficiencies before the system is fully implemented.

The design presented in this thesis is based on Hansen's user engineering principles, however, since the system has not been implemented, modifications may be necessary once users are able to experiment with the actual system.

5.2. The MUSSAT System Specification

Several different tools and techniques were used to develop a model of the MUSSAT system. SSA was used as the main analysis and design tool with additional text and diagrams used to augment the system design and user interface model. The additional types of diagrams included:

a form of State Transition Diagrams modified to incorporate pull-down menus;

a Data Structure Diagram showing the entities and relationships required in the MUSSAT data base; and

mock-ups of the MUSSAT menus, screen layouts and reports.

As was discussed in Chapter 1, the documentation produced using SSA does not constitute a full requirements specification. The additional components listed above were used to specify those parts of the requirements specification that the SSA functional specification did not cover.

DeMarco ([DEM78] p. 57) suggests that DFDs can be used to model the user interface, however, DFDs were found to be inadequate for this purpose and hence State Transition Diagrams were used. Data Flow Diagrams were found to be deficient for the following reasons:

> There is no way to model what the user will see on the VDU using DFDs, since data flows only described data content, not data format.
>
> An important part of the user interface is the sequence of changes to the screen and the sequences of responses required of the user. Data flow diagrams, as espoused by DeMarco and Gane & Sarson, do not show procedural sequences.[2]

Many of the features of the MUSSAT user interface are based on techniques used in the Apple Macintosh operating system. These features include:

pull-down menus;
icons;
dialogue boxes;
windows; and
cursor shapes as state indicators.

The MUSSAT user interface has been designed for use with a keyboard and an alternate cursor movement device with two buttons. This alternate cursor movement device is assumed to be a two-buttoned mouse, although a pen and tablet or puck and tablet could be used. A mouse has been selected as the most appropriate input device for MUSSAT for the following reasons:

---

[2]Ward and Mellor [WAR85] have developed an extended form of DFDs for use in developing real-time systems that allow depection of control and timing features of a system model.

The intended users of MUSSAT are familiar with and are favourably disposed towards using a mouse; and

a mouse requires relatively little desk-top space in which to operate compared to a tablet and pen or puck.

A two button, rather than one button mouse was selected because a second mouse button allows a one mouse button to be dedicated to the CANCEL or UNSELECT function. The advantage of a dedicated UNSELECT button is that the user can cancel a selection without moving the cursor; in user interfaces designed for use with a one mouse button, the user most often unselects an object or command by moving the cursor and selecting another object or command.

General features of the MUSSAT user interface are discussed in the following sub sections.

5.2.1. Mouse Buttons

Each of the (two) mouse buttons will have a consistent meaning throughout MUSSAT. The left-most button will be the SELECT button and the right-hand button will be the UNSELECT button.

The user will select an object within MUSSAT by moving the cursor so that it rests on the desired object, and then pressing the SELECT mouse button. To unselect a previously selected object, the user either 'clicks' the UNSELECT mouse button (that is, presses and

releases the mouse button in rapid succession), or selects another object using the SELECT button.

In the first case, where the user unselects using the UNSELECT mouse button, no object remains selected after the unselect is performed. When an object is already selected and the SELECT button is used to select a different object, the first object is automatically unselected and the second object selected.

5.2.2. Pull-down Menus

In keeping with Hansen's design guideline of selection not entry, the user issues all commands in MUSSAT using menus (except for the mouse button SELECT and UNSELECT commands and several window manipulation commands). Although not specified in this description of the MUSSAT interface, keyboard equivalents of each command could be incorporated in the system so that experienced MUSSAT users would be able to issue frequently used (if not all) commands without having to reference any menus.

Traditional full-screen menus were considered to be unsuitable for the MUSSAT interface because of the object oriented command language and the graphical style of interaction desired. The other two widely used alternative forms of menu interaction are pull-down and pop-up menus.

In the former case, the title of each menu appears as a soft-button in a line of soft-buttons (often called a menu bar), usually at the top of the screen. When one of these soft-buttons is selected, the corresponding menu appears beneath the menu title and the user is then able to select one of the options within the menu by dragging the cursor to the desired option and then releasing the mouse button[3]. The menu disappears when the user releases the mouse button or moves the cursor out of the menu.

In most cases, all commands associated with a menu are visible when a menu is viewed, however, only those commands that are valid at the point in time that the menu is viewed may be performed. Menu options that are unavailable are de-emphasised (by either dimming or changing the font of the option within the menu). A dimmed menu option may be selected in the same way as a non-dimmed menu option may be, but in the case of a dimmed menu option, no associated processing takes place.[4]

---

[3]The user 'drags' the cursor by pressing the SELECT mouse button and, while the SELECT button is depressed, positioning the cursor in the desired position. As the cursor is moved over a menu option, the option is highlighted.

[4]This technique of de-emphasising unavailable menu options is a standard feature of most Apple Macintosh software and other third party software such as Microsoft Word for the Macintosh.

Pop-up menus operate differently to pull-down menus; the user presses a mouse button to view one menu that comprises all commands that may be issued at that point in time. Pop-up menus are displayed either next to the cursor or at a fixed location within the viewing area. Selection of a command from a pop-up menu operates similarly to selection of an option from a pull-down menu.

The MUSSAT user interface incorporates pull-down menus and one icon menu. Pull-down menus were chosen rather than pop-up menus for the following reasons:

(1) During some stages of interaction with MUSSAT the number of available commands is large and a pop-up menu may not be able to display all of these commands within the height of the screen, especially if the menu is displayed where the cursor is positioned and not at a fixed location near the top of the screen;

(2) MUSSAT often allows the user to issue commands that affect one of several types of objects during any one state. The use of pull-down menus allows commands to be grouped into logical sub-sets of commands associated with the type of object they operate on. In contrast, pop-up menus present all available commands in one menu. It is expected

that grouping commands into logical sub-sets will make it easier for the user to find and select the desired command;

(3) Command names can be shorter using pull-down menus since the menu title can be used to qualify the object(s) the command can operate on.

Shorter command names have the advantage of occupying less space on the screen, although command names must not be abbreviated to the extent that the meaning of the command becomes ambiguous or unclear; and

(4) Pull-down menus allow the user to maximise the use of muscle memory. Each command always occupies the same position within a menu and each menu always occupies the same position within the viewing area, hence users familiar with the interface are able to select the desired commands not only by name but also by physical location.

In contrast, the contents of a pop-up menu may not maintain the same contents or physical ordering since only the commands able to be executed while in a particular state are displayed. Hence the user many not be able to use muscle memory to the same extent to select frequently used commands.

For the purposes of this thesis, those commands within a pull-down menu that are valid and may be activated (those command names in bold type) at a particular point in time will be referred to as 'active' commands within a menu. The complementary set of commands which may not be activated (the dimmed commands), will be referred to as the 'unavailable' commands within a menu.

5.2.3. Icon Menu

An icon menu is used in MUSSAT to present the user with a graphical representation of each type of element that may be used in the construction of DFDs. An example of how this menu might look is shown in figure 5.1.

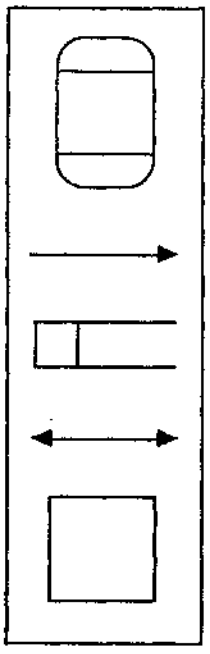


Figure 5.1 - The icon menu.

The icons represent (from top to bottom) a process, a one-way data flow, a data store, a two-way data flow and an external entity.

Unlike pull-down or pop-up menus, this menu is intended to remain visible, on the left hand side of the screen as long as a DFD is in an open state. The icon menu allows the user to construct DFDs by selecting the desired icon and specifying a position within the drawing area; MUSSAT then draws a duplicate of the icon at the selected position within the DFD.

Designing icons is not a trivial exercise ([GIT86], [MCC83], [LOD83]). The experimentation and research suggested in [LOD83] in order to produce icons designed for specific user groups is beyond the scope of this thesis, hence the shape and ordering of the icons in figure 5.1 is only a proposed menu design.

5.2.4. Dialogue Boxes

A dialogue box is a small window that appears in the viewing area in response to the user executing a command that requires parameters, confirmation, or a system response and hence is analogous to a command line in traditional systems.

The user is required to supply additional parameters for the successful execution of some commands within

MUSSAT. In most cases these parameters are text strings, such as the name of a DFD element or the name(s) of DFDs to print. In addition, the user is required to confirm the execution of operations that have potentially severe results, for example, deleting an existing MUSSAT project. The user is prompted for the input of command parameters and confirmation of potentially severe operations via dialogue boxes.

Three types of dialogue boxes will be used in MUSSAT:

(1) Confirmation dialogue boxes in which the user is requested to either confirm or cancel the execution of a previously selected command.

(2) Parameter specification dialogue boxes in which the user may specify a parameter required to execute a selected command.

(3) Error dialogue boxes in when an error message is displayed, the associated error condition of which is often caused by an error in the specification of a command parameter.

The use of each type of dialogue box is illustrated in the following examples of how the OPEN and DELETE commands from the DFD menu operate (each command is described in greater detail in Appendix V).

Figure 5.2a shows the dialogue box that would be displayed after the user has selected the OPEN command from the DFD menu.

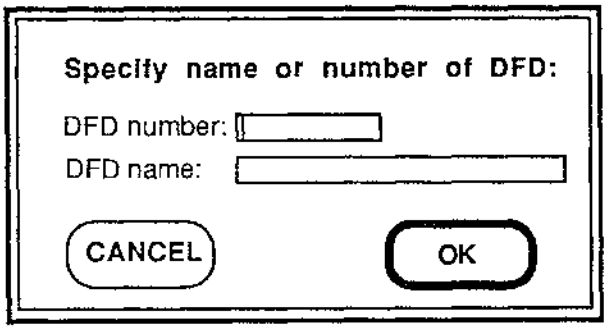


Figure 5.2a - Parameter specification dialogue box for the OPEN command.

The horizontal bar in the first empty rectangle in the dialogue box represents the cursor. Since the user is required to enter a text string the text entry cursor shape has replaced the select mode cursor.

The user may specify which DFD to open by either specifying the name or number of a DFD. If the user specifies a DFD that does not exist, then an error message will be displayed as shown in figure 5.2b.

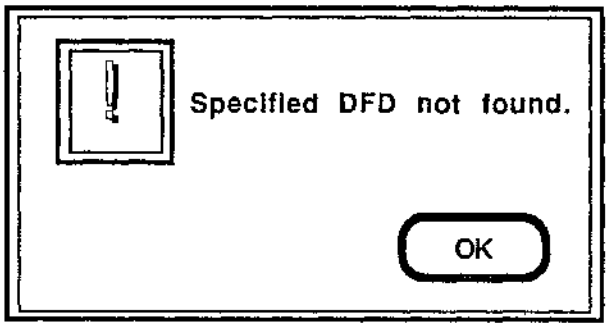


Figure 5.2b - Error dialogue box for the OPEN command.

The user must then select the OK soft button to remove the error dialogue box and return to the prompt box shown in figure 5.2a. Similar error dialogue boxes will be shown for other errors that may occur as the result of the values specified by the user in the parameter specification dialogue box.

Assuming that the user has specified the name or number of an existing DFD, then the DELETE command may be selected. Figure 5.2c shows the confirmation dialogue box that will be displayed as a result of selecting the DELETE command from the DFD menu.

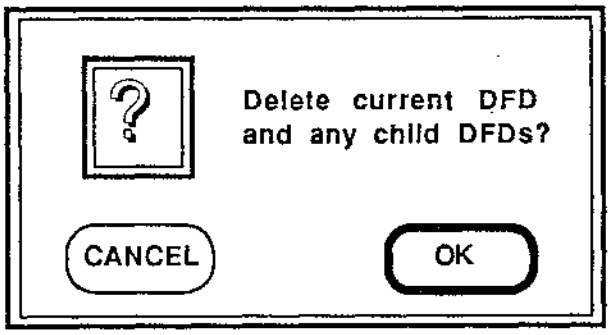


Figure 5.2c - Confirmation dialogue box for the DELETE command.

The user must select one of the two soft-buttons to remove the dialogue box from the screen. The OK button will cause the DELETE operation to be performed; the CANCEL button will terminate the DELETE command without deleting the DFD.

The parameter dialogue box shown in figure 5.2a requires the user to specify a text string. Another form of parameter dialogue box is used in MUSSAT: these require the user to select (rather than specify) the required parameter value.

Figure 5.2d shows the dialogue box that would be displayed as a result of selecting the DD DISPLAY command from the OPTIONS menu. The highlighted soft-button indicates the current value of the system parameter.

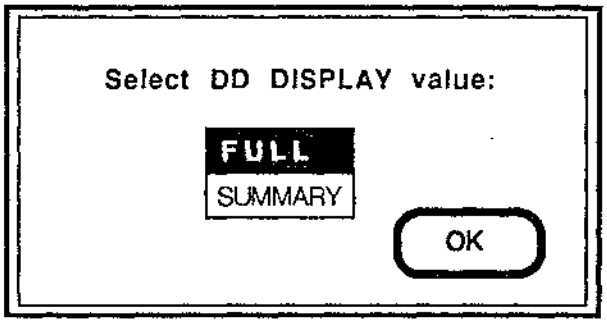


Figure 5.2d - Parameter specification dialogue box for the DD DISPLAY command.

The user may change the value of the system parameter by selecting the unselected value. The dialogue box is removed from the screen by selecting the OK button.

Keyboard input within parameter specification dialogue boxes is minimised (where possible) and in most cases the only parameters that the user must specify using a keyboard are object names and numbers. Where the user is required to select one of several options, each option is usually displayed within the dialogue box and the user selects the desired option with the mouse. In addition, soft buttons within the option box are used to either cancel or continue execution of the command.

In order to further reduce the amount of required user input, when a dialogue box is used as a confirmation dialogue (that is, the user is required to select between a CANCEL option and an OK option), the default option (which is usually the confirm option) will be

pre-selected and the user need only press the ENTER key on the keyboard in order to select the default response. To cancel the command requiring confirmation, the user selects the CANCEL soft-button in the dialogue box.

Dialogue boxes will not be required for conventional syntax or semantic errors caused by invalid command specification since the proposed implementation of pull-down menus does not allow the user to select inappropriate commands. The only error conditions that dialogue boxes will be used for will be system errors or invalid command parameter specification.

There are several reasons why option boxes rather than the traditional command line approach to parameter specification and confirmation have been incorporated in the MUSSAT user interface:

(4) Screen space is likely to be at a premium, especially when viewing DFDs. A command line may unnecessarily clutter the display area.

(5) A dialogue box minimises the amount of keyboard input required of the user since options need only be selected rather than specified.

(6) Some command parameters may be easier for the user to comprehend and specify in a graphical form, for

example, the number and layout of pages comprising a DFD.

A disadvantage of dialogue boxes is that the portion of the viewing area under the displayed dialogue box is obscured until the command that initiated display of the dialogue box is either executed or cancelled. It may be that the obscured portion of the screen contains data that the user needs in order to specify required parameters. Nevertheless, dialogue boxes are considered superior to the command line(s) approach for the MUSSAT user interface for the reasons outlined above.

Apart from example dialogue boxes used earlier in this section, the format of other dialogue boxes have not been specified in detail. A dialogue box will be required for each command that requires the user to specify parameters or confirmation. In addition each parameter specification dialogue box may have several associated error dialogue boxes.

The parameters that the user must specify for the successful execution of each command are identified in Appendix V. Commands that will require confirmation boxes are also identified in this section.

5.2.5. Windows

Windows in the MUSSAT interface incorporate scroll bars, a move bar and a close box to allow the user to manipulate the contents, placement and number of visible windows (respectively).

Figure 5.3 is an example window incorporating scroll bars, a move bar and a close box.

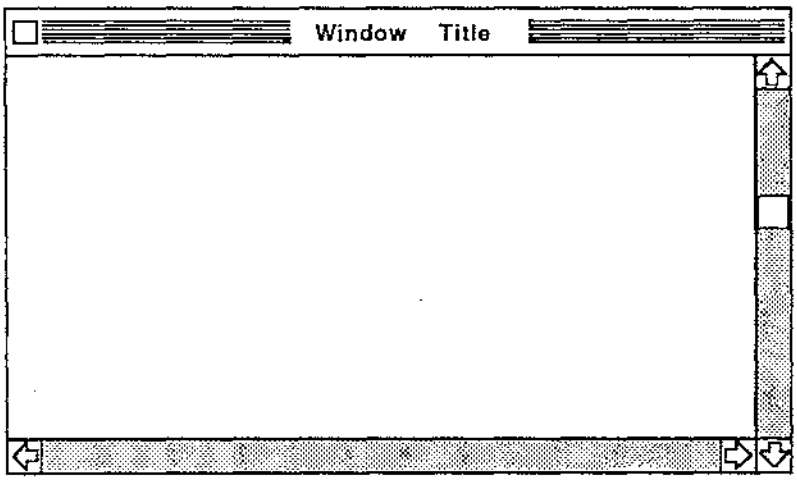


Figure 5.3 - An example window with close box, move bar, scroll box and scroll bars.

The purpose and use of each component of the window shown is discussed below:

Close Box

The close box is depicted as a small box in the upper left hand corner of a window, as shown in figure 5.3. To remove the window and its contents from the viewing area, the user selects the close box.

Selection of a close box in MUSSAT causes the contents of the window to be saved before the window is removed from the screen. If the contents of the window are in an inconsistent state, that is, unresolved errors exist in the displayed DFD or DD entry, then selection of the close box will result in a dialogue box with an appropriate message being displayed. The user is not able to close a window until any errors are resolved.

Move Bar

The move bar is the narrow horizontal bar at the top of the window and serves two purposes:

(1) To display the title of the document or diagram contained in the window. In MUSSAT, the title area of the menu bar will show the name of the DFD or DD entry that the window contains.

(2) To provide a means by which the user can position windows within the viewing area.

In MUSSAT the user will be able to move DD ENTRY windows within the current viewing area by selecting any portion of the move bar (including the window title) and dragging the window to the desired position. Since DD windows may obscure the underlying DD window the user may choose to overlap several DD windows with just the move bar showing.

The move bar in a DFD window will show the name, number and version number of the DFD.

MUSSAT does not allow the user to move DFD windows. To differentiate between those windows the user may move and those that must remain where positioned, movable windows will contain a series of horizontal lines in the menu bar (as does the window in figure 5.3). Windows that may not be moved (that is, DFD windows) will not contain these lines.

Scroll Bars

Each window may contain one or two scroll bars allowing the user to view a document or diagram that may be larger than the window viewing area. In figure 5.3 a window with two scroll bars is shown. The horizontal a scroll bar is the narrow bar at the bottom of the window with an arrowhead at each end. Similarly, the vertical menu bar is the narrow bar at the right hand side of the window.

When the view that a window shows is not the entire document or diagram, then a small box is shown in one or both of the menu bars to allow the user to change the view shown in the window. If the logical document is longer than the window viewing area, than a scroll box is shown in the vertical scroll bar, Similarly, if the document is wider than the window viewing are then

a scroll box is shown in the horizontal scroll bar. If the logical document is both longer and wider than the window viewing area, then a scroll box is shown in both menu bars.

The window in figure 5.3 has a small box in only the vertical menu bar, hence the underlying document is longer but not wider than the window viewing area. To change the view shown in the window, the user selects and drags the relevant scroll box to a new position within the scroll bar. For example, in figure 5.3 the user would move the scroll box in the vertical scroll bar up to view a portion of the underlying document that presently exists above that portion currently visible.

The width or height of a scroll bar represents the entire width or height (respectively) of the underlying document. Hence, movement of the scroll box to the top of the scroll bar in figure 5.3 will cause the window to display the top most portion of the underlying document, regardless of whether the document is one or five pages high.

In the MUSSAT interface a DFD window will have both a vertical and horizontal scroll bar, although scroll boxes will only be shown for DFDs that do not fit in one viewing area.

DD windows will only ever have a vertical scroll bar since the width of a DD entry is fixed. A scroll box will be shown in the vertical scroll bar when the length of the DD entry exceeds the length of the window viewing area.

Vertical scroll bars will also be used in dialogue boxes where the options presented will not fit within the viewing area of the dialogue box.

5.2.6. The Cursor

The normal MUSSAT cursor shape used to select objects and commands will be an arrow, as shown in figure 5.4a, however, in order to provide the user with feedback when a time-consuming process is taking place, an alternate cursor shape will replace the normal cursor.

Figure 5.4a - Normal MUSSAT cursor shape.

Two possible 'wait' cursor shapes are shown in figure 5.3b: an hourglass icon and a clock icon.

Figure 5.4b - Alternate 'wait' cursor shapes.

The normal cursor shape should be replaced with a 'wait' cursor shape as soon as a process that is likely to be time consuming is activated, and should be replaced with the normal cursor shape once the process is completed. The user should be able to move a 'wait' cursor around the screen as normal, but should not be able to perform any other manipulation of the display (such as viewing a pull-down menu) or select any other command (including scroll) until the 'wait' cursor is replaced with the normal MUSSAT cursor.

In addition to the 'wait' cursor shape two other cursor shapes will be used when MUSSAT is in 'draw' or 'text' mode. When specifying the location of a new DFD object, or the path of a data flow, the cursor shape should change since the user may not use the cursor in the select mode until the drawing operation is terminated. A pencil shaped cursor, as shown in figure 5.4c, is proposed as the 'draw' state cursor.

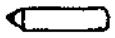


Figure 5.4c - Proposed 'draw' cursor shape.

An alternate cursor shape is also required when the user is expected to specify or edit a text string. The 'text' cursor shape is commonly shown as a blinking bar, or a reverse video rectangle at the point of text insertion. Either of these approaches would be suitable for MUSSAT.

5.2.7. Example Screen Formats

Figures 5.5a and 5.5b are mock-ups of the MUSSAT user interface incorporating the features discussed in this section.

In figure 5.5a the DFD menu title in the menu bar has been selected and the cursor has been positioned over the DELETE command.

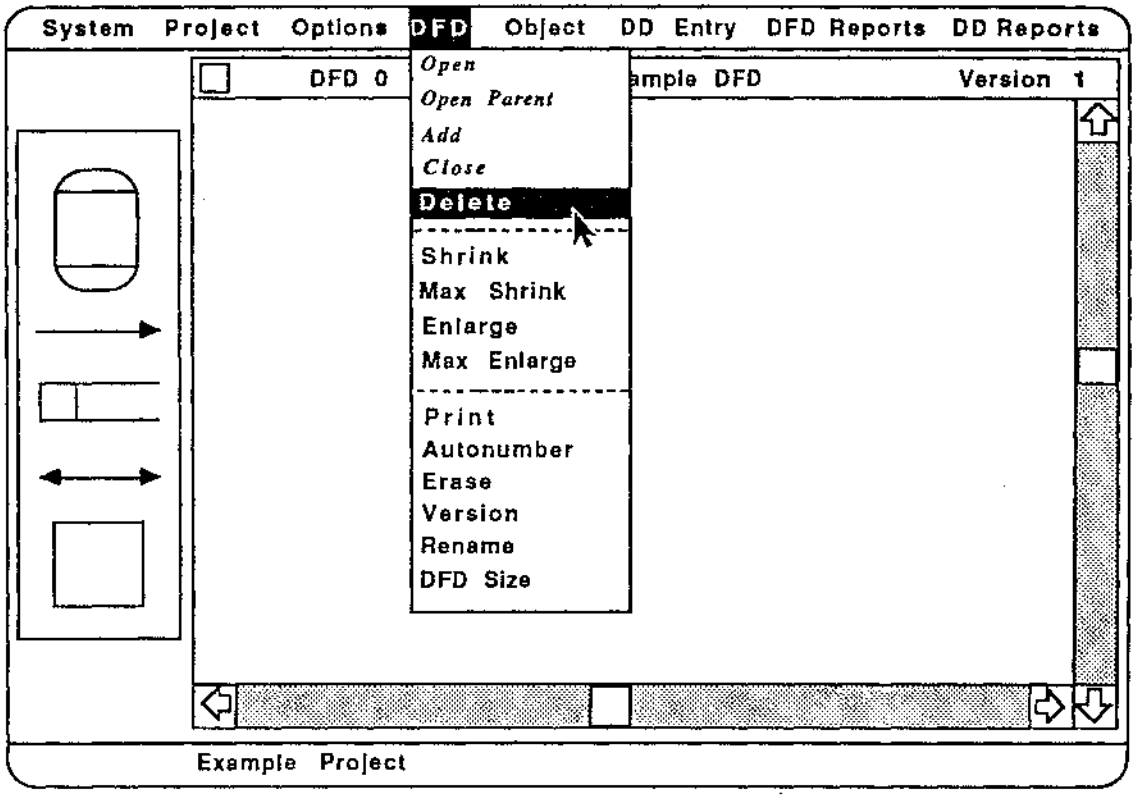


Figure 5.5a - MUSSAT screen display and pull-down menu format.

If the user were to release the SELECT mouse button, with the cursor positioned over the DELETE command, then the menu would disappear and the dialogue box shown in 5.5b would appear.

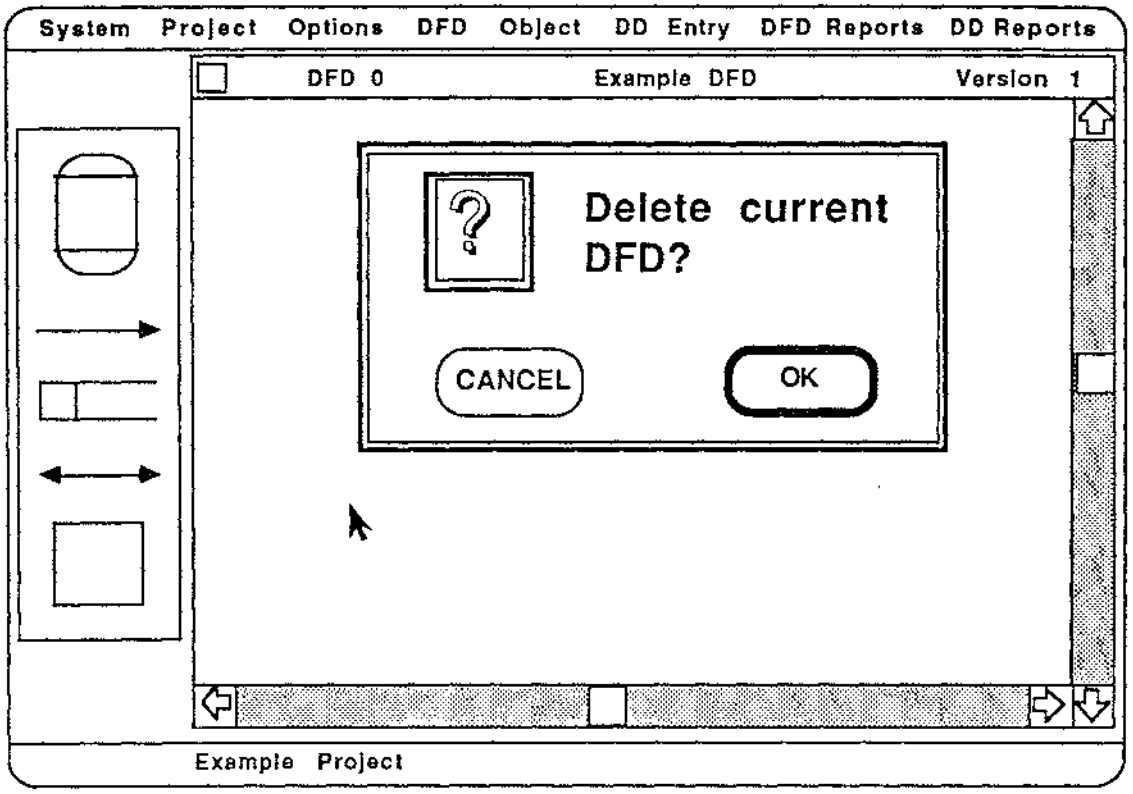


Figure 5.5b - MUSSAT screen display and dialogue box format.

The user would then have to select one of the soft-buttons at the bottom of the dialogue box, or press the ENTER key in order to continue.

An additional feature shown in figures 5.4 and 5.5 but not discussed in the preceding section, is the PROJECT line at the bottom of the display. The PROJECT line always contains the name of the MUSSAT project that the user is working with. The PROJECT line will be blank if the user has not selected a project to work with.

State Transition Diagrams [BIE83] were used to describe which commands, including those available in the pull-down

menus, are valid at any point in time while using MUSSAT. Appendix III consists of an explanation of the State Transition Diagram notation used, the State Transition Diagrams, and examples of the MUSSAT pull-down menus.

Each of the MUSSAT commands shown in the State Transition Diagrams in Appendix III, when executed will cause certain processing to be performed. Most of the MUSSAT commands will affect both the user's view of the system, as seen on the VDU, and the underlying data model. Appendix IV contains a high level description of the affects of each MUSSAT command introduced in Appendix III on both the user interface and the data model. Diagrams are used to help explain the more complex commands.

Structured Systems Analysis ([DEM78], [GAN80]) was used to develop a detailed model of the data and processes required to carry out the activities described in Appendix IV. The resulting functional model, which is included as Appendix VI, consists of the following components:

a set of Data Flow Diagrams;
a Data Dictionary;
a Data Model Diagram; and
examples of each of the reports produced by MUSSAT.

Appendix VI contains a cross-reference of MUSSAT command names and Data Flow Diagram processes. There is not always

a one to one correspondence between a command and the processes that must be activated in the Structured Systems Analysis model in order to perform the command. Hence, a cross-reference was created to shown which process need to be activated in response to the user issuing a MUSSAT command.

Appendix VII consists of a discussion of some of the more complex features of the MUSSAT design and Appendix VIII describes several improvements to the design that could be made to enhance the functionality of MUSSAT.

CHAPTER 6

Structured Systems Analysis in Automated Environments

The chapter introduces the concept of CASE tools and describes the similarities between MUSSAT and other CASE tools, and also describes the differences in the underlying philosophy of CASE technologies and MUSSAT. The second part of this chapter gives brief overview of two existing CASE tools that support SSA.

6.1. The Place of MUSSAT in CASE Technologies

When research work on this thesis was started, in 1986, there were relatively few automated design tools such as MUSSAT available in the marketplace, and even fewer available in New Zealand. Since then, the number and sophistication of such tools has risen dramatically.

The acronym: CASE (Computer Aided Software Engineering), refers to all automated design tools that help a computer professional design, implement and maintain information systems. CASE tools currently available vary widely in the type of assistance they provide. Some CASE tools address a large portion of the system lifecycle, others concentrate on one or two phases. Still other tools provide support in strategic planning and project management.

Existing CASE tools also vary in the design methodologies they support. System analysis and design methodologies currently supported by CASE tools include: SSA ([DEM78] and [GAN80]), ISAC [LUN81], and Information Engineering [MAR81].

McClure has developed a system of classifying CASE tools [MCC87] using the following three categories:

CASE Toolkits;

CASE Workbenches; and

CASE Methodology Companions.

CASE products in the Toolkit category provide automated assistance in one or more phases of the system lifecycle, such as analysis and design or code production, or assistance in the development of a particular type of system such as real-time systems.

CASE Workbenches provide support for the entire software life cycle. The tools provided in a CASE Workbench are designed to work together and hence the outputs of one phase of the life cycle are automatically available to tools used during the following phases. The final output produced using a CASE Workbench is an executable software system.

CASE Methodology Companions consist of a set of tools that automate a particular software development methodology. The user of a Methodology Companion is guided through the steps of the methodology and, in some cases, may not be allowed to

proceed to the next phases if errors or inconsistencies exist in any of the documentation developed in the preceding phase.

The ultimate goal of CASE technologies is to generate production systems directly from system specifications.

Using McClure's categories, MUSSAT would be classed as a CASE Toolkit, and more specifically, an Analyst Toolkit since the outputs produced using MUSSAT are part of the requirements specification produced during the systems analysis phase of the software lifecycle. Although MUSSAT addresses only one phase of the software lifecycle, when many of the existing CASE tools provide wider coverage, it must be remembered that the objective of the design exercise documented in this thesis was to provide automated support for SSA, not the entire system lifecycle. The development of automated support for the whole lifecycle is an enormous undertaking which require hundreds of work years of effort. Hence, although MUSSAT may be described as a CASE tool, MUSSAT and the general CASE philosophy are dissimilar.

Nevertheless, it is interesting to compare the user interface incorporated in MUSSAT with the interfaces of similar CASE tools.

Although there are a considerable number of CASE tools

available that support SSA[1] , this writer has only had first-hand experience of three such tools: Excelerator from InTech Technologies and PCSA and Teamwork/SA from Cadre Technologies.[2] Of these two tools, the writer is most familiar with Excelerator, having drawn the DFDs and Data Model Diagrams in Appendix V using this tool.

6.2. Two CASE Tools

The following sub-sections provide an overview of the user interface of Teamwork/SA and Excelerator, both of which are available in New Zealand.

Excelerator

Excelerator was one of the first CASE tools available in New Zealand and is used in several large DP installations in the country including the Reserve Bank of New Zealand. It offers facilities to maintain a data dictionary and draw the following types of diagrams:

Data Flow Diagrams;

Data Model Diagrams;

Structure Charts; and

---

[1]A number of these tools are listed in Chapter 1.

[2]A demonstration version of PCSA was purchased for evaluation, however PCSA was later acquired by Cadre Technologies and is now the PC version of their Teamwork tools. This writer was able to examine Teamwork/SA some time after the demonstration version of PCSA was purchased. Both PCSA and Teamwork/SA have similar user interfaces and although Teamwork/SA is discussed, the comments made are equally applicable to Teamwork/PCSA.

free-form Presentation Graphs.

Both the DeMarco and Gane & Sarson DFD conventions are supported; the system manager may select which format will be used by all other users of the system. Various forms of Data Model Diagrams are similarly supported.

Excelerator uses a two-button mouse: the left-hand button serves as the SELECT command, and the right-hand button corresponds to the CANCEL command. The style of interaction is prefix oriented in contrast to MUSSAT's postfix interface.

The Excelerator interface is menu based with one letter keyboard equivalents available for most commands. The user may select menu options by either entering the keyboard mnemonic or by selecting the menu option with the mouse. In cases where the user is required to enter the name of a diagram, a wildcard character may be used which will cause a list of all diagram names matching the specified string to be displayed. The user may then select the desired diagram from the list.

Multiple windows are not used, hence the user may only view one DFD or DD entry at a time. If the user wishes to view a DD entry while a DFD is open, then the DFD must be closed and the DD subsequently opened.

When manipulating a DFD, the user selects commands from a menu permanently visible down the left-hand side of the screen. The menu consists of two parts: the upper portion which contains the general DFD manipulation commands, and a scrollable region at the bottom of the menu panel which serves as a sub-menu when a command that has further sub-options is selected.

The bottom line of the screen is used as a message line in which error messages, prompt messages and confirmation requests are displayed.

A DFD may be viewed at one of three zoom levels. The user may also select which portion of the DFD to view, when the DFD is larger than the viewing area, by selecting a position in the navigation window. The navigation window is a small region, approximately three centimetres square, that appears below the menu panel and contains a representation of the DFD visible in the DFD viewing area. Individual DFD elements are represented in the navigation window as dots. The user can change the portion of the DFD visible in the DFD viewing area by selecting a point within the navigation window that will serve as the centre of the DFD viewing area.

The user creates DFD elements by selecting the appropriate element type from the menu panel and then

selecting a position in the DFD viewing area. The object is drawn once a valid position is selected; objects other than data flows may not overlap any other object.

A command in the menu panel remains selected until another command is selected or the CANCEL mouse button is pressed, hence a user is able to repeatedly draw or delete DFD elements without having to re-select the appropriate command for each DFD element.

Two system options are available that allow the user to tailor how a data flow is drawn:

> The user may specify that Excelerator should decide where, on the perimeter of the source and destination objects, a data flow will be attached to. Alternately, the user may specify that Excelerator should allow the user to select the position to attach the data flow to for each source and destination object.
>
> The second option allows the user to specify whether Excelerator should automatically determine the path a data flow will follow, given two end points, or whether the user will select intermediate points for the data flow path.

It is this writer's experience that data flows drawn using Excelerator's automatic data flow drawing algorithm often result in unnecessary data flow intersections and turning points. In addition, data flows are sometimes drawn intersecting processes, data stores or external entities. Even when selecting two objects that appear close together and in the same x or y-plane, the resulting data flow may still contain unnecessary 'kinks'.

Excelerator does not offer an equivalent of the MUSSAT 'COPY' command to create duplicates of data flows. Nor does Excelerator provide a full equivalent of the MUSSAT 'REDRAW' command to respecify the source/destination points or the path of a data flow; using Excelerator, the user may respecify the point on the source or destination object that a data flow is connected, but can only change the source or destination element of a data flow by deleting and redrawing the data flow.

Excelerator does not use 'dragging' to reposition existing DFD elements: instead the user selects the MOVE command from the menu panel, selects the object to be moved, and then selects a new position. If the selected object is not a data flow, then all connected data flows are 'rubber-banded', although, as with drawing new data flows, the 'rubber-banded' data flows can

follow unusual paths.

When selecting end points for data flows, Excelerator does not check to ensure whether a valid type of data flow end point has been selected. For example, a data flow can be drawn between two data stores, or with the same element serving as both the source and destination element. The user may, however, request a report that lists all such inconsistencies for a specified set of DFDs.

The user is able to enter a description of each DFD element and subordinate data item entries into a DD, however some of the fields included in the template DD entry are not relevant to the analysis stage. For example, the template includes a field that allows the user to specify a language dependant representation of a data flow or data item. In addition, the Excelerator DD does not support either the Gane & Sarson [GAN80] or DeMarco [DEM78] data definition languages.

Excelerator allows a user to maintain a DD and draw several types of diagrams, including DFDs and hence helps reduce the clerical overhead involved in using SSA. Excelerator also is able to detect balancing and DFD construction rule violations, although in the latter case, errors are detected in a batch mode rather than in real-time. Although Excelerator provides some

useful features, this writer believes that other CASE tools provide better support for SSA and incorporate user interfaces that are easier to use.

Teamwork/SA

Teamwork/SA is one of a number of CASE tools available from Cadre Technologies. The entire Teamwork environment consists of the following tools:

Teamwork/SA;
Teamwork/SD;
Teamwork/RT;
Teamwork/IM; and
Teamwork/Access.

Teamwork/SA is the systems analysis package; Teamwork/RT is used in conjunction with Teamwork/SA to model real-time systems. Teamwork/SD addresses the systems design portion of the software lifecycle and incorporates the Structured Design methodology. Teamwork/IM is the information modelling module. The final member of the Teamwork product family, Teamwork/Access, allows the other Teamwork products listed above to be used in a workbench environment.

The user interface incorporated in Teamwork/SA requires a MicroSoft compatible mouse and incorporates multiple windows, pop-up menus, dialogue boxes, and selector

lists. Teamwork/SA supports the DeMarco version of SSA.

The user may view multiple DFDs and DD entries at any one time. The window formats used in the Teamwork tools incorporate the same features as Apple Macintosh windows, and hence are similar to MUSSAT windows.

Most Teamwork commands are activated using pop-up menus. Three types of pop-up menus are used [PER87]:

a blank space menu;
a global command menu; and
10 object oriented menus.

As in MUSSAT, inactive commands are de-emphasised in the menu, and selection of a de-emphasised menu option has no effect. A menu is displayed in response to pressing a mouse button; the particular menu displayed depends on where the cursor is positioned when the mouse button is pressed.

Process, external entities and file DFD elements are created by positioning the cursor at the desired position, pressing the mouse button and then selecting the object type from the resulting pop-up menu.

Data flows are drawn by positioning the cursor on either the source or destination object, pressing the

mouse button and then selecting either the SOURCE or DESTINATION command option, depending on the role the selected element will assume. A 'rubber-band' line then is attached to the cursor and may be dragged to another point. The user may specify the path of the data flow using other commands. The data flow is terminated when a valid end point is selected. Data flow line segments are initially drawn as straight lines, but additional commands allow the user to instruct Teamwork/SA to redraw one, or, all data flows in the DFD with curved lines.

Data flows can be moved by dragging individual line segments of the data flow. Other DFD elements are also moved by dragging, and any connected data flows are 'rubber-banded'.

As with MUSSAT, Teamwork/SA enforces DFD drawing conventions at the time the DFD is drawn. For example, where Excelerator would allow a user to draw a data flow between two data stores, Teamwork/SA will dislay an error dialogue box if the user selects an illegal end point for a data flow.

Teamwork/SA also allows the user to move portions of one DFD into another DFD. This feature is likely to be useful in repartitioning existing DFDs.

Teamwork/SA also provides assistance in balancing between parent process and child DFD when drawing the child DFD. Flows into and out of the parent process are shown on the left and right-hand sides (respectively) of the child DFD drawing area and additional data flow commands are used to include these data flows in the child DFD.

The writer's experience with Teamwork/SA was limited to manipulating DFDs and hence no comments can be made about the underlying DD.

Teamwork/SA provides automated support for SSA and incorporates a state-of-the-art user interface incorporating many of the features that are associated with 'user-friendly' systems. The tool is based closely on Demarco's version of SSA and provides sophisticated features that assists the analyst in creating, refining, and maintaining a set of DFDs and a DD. Teamwork/SA also can also be used in conjunction with further tools in the Teamwork family to support other phases of the system life cycle. Although only having had a limited exposure to Teamwork/SA, this writer prefers Teamwork/SA to Excelerator for its sophisticated user interface and in-depth support of the SSA methodology.

The two tools presented in this section illustrate some of the features and capabilities of automated SSA tools currently available. CASE Technologies represent a new type of software product; up until a few years ago systems analysts and designers had very few automated tools specifically designed to assist in the development of software systems. The number and quality of CASE tools is expected to increase significantly within the next few years with more powerful, easy to use products becoming available at a reasonable cost.

CHAPTER 7

Concluding Remarks

Structured Systems Analysis is a widely used systems analysis methodology, probably the most commonly used analysis methodology in New Zealand. Even though SSA has its weaknesses, the methodology provides a set of tools and techniques that all parties involved in the development of a system, especially users, can understand and use. The usefulness of data flow diagrams lies partly in the ability to hide details of the data transformations in lower level data flow diagrams, and partly in the flexibility of what a data flow diagram can include.

Although SSA is easy to use and produces documentation that is easy to read, there are some problems with using the methodology in a manual environment, in particular:

> the high clerical overhead required to keep the DD and DFDs consistent; and
>
> the difficulty in determining the scope of effect of changes to any particular DD item.

Earlier chapters of this thesis identified the activities within SSA for which the users required automated support. A design was then presented for MUSSAT, a software tool which provides automated support for SSA. The resulting

functional specification of MUSSAT shows that automation of many aspects of SSA in a manual environment is possible, although not a trivial exercise. MUSSAT incorporates functions that are expected to significantly reduce, if not eliminate, those problems of SSA in a manual environment listed above.

Because MUSSAT automatically updates the DD in response to changes made to DFDs, the clerical overheads inherent in using SSA in a manual environment are not present in MUSSAT. In addition, the user is not allow to make changes to either the DD or DFDs that break the SSA rules, or compromise the integrity of the DD. MUSSAT also provides an automatic cross-referencing facility so that a user can easily determine the scope of effect of changes to any particular DD item.

MUSSAT is expected to significantly decrease the amount of time required to draw and maintain a set of DFDs. Time savings are expected to be greatest when amending existing DFDs since, unlike SSA in a manual environment, the user is not required to redraw the entire DFD. It is likely that MUSSAT will also allow a user to draw a new DFD more quickly than in a manual environment since the user is not required to draw each shape, only to specify a position or data flow path.

The reports produced by MUSSAT provide the user with details that assist in detection of errors and inconsistencies. A detailed DFD balancing report can be produced which checks a specified hierarchy of DFDs for imbalances between each parent process and child DFD in the hierarchy.

The concept of a balanced set of DFDs, where the net flows into and out of a DFD are equivalent to the flows into and out of the parent process, is fundamental to the SSA methodology. Hence, the ability to detect and correct balancing errors in a set of DFDs is important. However, checking a set of DFDs for balancing errors manually can be tedious and error prone. The DFD balancing function provided in MUSSAT automates the detection of balancing errors and produces a report which identifies data flows that do not balance according to the SSA rules. The user may then decide which of the data flows represent balancing errors and which represent trivial error flows which need not balance.

Although it would have been possible to enforce the DFD balancing rules by not allowing the user to create any further child DFDs, or by not allowing the user to exit the system while balancing errors remained, it was decided that such an approach would not be user friendly. In addition, there are many reasons why the user may wish to leave the a DFD unbalanced; not all imbalances between a parent process and child DFD are error conditions. For example, minor error flows need not balance.

Other MUSSAT reports allow the user to review DFD statistics, to determine which DFDs are incomplete, and which entries in the DD are not used. Further MUSSAT reports allow the user to produce printed copies of DFDs and the DD.

Computer Aided Software Engineering (CASE) tools exist that support the specification of systems using SSA. MUSSAT can be considered to be a CASE tool since it provides automated support for part of the systems development life cycle.

The first SSA CASE tools were essentially drawing tools. Later tools incorporated more sophisticated checking rules and a data dictionary. Many of the latest CASE products offer a family of tools to support significant portions of the systems development lifecycle.

Most CASE tools currently available address sub-sets of the system lifecycle phases, although some CASE tool suppliers claim that their products cover the entire lifecycle. CASE tools that are claimed to support the full lifecycle tend to provide stronger support in either the specification and design phases or in the detailed design and code generation phases.

Although many CASE tools are too expensive for small DP budgets, several PC-based CASE tools are available [PER87], making automated support for SSA economically feasible for almost all organisations. In addition, some of these low-cost tools can interface with other CASE tools that address

different phases of the system lifecycle.

Most CASE tools supporting SSA have incorporated either the DeMarco [DEM78] or Gane & Sarson [GAN80] variants, although a small number of CASE tools also offer support for real-time SSA [WAR85]. Some CASE tools also provide support for screen and report design, although, to this writer's knowledge, there are no existing CASE tools that have modified the SSA methodology to incorporate design of the user interface.

Existing CASE tools that provide automated support for SSA help overcome some of the weaknesses of SSA by including rigorous consistency and completeness checks, and by providing system design tools that help bridge the gap between analysis and design. In addition, CASE tools incorporating real-time SSA provide some support for modelling a class of systems that the original SSA methodology did not address.

Although MUSSAT (and existing CASE tools that support SSA) are expected to make SSA easier to use, there are still areas of the methodology that could be improved and extended to make it more applicable to types of systems that SSA is currently unable to model.

A SSA specification is not a complete functional requirements specification since the system model produced does not include a model of the user interface and preliminary screen and report formats. Further research is required to

develop a form of SSA that produces a full functional specification, including these features.

The variants of SSA described by DeMarco and Gane & Sarson are best suited to business-type applications with relatively simple user interfaces. However, the declining cost and increasing performance of hardware within the past five years has made high quality graphical interfaces economically feasible for almost all types of computer systems. The SSA methodology itself, has yet to incorporate tools and techniques to model such interfaces. Although DeMarco suggests that DFDs should be used to model the user interface, this writer believes that State Transition Diagrams are a more appropriate modelling tool for almost all user interfaces. A modified form of State Transition Diagrams were developed and used to model parts of the MUSSAT user interface.

In addition, the original SSA variants ([DEM78] and [GAN80]) are not suitable for modelling real-time systems or parts of systems where timing considerations are important. Although, Ward and Mellor [WAR84] have developed a modified form of SSA tailored especially to the needs of real-time systems.

The flexibility of data flow diagram representation can also be a disadvantage since the SSA rules presented by DeMarco [DEM78] and Gane & Sarson [GAN80] must often be tailored in

order to represent characteristics of a system that are not explicitly covered in the general SSA rules. For example, several data flow naming conventions were developed during the design of MUSSAT in order to depict characteristics of the system that were considered to be important but were not addressed by either DeMarco [DEM78] or Gane & Sarson [GAN80].

Although the ability to tailor the methodology to suit the needs of a particular development project or organisation is attractive, this writer believes that a detailed set of SSA rules should be developed that covers all aspects of DFD development. This form of SSA could then serve as a standard, removing the need for organisations to define their own versions of SSA.

Like other systems analysis methodologies, SSA has its weaknesses. Nevertheless, SSA is superior to other systems analysis methodologies in some aspects because the functional specification produced using SSA is a good communication tool between analysts, users and designers.

MUSSAT and other CASE tools that support the SSA methodology are able to reduce, if not remove, many of the difficulties inherent in SSA used in manual environments. However, this writer believes that the SSA methodology itself needs to be revised and further specified to make the methodology more useful for defining on-line, interactive systems.

References

[BIE83] "Flowcharting Revisited." New Zealand 7th Annual Computer Conference. (Wellington, N.Z., 1983), pp.123-139.

[BOE81] Boehm, Barry W. Software Engineering Economics. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.

[CAR82] Card, Stuart K. "User Perceptual Mechanisms in the Search of Computer Command Menus". Eight Short Papers in User Psychology. Edited by Thomas P. Moran. Cognitive & Instructional Sciences Series, Ca.: Xerox Corporation, 1982.

[CHE86] Cherry, Joan M. "An Experimental Evaluation of Prefix and Postfix Notation in Command Language Syntax." International Journal of Man-Machine Studies , 24: 1986, pp. 365-374.

[CON82] Connor, Michael F. "Structured Analysis and Design Techniques." Systems Analysis and Design: A Foundation for the 1980's. Edited by William W. Cotterman (et al). New York: North Holland Publishing Co., Inc., 1981, pp.213-234.

[DAV83] Davis, Willilam S. Systems Analysis and Design: A Structured Approach. Reading, Mass., Addison-Wesley Publishing Co., 1983.

[DEL82] Delisie, Norman M.; Menicosy, David E. and Kerth, Normal L. "Tools for Supporting Structured Analysis." Automated Tools for Information Systems Design. Edited by H. -J. Schneider and A. I Wasserman. Amsterdam: North-Holland Publishing Co., 1982.

[DEM78] Structured Analysis and System Specification. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1978.

[DOC86] Docker, Thomas W. and Tate, Graham. Executable Data Flow Diagrams Massey Computer Science Report 86/1. Palmerston North, N.Z.: Massey University, 1986.

[EMB81] Embly, David W. and Nagy, George. "Behavioral Aspects of Text Editors." Computing Surveys , 13: 1 1986, pp.33-70.

[FOL82] Foley, James D. and Van dam, Andries. Fundamentals of Interactive Computer Graphics. The Systems Programming Series. Reading, Mass., : Addison-Wesley Pub. Co., 1982.

[GAI85] Gait, Jason. "An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows." IEEE Transactions on Software Engineering. 8: 1985, pp. 714-717.

[GAN80] Gane, Chris and Sarson, Trish. Structured Systems Analysis: Tools and Techniques. Missouri: Improved System Technologies Inc., 1980.

[GIT86] Gittins, David "Icon-Based Human-Computer Interaction." International Journal of Man-Machine Studies , 24: 1986, pp. 519-543.

[HAN71] Hansen, Wilfred J. "User Engineering Principles for Interactive Systems." Interactive Programming Environments. Edited by David R Barstow (et al). New York: McGrah-Hill Book Company, n.y.

[HEM82] Hemenway, Kathleen. "Psychological Issues in the Use of Icons in Command Menus." Eight Short Papers in User Psychology. Edited by Thomas P. Moran. Cognitive & Instructional Sciences Series, Ca.: Xerox Corporation, 1982.

[JAC83] System Development. Englewood Cliffs, N.J.: Prentice-Hall International, 1983.

[LOD83] Lodding, Kenneth W. "Iconic Interfacing." IEE Computer Graphics and Applications. 3, No. 2: 1983, pp. 11-20.

[LUN81] Lundeberg, Mats: Goldkuhl, Goran and Nilsson, Anders. Information Systems Development , Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.

[MAR81] Martin, James and Finkelstein, Clive. Information Engineering Volume I. Lancashire, England: Savant Research Studies, 1981.

[MAR87] Martin, James Recommended Diagramming Standards for Analysts and Programmers. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1987.

[MCC83] McCleary, George F. Jr. "An Effective Graphic 'Vocabulary'." IEEE Computer Graphics and Applications, 3, No. 2: 1983, pp. 46-53.

[MCC87] McClure, Carma. Computer-Aided Software Engineering Presentation. Sydney, Australia: Digital Consulting, Inc., 1987.

[MOR81] Moran, Thomas P. "An Applied Psychology of the User." Computing Surveys , 13, No. 1: 1981, pp. 1-11.

[NOM86] Norman, Kent L.; Weldon, Linda J. and Shneiderman, Ben. "Cognitive Latouts of Windows and Multiple Screens for User Interfaces." International Journal of Man-Machine Studies , 25:1986, p. 229-248.

[PAR86] Parnas, David Lorge and Clements, Paul C. "A Rational Design Process: How and Why to Fake It." IEEE Transactions on Software Engineering , SE-12, No. 2: 1986, pp. 251-257.

[PER87] Perrone, Giovanni. "Low-cost CASE: Tomorrow's Promise Emerging Today." Computer , 20, No. 11: 1987, pp. 104-110.

[POW84] Powers, Michael J.; Adams, David R. and Mills, Harlan D. Computer Information Systems Development: Analysis and Design. Cincinnati, Ohio: South-Western Publishing Co., 1984.

[ROS82] Rosenberg, Jarret. "Evaluating the Suggestiveness of Command Names." Eight Short Papers in User Psychology. Edited by Thomas P. Moran. Cognitive & Instructional Sciences Series, Ca.: Xerox Corporation, 1982.

[SHN80] Shneiderman, Ben. Software Psychology. Cambridge, Mass.: Winthrop Publishers, Inc., 1980.

[TAT85] Tate, Graham and Docker, Thomas W. G. "A Rapid Prototyping System Based on Data Flow Principles." Massey Computer Science Report 85/3, Palmerston North, New Zealand: Massey University, 1985.

[TAT86] Tate, Graham and Hayward, Richard. Information Systems Strategy - Functional Versus Data Analysis. Unpublished Paper.

[TEA85] Teague, Lavette C. Jr. and Pidgeon, Christopher W. Structured Analysis Methods for Computer Information Systems. Chicago: Science Research Associates Inc., 1985.

[TEO82] Teorey, Toby J. and Fry, James P. Design of Database Structures. Prentice-Hall Software Series. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.,

1982.

[WAR84] Ward, Paul T. and Mellor, S.J. Structured Development for Real-Time Systems. New York: Yourdan Press, 1985.

[WIE84] Wiener, Richard and Sincovec, Richard. Software Engineering with Module-2 and Ada , New York: John Wiley & Sons, 1985.

Volume II

Appendix I

CASE Tool Suppliers

Abvent
9903 Santa Monica Blvd/Suite 268
Beverly Hills
CA 90212
USA

Cadre Technologies Inc.
222 Richmond Street
Providence
RI 02903
USA

DeLandgraff Consultants[1]
P.O. Box 79201
Royal Heights
Auckland
New Zealand

Index Technologies Corp.
Five Cambridge Centre
Cambridge
MA 02142
USA

McDonnell Douglas Computer Systems
400 MacArthur Blvd
Newport Beach
CA 92660
USA

Nastec Corp.
24681 Northwestern Highway
Southfield
MI 48075
USA

StructSoft Inc.
24 Homer Street
Parsippany
NJ 07054
USA

[1]New Zealand distrubutor for Interprogram, The Netherlands.

Appendix II

SSA Rules Used in MUSSAT

The SSA rules outlined in this appendix are described more fully in section 4 of this thesis.

1. Data flow diagrams

DFD size

(1) Most DFDs should fit on one A4 sheet of paper, however, larger DFDs, consisting of any number of A4 pages (arranged in a rectangular shape) may be drawn.

(2) The number of DFD elements allowed in one DFD is not limited, although data stores, external entities and processes may not overlap and data flows must be a certain minimum length.

DFD construction

(1) A process must be named but not necessarily numbered before it is exploded into a lower level DFD.

(2) Processes may be numbered at any stage of DFD development.

(3) Any DFD element may be drawn (fully or partially) outside a DFD boundary. Such notation has no associated semantics, but serves as a notational tool for showing the net sources or sinks of data (or net flows in or out) in a child DFD.

(4) DFDs need not be specified in a strictly top-down fashion, that is, a child DFD may be specified before the parent DFD is drawn and the relationship between the two explicitly specified later.

(5) DFD element names may consist of any combination of printable characters, although the DeMarco relational operator '+' has a special meaning in data flow names. Nevertheless, it is good practice to avoid using the other relational operators: [,], |, {, and } in DFD element names to avoid confusion.

(6) DFD element names are not case sensitive, that is, CUSTOMER-NAME, Customer-Name and customer-name all identify the same DFD element.

(7) A DFD elements must be named before further user defined fields may be specified in the DD.

(8) Any unnamed DFD elements, except for unnamed data flows to or from data stores, are ignored in system generated reports.

Data stores and external entities

(1) Data stores, external entites and data flows may be duplicated, that is, multiple copies of a DFD element with the same name may exist.

(2) All occurrences of a duplicated data store, external entity or data flow share the same DD entry.

(3) Duplicates of data stores and external entities are not marked with horizontal or diagonal lines (respectively).

(4) An external entities does not have a lower case identifier in its upper left hand corner.

(5) Data stores are drawn as narrow rectangles open at the right end.

(6) Data stores may be numbered with any short string of characters.

(7) The default numbering system for data stores is that used by G & S.

(8) Local data stores are, by default, numbered in the same way as non-local data stores.

Processes

(1) Process bubbles are drawn as vertically oriented rounded rectangles.

(2) Process bubbles include an optional comment section in the lower section of the bubble which may include any combination of printable characters. The number of characters allowed is implementation dependent.

(3) The top portion of a process bubble shows the last digit of the process number, that is, a digit unique within the DFD in which the process resides.

Data flows

(1) Data flows are drawn as horizontal and vertical lines joined with rounded corners and with an arrowhead at one or both ends.

(2) Unnamed data flows between external entities, processes or other external entities are ignored in system generted reports; such data flows are considered to be incompleteley specified.

(3) For the purposes of system generated reports, single and double headed unnamed data flows to or from a data store are interpreted as containing the entire contents of the data store.

(4) Retrieval keys may optionaly be specified for single headed named data flows from a data store. If specified, the retrieval key is stored in the DD.

All occurrences of the same data flow will have the same retrieval key.

(5) Net data flows in to and out of a child DFD need only have one end-point specified on the child DFD, although the source or destination DFD element shown in the parent DFD may be duplicated in the child DFD. If the source or destination element is redrawn in the child DFD then MUSSAT will still recognise the data flow as a net flow into or out of the child DFD.

(6) All data flows must be connected to a process at either one, or both end-points.

(7) If two data flows intersect, one of the data flows should be drawn with the 'little hoop' convention at the point of intersection.

(8) Data flow names may consist or two or more data flow names, each separated by a '+', in which case each component of the compound name should be defined in a separate data flow definition in the DD.

2. Data Dictionary

(1) The DD consists of seven types of entries:

data element

data store

data structure

data flow

process

sub-process

external entity.

(2) The following information, for each type of DD entry, is stored in the DD:

data element

    name
    alias entry
aliases
    notes
    definition - values and meanings
aliases.

data structure

    name
    alias entry
    aliases
    notes
    definition
    volume.

data flows

    name
    alias entry
    aliases
    notes
    definition
    volume
    type.

data stores

name
    alias entry
    aliases
    notes

definition
number
organisation.

processes

name
notes
definition
type
number.

sub-process

name
notes
definition
type.

external entities

name
notes
data flows in and out.

(3) The DeMarco relational operators are used in a data item description language for the definition of data stores, data flows and data structues.

(4) Processes and data elements are defined in free-form text.

Appendix III

MUSSAT State Transition Diagrams

Figures III.2 to III.6 are State Transition Diagrams (STDs) representing the MUSSAT user interface. The form of State Transition Diagrams (STDs) is based on a notation described in [BIE83].

Bieleski [BIE83] uses STDs to document the changes to a VDU screen, as seen by the user, that occur as the result of entering commands, replying to system prompts and selecting options from menus. The five types of nodes used to represent these changes are shown in figure III.1a.

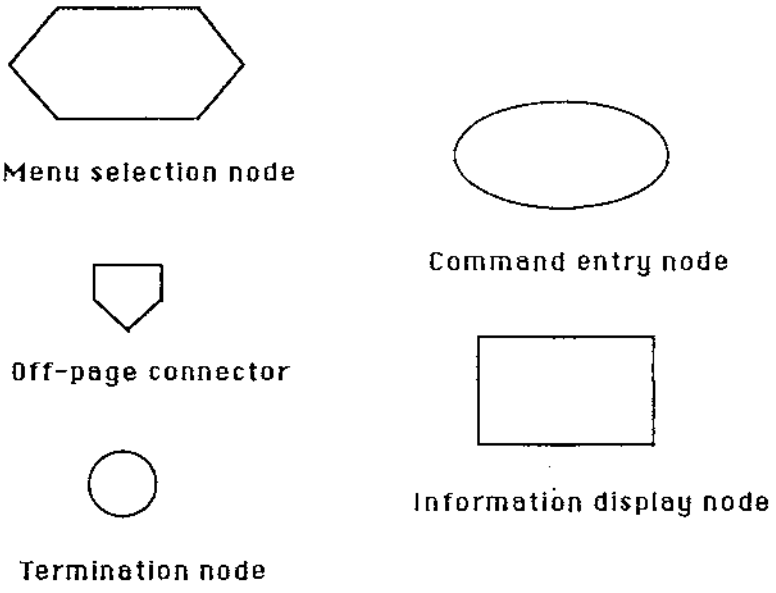


Figure III.1a - Bieleski's STD node types.

The nodes within a STD are connected by transition paths, as shown in figure III.1b. A transition path may include parameters that the user must specify in order to complete the transition from one state to another, and may also include branch points.

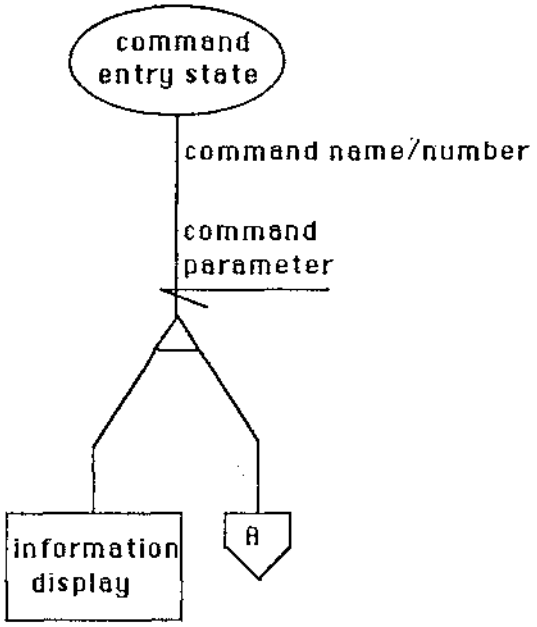


Figure III.1b - An example STD transition path.

A branch point indicates that some underlying processing is performed in order for the system to determine which path the transition should follow. Bieleski's STD notation is described in greater detail in [BIE83].

It was necessary to adapt Bieleski's notation in order to represent pull-down menus. In addition to the nodes shown in figure III.1a, two additional node types have been used. These node types, as shown in figure III.1c are:

(1) a menu node with bold outline, the 'Menu State' node type; and

(2) an off-page connector with bold outline, the 'Menu State Offpage Connector'.

Menu state node Menu state off-page connector

Figure III.1c - Additional node types.

A Menu State node is used to group together those commands within all of the pull-down menus that are active at a particular stage of the user's interaction with MUSSAT. For example, the STD shown in figure III.3 gives an overview of all the commands available to the user during Menu State B, that is, once a MUSSAT project has been selected.

In some menu states, some of the pull-down menus are not shown. For example, Menu State B (figure III.3) does not include the SYSTEM, ICON or SELECTED OBJECT menus. The absence of these menus from the STD of figure III.3 signifies that all of the commands within these menus are inactive within Menu State B.

The Menu Offpage Connector is used similarly to the standard off-page connector, except that the off-page reference is to a Menu State node rather than a menu node.

The STDs shown in figures III.2 to III.6 provide an overview of the general structure of the MUSSAT user interface. Two departures from the standard STD notation have been made in order to provide a concise description of the interface.

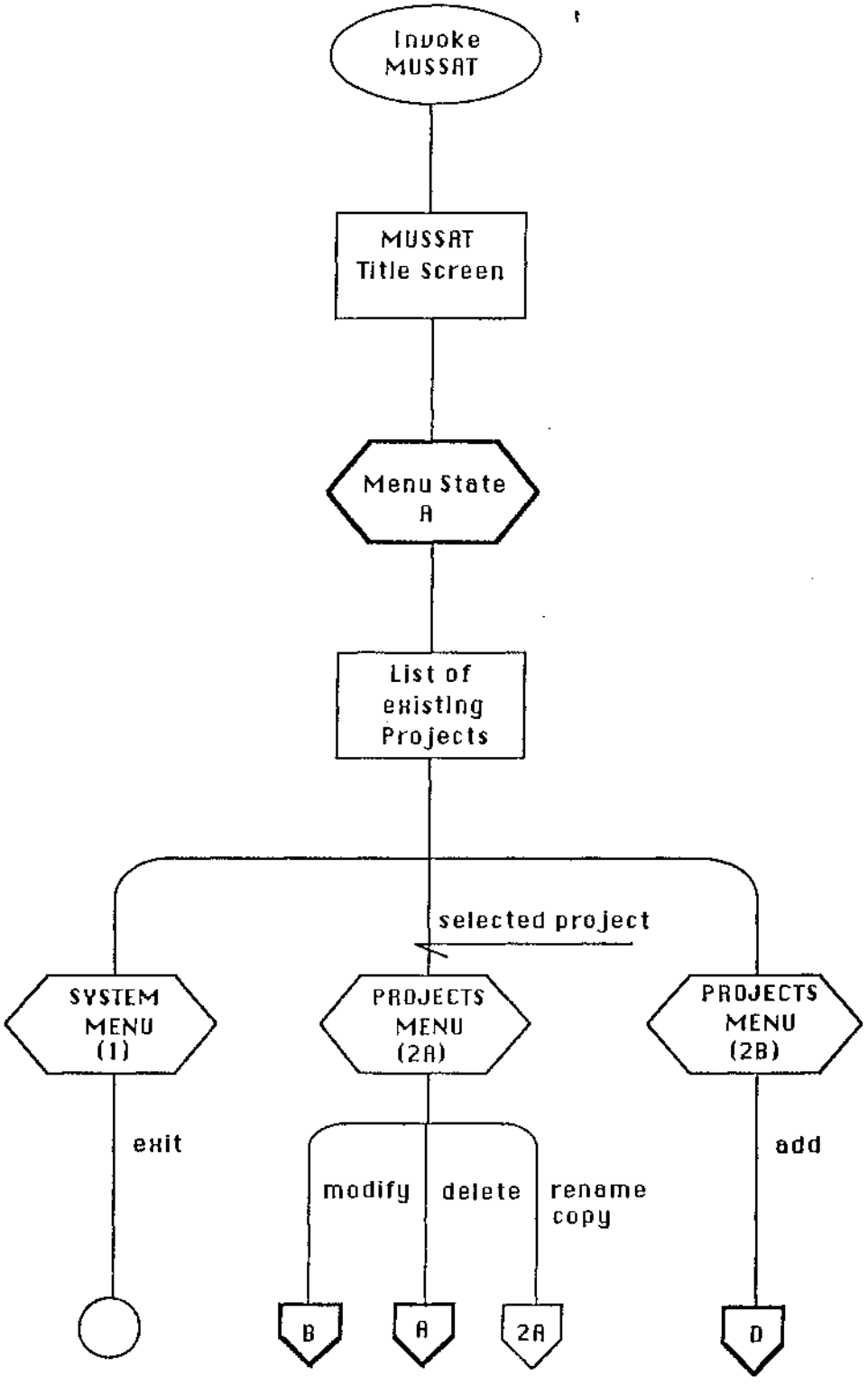


Figure III.2 - System Entry State

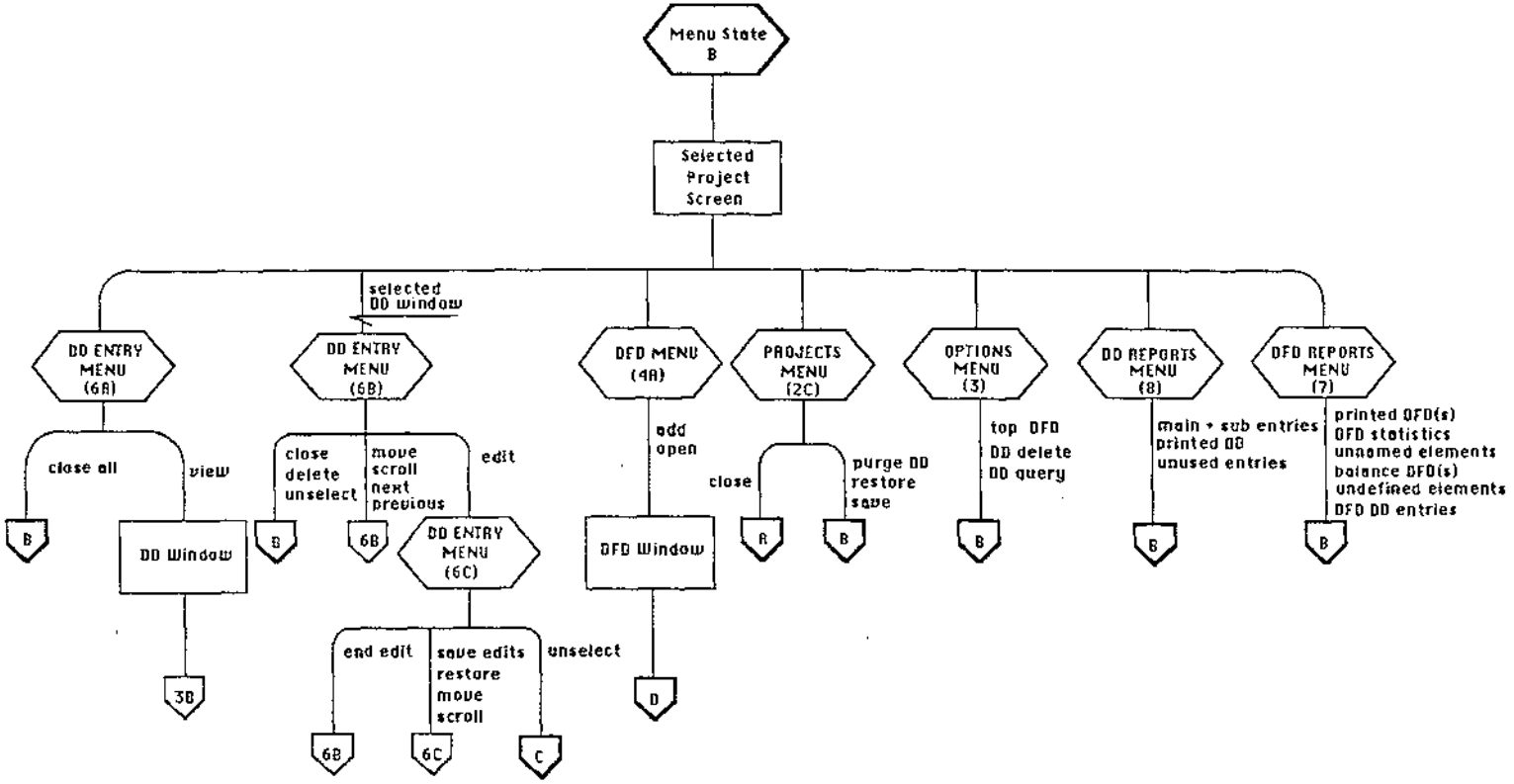


Figure III.3 - Selected Project State

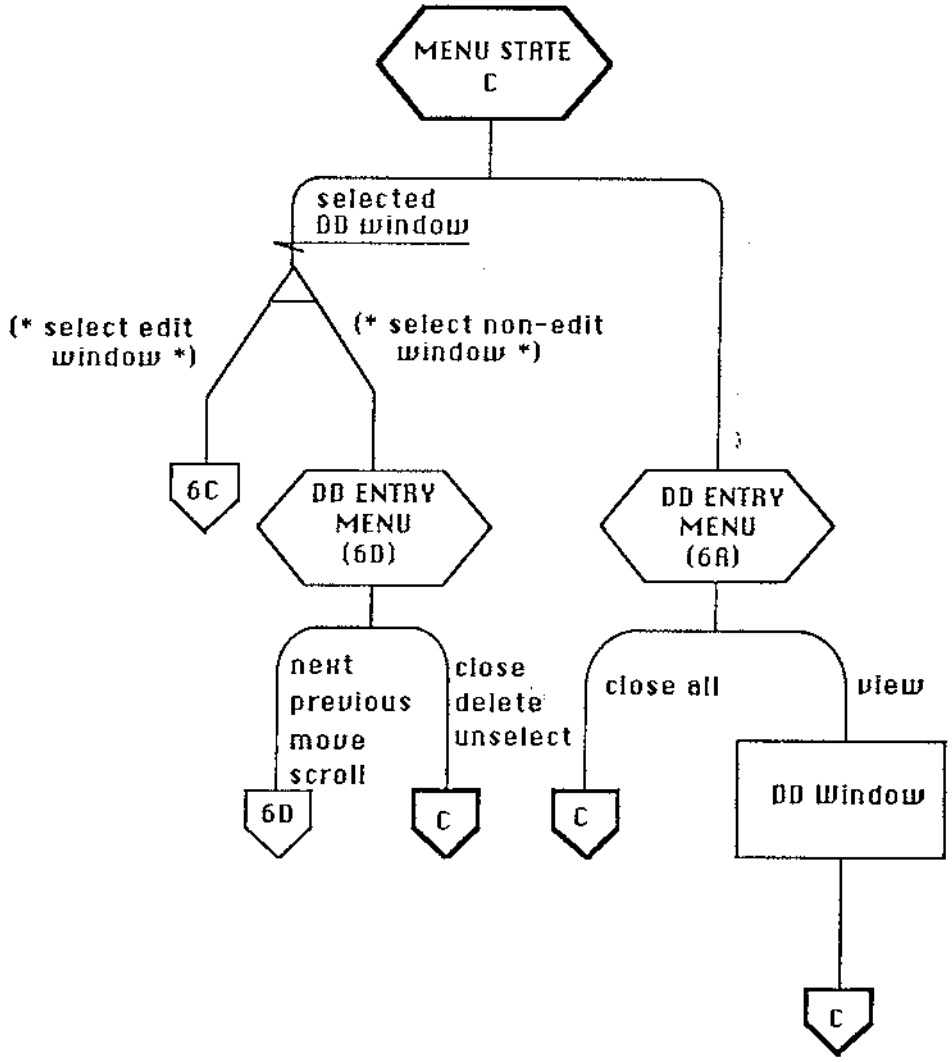
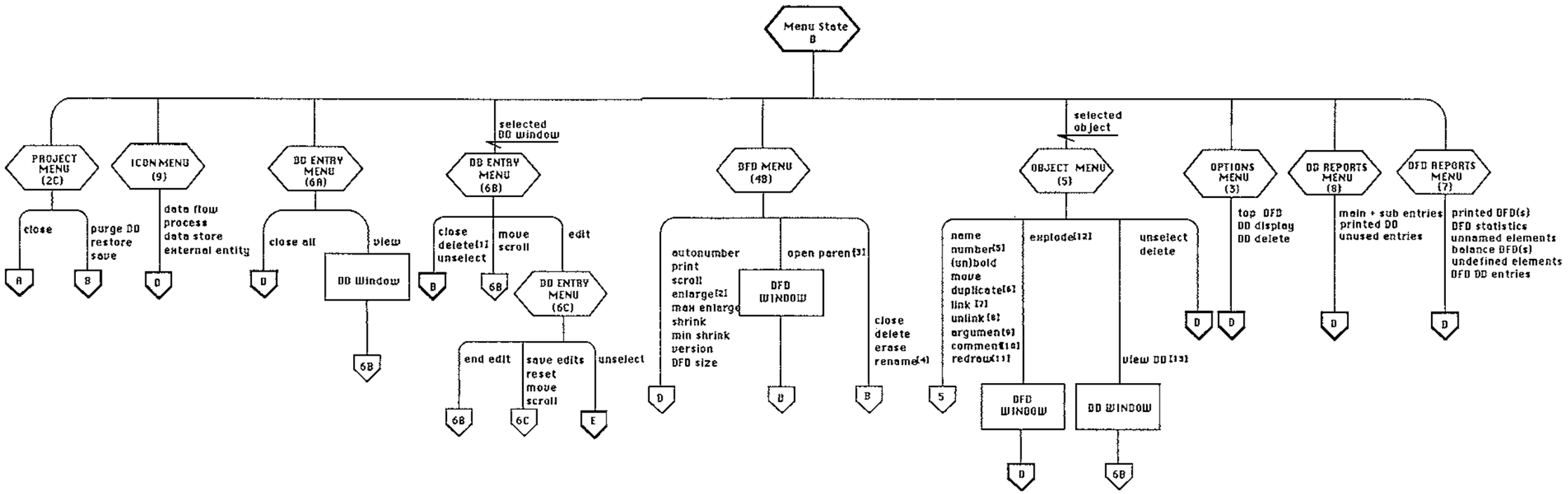


Figure III.4 - Edit State (no selected DFD)



[1] Only DD entries with UNUSED status may be deleted.

[2] Enlarge or maxenlarge may only be selected if the current scale is less than full size; vice versa for maxshrink and shrink.

[3] Only active for DFDs that have a parent DFD.

[4] External entities may not be numbered.

[6] Processes and data flow lables may not be duplicated.

[5] A DFD may only be renamed if it is not a child DFD of another process.

[7] Only a named process, with no child DFD may be used as the object of the LINK command.

[8] Only a process with a child DFD may be used as the object of the UNLINK command.

[9] An argument may only be specified for a data flow flowing from a data store.

[10] A comment field may only be specified for a process.

[11] Only active for data flows.

[12] Only processes may be 'exploded'.

[13] The DD entry of unnamed DFD elements may not be viewed.

Figure III.5 - Selected DFD State

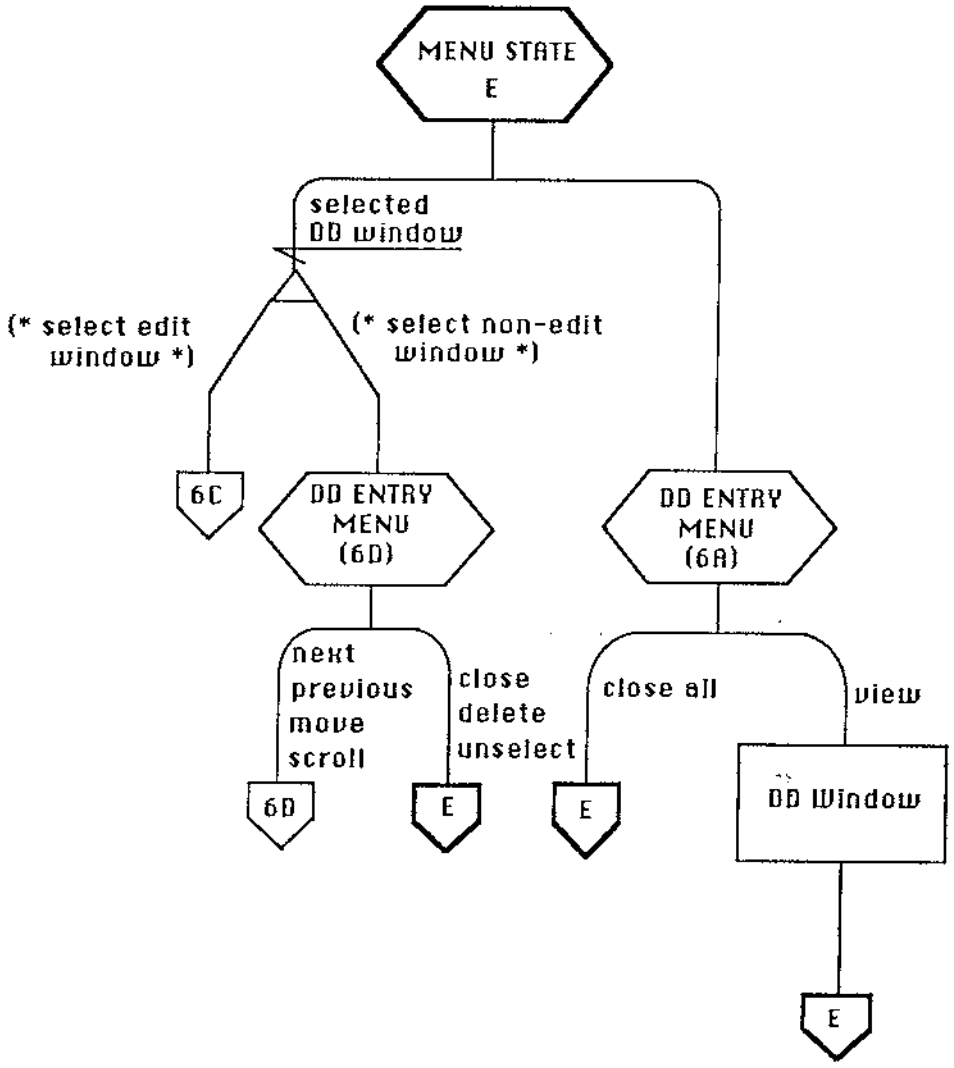


Figure III.6 - DD Edit State (selected DFD)

These departures are:

(1) The parameters that the user must supply for some commands are not shown. For example, the user must specify a DD ENTRY name for successful execution of the VIEW command contained in the DD ENTRY MENU (6A) of Menu State B (figure III.3).

In the case of the VIEW command, the necessary parameter could have been shown on the transition path from the menu node as shown in figure III.7a.

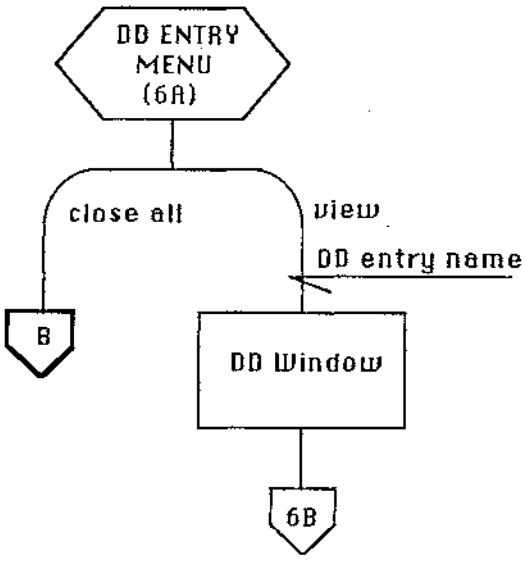


Figure III.7a - Simple command parameter on a STD transition path.

If, however, the required parameter(s) were shown for all commands in MUSSAT then the STDs would be less concise since each command with unique parameters would require a separate transition path from the parent menu node; in figures III.2 to III.6 all commands with a common end state are grouped together on one transition path. Figure III.7b shows how the DD ENTRY MENU (6B) from figure III.3 would be represented if command parameters were included.

(2) In addition, Menu State D (figure III.5) includes a series of footnotes which qualify when several commands can be activated, rather than using a separate transition path and logical menu for each of these commands.

For example, if the STD notation were used to accurately describe the DELETE command of the DD ENTRY MENU (6A) in figure III.5, then a new version of the DD ENTRY MENU would be required to represent the qualification specified in footnote [1]: that only DD entries marked as unused may be deleted using the DELETE command.

Figure III.7c is the full STD representation of the DELETE option showing the additional transition path and menu format (DD ENTRY MENU(6*)) required. Similar expansion of the existing STDs would be required to depict the other annotated commands within figure III.5

if the STD notation was strictly observed.

Figure III.8 shows each of the menus in the STDs as they would appear to the user upon selecting the appropriate menu title within the menu bar. When implemented, each of the annotated commands within figure III.5 would require separate menu states. That is, if the user were to select a process bubble within a DFD, and were then to view the OBJECT MENU, then the DUPLICATE command would be inactive in order to comply with footnote [6] in figure III.5.

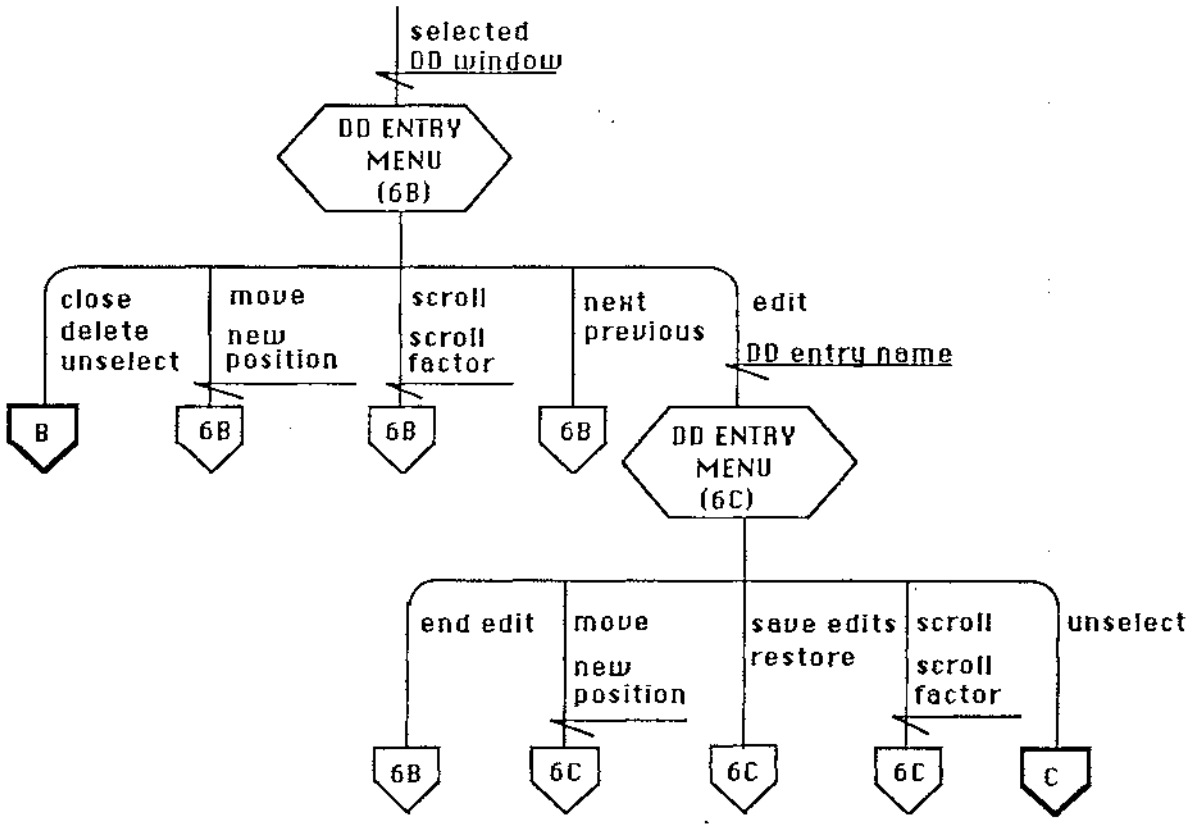


Figure III.7b - DD ENTRY MENU (3B) with all command parameters shown.

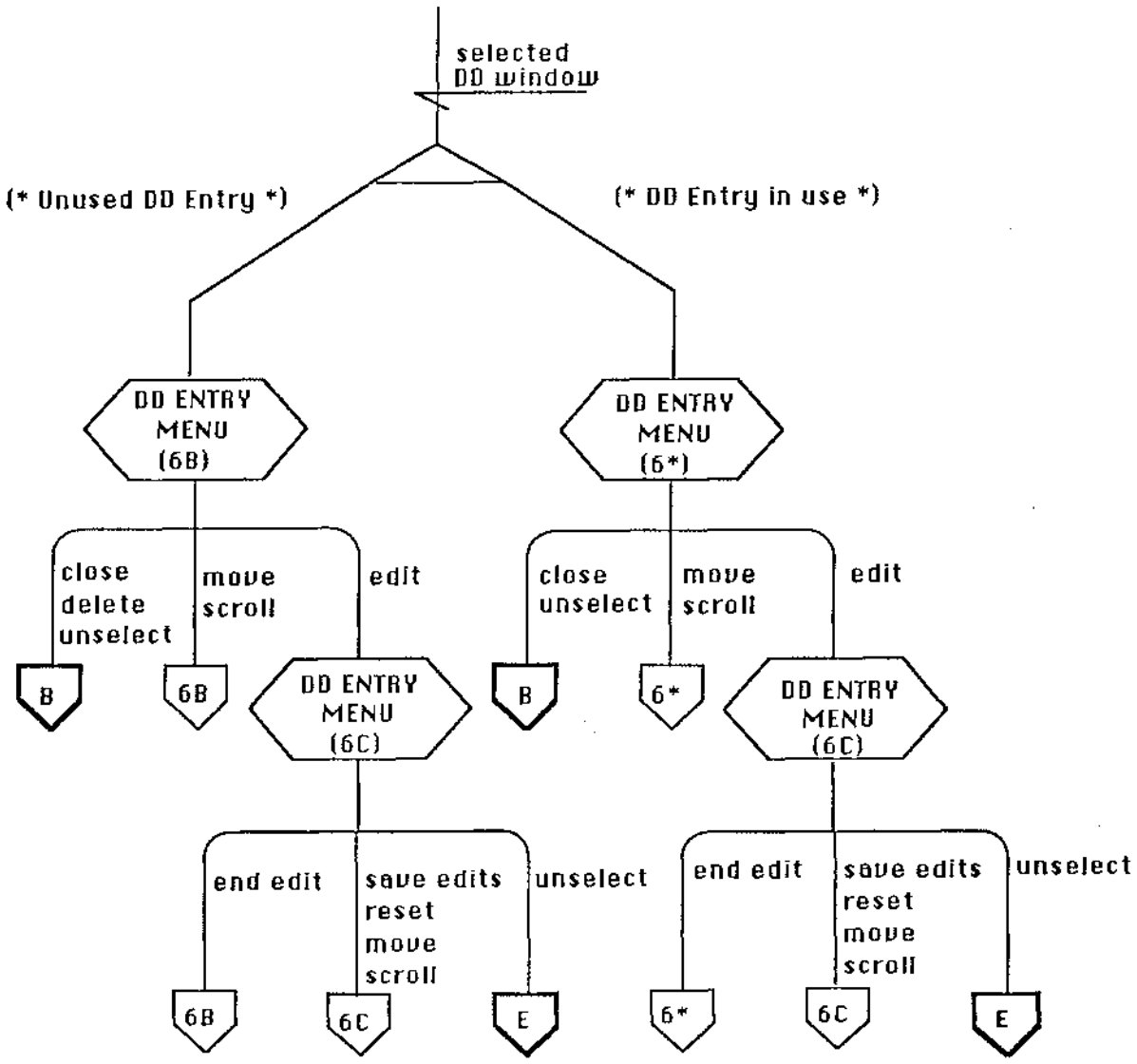


Figure III.7c - Expanded STD representation of the DELETE DD ENTRY command.

In some cases, the number of options shown within the menu formats of figure III.8 is less than the number of options shown on transition paths from the relevant menu node. For example, in figure III.3 the commands UNSELECT, MOVE, SCROLL, NEXT and PREVIOUS are shown as being active within the DD ENTRY MENU (6B), yet the corresponding menu format (figure III.8c) does not include these commands. Similarly, the MOVE command in the OBJECT MENU (5) in Menu State D (figure III.5) does not appear in the corresponding menu format (figure III.6e).

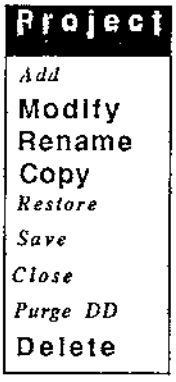
Commands that are shown in STDs and not in the menu format diagrams are still active commands, but are not activated by selection from a pull-down menu. For example, the SCROLL command that logically belongs in the DD ENTRY menu is activated by using the scroll box within the vertical scroll bar (as discussed in Appendix III). And the NEXT and PREVIOUS commands are activated by selecting the appropriate soft-button at the bottom of the DD Entry window. The MOVE command that logically belongs in the DD ENTRY MENU is activated by dragging the window using the move bar.

The MOVE command shown as part of the OBJECT MENU in figure III.5 is activated by the user selecting a DFD element and dragging the object (and any connected data flows) to the desired position. A detailed explanation of manipulating DFD elements is given in section 6.

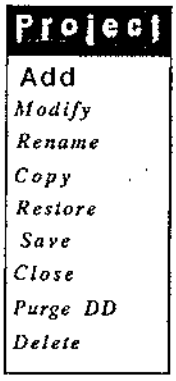
In addition to the commands identified above, there is one special command, CLOSE (in DFD MENU (4b), figure III.5, and in DD ENTRY MENU formats (6B) and (6D), as shown in figures III.3, III.4, III.5 and III.6). This command is shown in the corresponding menu format diagrams in figures III.6d and III.6c (respectively) and hence may be activated by selecting the command from the pull-down menu. The CLOSE command may also be activated by selecting the close box in the left-hand corner of the surrounding window frame. Unlike the CLOSE command in the DFD MENU and the DD ENTRY MENU, the CLOSE command in the PROJECTS MENU (2) may only be activated by selection of the command from the pull-down menu, since a project is not enclosed within a window frame.

System Menu 1

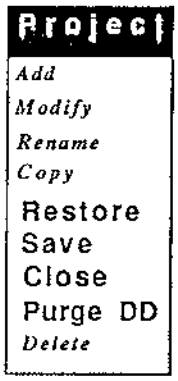
Figure III.8a - Menu Format 1



Projects Menu 2A



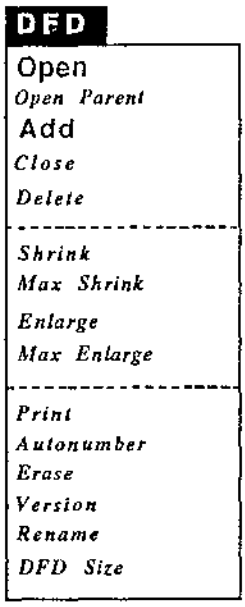
Projects Menu 2B



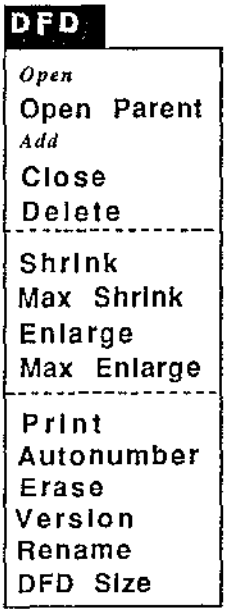
Projects Menu 2C

Figure III.8b - Menu Formats

Options Menu 3

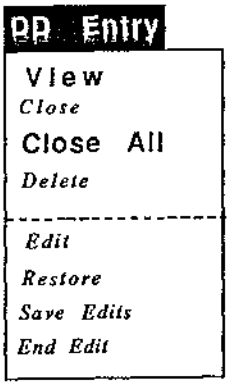


DFD Menu 4A

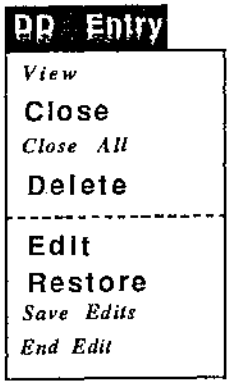


DFD Menu 4B

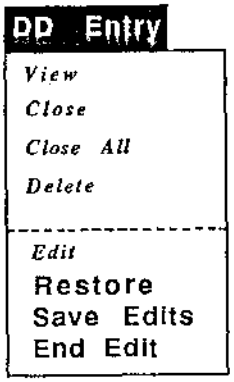
Figure II8.c - Menu Formats (cont.)



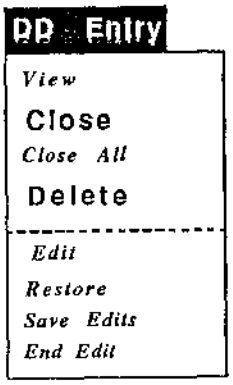
DD Entry Menu 6A



DD Entry Menu 6B

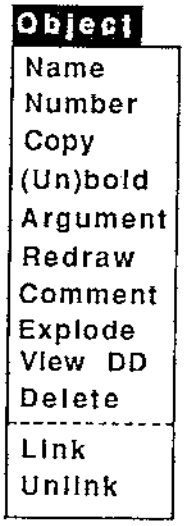


DD Entry Menu 6C

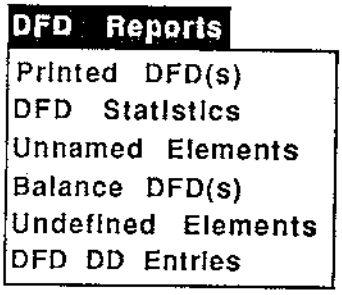


DD Entry Menu 6D

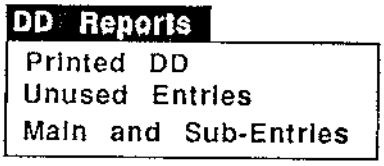
Figure III.8d - Menu Formats (cont.)



Object Menu 5



DFD Reports Menu 7



DD Reports Menu 8

Figure III.8e - Menu Formats (cont.)

Appendix IV

MUSSAT Commands in Detail

Each MUSSAT command shown in the STD diagrams in Appendix III is described in more detail in this appendix. The parameters required and the results of the successful execution of each command are described. The commands appear in the same order as shown in the menu formats in Appendix III.

ICON MENU

The icon menu allows the user to draw DFD elements in the displayed DFD. Each of the icons in the icon menu is drawn to scale, so that the DFD objects within a DFD viewed at the first zoom level are the same size as the icons.

To add a new process, data store or external entity, the user selects the appripriate icon, moves the cursor to the desired position in the DFD, and then clicks the SELECT mouse button. The selected position indicates the position that the centre of the selected DFD element will occupy.

In the case of an external entity, process or data store, if selected position would result in the DFD element overlapping an existing DFD element, then MUSSAT will not allow placement of the icon in the selected position; an audible alarm will be sounded and the DFD element will not be drawn. The user may then specify another position or press the

UNSELECT mouse button to terminate the drawing operation.

If, however, the specified position would not result in the selected DFD element overlapping any other DFD element, then the new element will be drawn, with the centre of the DFD element at the selected position.

To draw a data flow, the user must specify the start point, end point and any intermediate turning points for the data flow. An automatic data flow layout algorithm has not been incorporated in MUSSAT; the user has complete control over the path that a data flow will follow.

The general procedure for drawing a data flow is the same for both single and double headed data flows, although each type of data flow has different rules that determine valid start and end points.

A one headed data flow must be connected to a process at either one or both ends of the data flow. The start and end points of a two headed data flow must be a data store and a process.

If path specified for a data flow causes the data flow to cross one or more existing data flows, then a hoop will be drawn on the new data flow at the point of intersection. A data flow may not intersect a data store, process or external entity. If a data flow intersects a data flow name or argument label, then the data flow will be drawn 'under-

neath' the labels, so that the portion of the data flow that intersects a data flow label is obscured by the overlying labels.

The user draws a data flow by performing the following steps:

(1) Select one of the two data flow flow icons (if not alrady selected as the result of aprevious draw operation);

(2) Select a valid starting position for the data flow;

(3) Specify zero or more line segments making up the data flow path; and

(4) Select a valid end point for the data flow.

The following example describes the steps carried out to create the a single headed data flow between two processes. Figures V.1a to V.1d are used to illustrate the example.

The user first selects the single headed data flow icon from the icon menu. As a result, the selected icon is highlighted and the and, if necessary, the 'select' cursor shape replaced with the 'draw' cursor shape. The user then selects a valid start point for the data flow, either an existing DFD object or an unnocupied space in the DFD. In this example a process has been chose as the start point. MUSSAT then draws a small 'construction box' on the perim-

iter of the selected DFD object as close to the selected point as possible (figure V.1a).

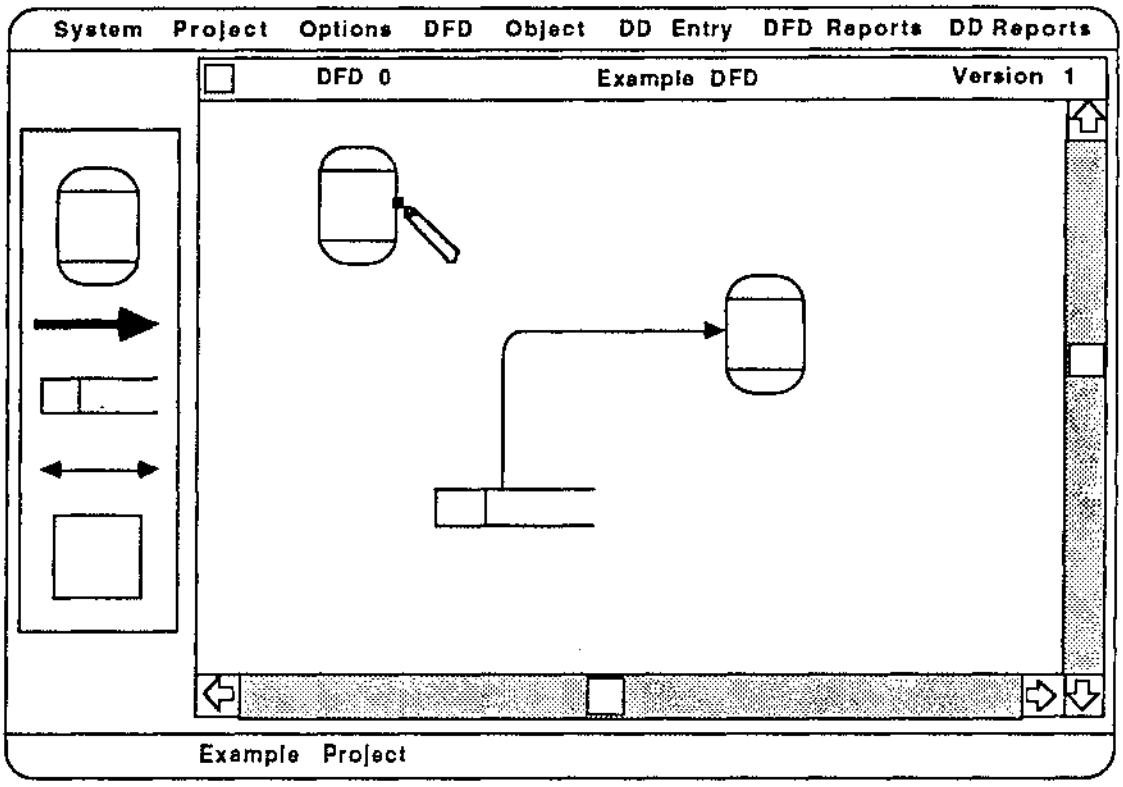


Figure IV.1a - Selecting the starting point of a data flow.

If the user were to have selected an unnocupied point in the DFD as the start point, then the construction box would have been drawn at the point indicated by the cursor.

The user then pulls the line connected to the construction box to a second point and presses the SELECT mouse button[1]. A second construction box will appear at the second

[1]Unlike 'dragging' the cursor, when 'pulling' the cursor to a new position the user does not keep the SELECT mouse button pressed. The data flow line remains fixed to and follows the cursor, until the SELECT button is pressed.

selected point, and the previously specified line segment becomes fixed (figure V.1b).

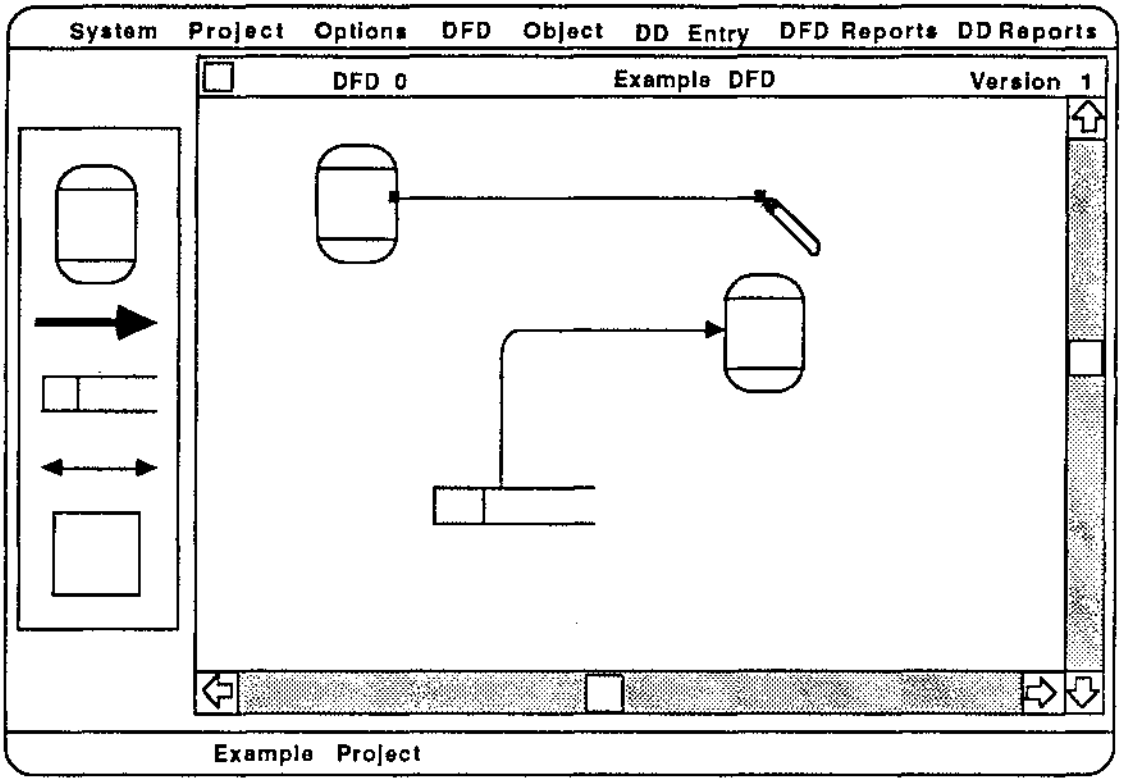


Figure IV.1b - Creating a data flow line segment.

If an existing DFD element is selected as the starting point of a data flow, then the user may only pull the data flow line perpendicular to the side of the DFD object on which the construction box is drawn. In other cases, where the construction box is not drawn on an existing DFD element, the data flow line may be dragged either 90 or 180 degrees to the construction box. Furthermore, the user is unable to pull a data flow line through an existing process, data store or external entity.

As many line segments as required may be thus specified. MUSSAT will end the drawing operation when a valid end point for the data flow type is selected as the end point of a data flow line segment.[2] An audible alarm will be sounded and the data flow will not be completed if an invalid end point is selected. Figure V.1c shows the cursor positioned to select the second process as the end point of the data flow.

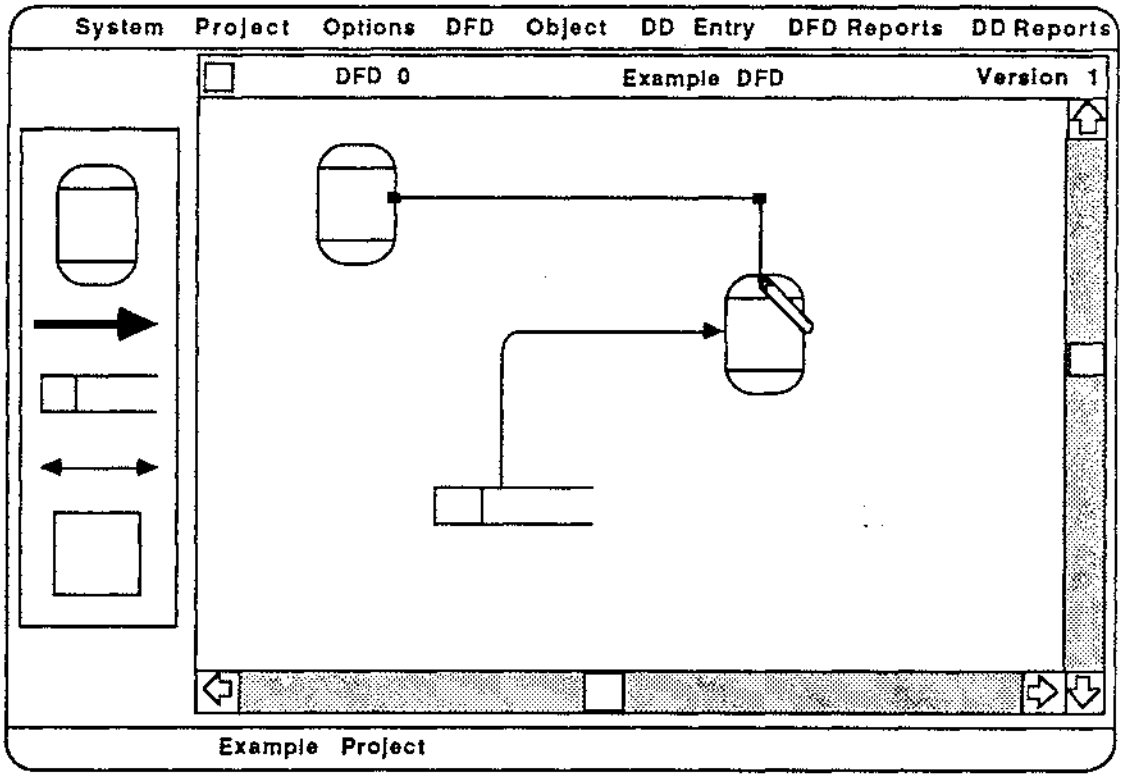


Figure IV.1c - Selecting a data flow end point.

[2]To draw a data flow that is to end at an unnocupied point in the DFD, the user pulls the final line segment to the desired end position and then presses the ENTER key. The data flow arrow head will be drawn at the point indicated by the cursor.

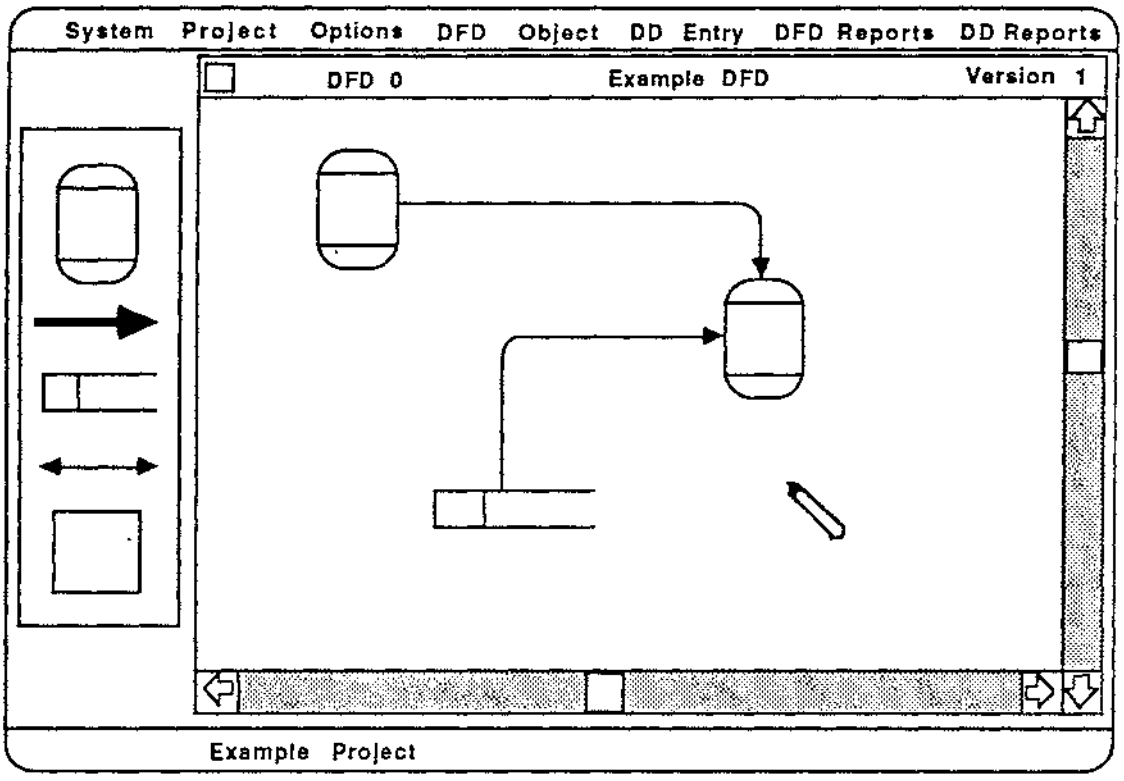


Figure IV.1d - The completed data flow.

Once a data flow is fully specified, MUSSAT will remove the construction boxes, will redraw any turning points in the data flow path with rounded corners, will draw hoops at any point where the new data flow intersects an existing data flow, and will draw the required arrowhead(s) on the appropriate end(s) of the data flow. In the case of a single headed data flow, the arrowhead will always be drawn on the end of the last specified data flow segment (figure V.1d).

At any stage, when specifying a data flow path, the user may press the DELETE mouse button to erase the current line segment. This will allow the user to respecify the last completed line segment. The DELETE mouse button may be repeatedly pressed until all data flow points, including the start point is removed.

The selected icon within the icon menu will remain selected after creating a new DFD element so that multiple elements of the same type may be created without the user having to select the appropriate icon each time.

Once a DFD element has successfully been created, a DD entry is automatically made for the new element with information obtained from the DFD. This information incudes: the DFD element position, line width (automatically set to normal for a new DFD element), and the DD key(s) of any other connected DFD elements.

SYSTEM Menu

EXIT

This command terminates MUSSAT and returns the user to the host operating system.

PROJECT Menu

ADD

The user may create a new MUSSAT project using the ADD command. A dialogue box will be used to prompt the user for a project name, which must be non-null and unique. Once a project name is specified, the project is created and an empty DFD named 'DFD 0' is created and displayed. The value of TOP DFD (see OPTIONS menu commands) is automatically set to 'DFD 0'.

MODIFY

After selecting a project to work on from the list of existing project names, the user may then make modifications or simply view, the selected project using the MODIFY command. No DFD or DD entry is automatically

displayed as the result of executing the MODIFY command.

RENAME

The RENAME command allows the user to rename a project selected from the displayed list of existing projects. The user will be prompted to specify a new name for the selected project. The new name must be non null and unique. The list of existing projects visible in the display area is updated and the renamed project remains selected after successful execution of a RENAME.

COPY

The COPY command allows the user to make a copy of any of the MUSSAT projects. The user is prompted via a dialogue box to specify a name for the copy of the project. Once a non null, unique name is specified, the selected project is copied and the name of the copy of the project is added to the list of existing projects. The original copy of the project in the list of projects remains selected after the COPY is completed.

RESTORE

This comand allows the user to replace the current version of the project accessed via a MODIFY or ADD command (from the PROJECT menu) to the last saved version (created by executing the SAVE command), or, if SAVE

has not been used, then to the state of the project before the MODIFY or ADD operation was activated. The user will be required to confirm activation of the RESTORE operation via a dialogue box.

All DFDs and DD entry windows open before the RESTORE is performed will be closed as the result of a RESTORE operation.

SAVE

The SAVE command allows the project, in its curent state, to be saved, replacing the last saved version of the project. If a RESTORE operation is subsequently activated, then the project is restored to the state as at the last SAVE command.

The user will be required to confirm activation of a SAVE via a dialogue box. The viewing area will remain unchanged after a SAVE operation.

CLOSE

The CLOSE command allows the user to exit from the project being created or amended, back to the main display where the list of existing projects is displayed.

A dialogue box is displayed and the user will be required to specify whether any changes made since the last SAVE operation should be saved before exiting from the current project. Depending on the option selected,

the project is either saved (as for the SAVE operation) and closed, or simply closed. All open DFDs and DD entries will be removed from the screen.

PURGE DD

This operation allows the user to delete all DD entries marked as UNUSED from the project data dictionary. Any DD entries that are unused and visible when the command is executed will be removed from the screen once the PURGE DD operation is completed. The user will be required to confirm the execution of this command via a dialogue box.

DELETE

All DFDs and DD entries for project selected from the list of existing projects are removed from the system. The user will either confirm or cancel the execution of the DELETE command via a dialogue box. The selected project name will be removed from the list of existing projects displayed on screen. No project will remain selected after a successful DELETE is performed.

OPTIONS Menu

The four commands within the Options Menu allow the user to specify values for each of the MUSSAT project options. Three of the commands in this menu: DD DISPLAY, DD DELETE and DFD BORDER allow the user to select one of two values for the associated parameter. Alternatively the user may

accept the MUSSAT default value. The value of the parameter remains set at the selected value until another value is selected. The values set by the user are also retained between MUSSAT sessions.

The fourth command, TOP DFD, will be given a default value by MUSSAT which may be changed by the user. In addition, the value specified for TOP DFD will automatically be set to null if the DFD specified as the TOP DFD is deleted.

DD DISPLAY

This option allows the user to specify the format of displayed DD entries. The two values for the DD DISPLAY option are FULL and SUMMARY (the latter being the default value).

The former value causes the system maintained fields of the DD entry to be included in the report. Such fields include, for each occurrence of a DFD element, the name of the DFD in which the occurrence exists and the source and destination elements (for data flows) or data flows in and out (for all other DFD elements). For data flow, data structure and data element entries, all DD entries that use the entry in their definition field(s) are listed in the USED BY field.

The summary entry format is a briefer format and does not include the system maintained information identified above. This format includes the fields that the

user may edit, all aliases, and an indication of whether the entry is used or unused.

If the user changes the value of DD DISPLAY, then any DD entry windows open at the time will be redrawn to conform to the new DD DISPLAY value.

A dialogue box is used to display the current value of DD DISPLAY before the user is given the option of resetting the value.

DD DELETE

This option determines what will be done with DD entries that are no longer required after either changing the definition field of a DD entry, or deleting a DFD element.

The default setting of SAVE means that all entries no longer required are marked as unused and kept in the DD. These entries may later be deleted from the DD by the user. The other possible setting, DELETE, automatically deletes all DD entries no longer required, except for the entry named that serves as the main entry of a deleted DD entry with an alias.

If the DELETE option is selected, all entries no longer required as the result of subsequent operations will be deleted; any DD entries marked as unused before the DELETE option is selected will remain in the DD.

A dialogue box is used to display the current value of DD QUERY before the user is given the option of resetting the value.

TOP DFD

This operation allows the user to specify the DFD at the top of the current project's DFD hierarchy. The top DFD cannot have a parent process. This operation is necessary since unlinked DFDs may exist as a sub-hierarchy within the current SSA project.

If TOP DFD has a null value, then, for the generation of reports that require the name of a DFD to use as the top of a hierarchy of DFDs, the user will be required to specify a DFD name before any further processing is performed. Conversely, if TOP DFD has a valid value, then MUSSAT will allow the user to either accept the value specified as TOP DFD or the user may specify a different DFD name as a parameter to the command.

When a MUSSAT project is created TOP DFD is automatically set to the first DFD created (see the ADD command in the Project Menu). If the DFD corresponding to the value in TOP DFD is subsequently deleted, then TOP DFD is automatically reset to null.

When the TOP DFD is selected, the name of the current TOP DFD is displayed and the user is prompted to specify a new value for TOP DFD.

DFD BORDER

The DFD BORDER system option allows the user to specify whether or not DFDs within the project should be drawn with surrounding borders as in the G & S notation.

If the DFD BORDER system option has the value DRAW BORDER, then all existing DFDs any DFDs created at a later stage within the project will be drawn with a surrounding border. The border will be drawn a fixed distance in from the edge of the DFD
with enough space left for a data store or external entity to be drawn outside the border. The border has no significance to MUSSAT, but it does allow the user to draw DFDs that conform to the G & S conventions.

The current value of DFD BORDER will be displayed in a dialogue box and the user will be able to select a new value, if desired. If the value of DFD BORDER is changed and a DFD is currently visible, then the DFD will be redrawn to conform to the new value. The default value will be NO BORDER.

DFD elements may overlap the border, in which case the DFD element will obscure the underlying portion of the border.

DFD Menu

The commands within the DFD Menu fall into two categories: those that affect the way MUSSAT presents the DFD to the user, and those that allow the user to modify certain attributes of the DFD.

Four of the commands within the former category allow the user to specify the level of magnification at which a DFD is viewed and manipulated. These comnmands are : SHRINK, MAX SHRINK, ENLARGE and MAX ENLARGE.

At least three magnification, or MUSSAT. The following desired features should be taken into consideration when scale factors are determined for a particular hardware configuration:

(1) The highest zoom level should allow the user to view a representation of the entire DFD within the viewing area. This feature is linked to the maximum number of A4 pages allowed in one DFD, that is, the greater the number of pages allowed in a DFD, the higher the scale factor will need to be in order to view the entire DFD.

(2) The second zoom level should allow the user to view as much of the DFD as possible while maintaining high enough resolution so that the user can manipulate individual DFD elements, although the user may not necessarily be able to read text labels.

(3) The first zoom level should allow the user to view the DFD with DFD elements the same size the DFD elements in the icon menu. All text labels should be easy to read at this leel.

When a DFD is created it will initially be displayed using the first zoom level.

Page divisions will be shown when the DFD spans more than one A4 page.

OPEN

An existing DFD within the current project may be displayed and subsequently modified using the OPEN command. The user will be prompted for the name or number of the DFD to retrieve via a dialogue box. The DFD view displayed and the zoom level used will be the view and zoom level that the DFD was last viewed at.

If a DFD is already displayed when the OPEN operation is activated, the current DFD is closed and the newly specified DFD becomes the current DFD. If the DFD is larger than the viewing area, then the top, left-most portion of the DFD is displayed.

OPEN PARENT

The parent DFD of the open DFD may be opened by selecting the OPEN PARENT command in the DFD menu. This command will only be active if the displayed DFD has a

parent DFD. The current DFD will be closed and the parent DFD displayed.

ADD

This command allows the user to add a new DFD to the current project[3]. The user will be required to specify the name of the new DFD via a dialogue box. The DFD name must be unique and cannot be the same as an existing DFD, process or unused process DD entry.

When first displayed, the dialogue box will display a default name for the DFD. The default name will be "UNTITLED #" where # is the next unused number in the default numbering range. The default name may either be accepted or overtyped with another name. The user cannot specify a DFD name that is the same as an already allocated default name, or a name that is in the default name format with the '#' number greater than the number of the default name offered in the dialogue box. After a valid name is specified, a new (empty) DFD will be created and displayed. The DFD created as the result of an ADD operation will not be linked to any other DFD or process.

---

[3]There are two commands that allow the user to create new DFDs within a specified project: the ADD command from the DFD Menu and the EXPLODE command from the Selected Object Menu.

CLOSE

The CLOSE command allows the user to remove the current DFD from the screen and may be activated either by the close box in the DFD windown, or by selecting the CLOSE command in the DFD menu. The DFD window is removed from the viewing area.

If the DFD has been amended then the DATE LAST AMENDED field for the DFD is updated to the current date.

DELETE

The DELETE operation allows the user to delete the current DFD, and all child DFDs from the current project. Depending on the value of the system parameter, DD DELETE, DD entries that are no longer required after the DFD(s) are deleted will be either marked as UNUSED, or deleted from the DD. The user will be required to confirm the execution of the DELETE operation via a dialogue box. The displayed DFD will be removed from the viewing area after successful execution of the DELETE command.

SHRINK

The SHRINK command allows the user to increase the zoom level by one. For example, if the DFD is currently being viewed at the first zoom level, then the SHRINK command will cause the DFD to be redrawn using The second zoom level.

MAX SHRINK

The MAX SHRINK command causes the DFD to be redrawn using the highest zoom level, regardless of the zoom level the DFD was displayed at prior to the MAX SHRINK command being activated.

ENLARGE

The ENLARGE command is the opposite of the SHRINK command: the DFD is redrawn using the next lowest zoom level.

MAX ENLARGE

The MAX ENLARGE command causes the DFD to be redrawn using the lowest zoom level.

PRINT

The PRINT command allows the user to obtain a printed copy of the DFD currently being viewed. This operation requires fewer user specified parameters to execute than the printed DFDs report and is intended to be a faster way of printing a copy of the current DFD.

The user will be required to confirm the execution of the PRINT command via a dialogue box.

AUTO NUMBER

Selection of the AUTO NUMBER command causes MUSSAT to number all named but not numbered processes and data stores within the current DFD.

In the case of data stores, a list of allocated numbers is maintained by the system (numbers previously allocated to elements that have been deleted will be re-used where appropriate). If, within the current DFD, there exists a new, named but unnumbered data store, then the data store is numbered with the next free number. All future occurrences of the data store will use the same number. The numbering system used for automatic numbering of data stores and processes is the default G & S numbering system.

Processes are numbered left to right, top to bottom across the DFD replacing any existing numbers. Child DFDs of processes are also renumbered if the new number is different from the previous number.

A dialogue box will be used to confirm the execution of this command since automatic numbering will repalce any previous numbers and will therefore affect the numbering of any child DFDs.

ERASE

The ERASE command erases all DFD elements in the current DFD and then allows the user to redraw the DFD using the DD entries associated with the named elements in the original DFD.

When an ERASE is performed, a list is kept of all DD entries that were used in the original version of the

DFD. As the DFD is redrawn, any DFD element of the same type and name as a DD entry on the list is automatically linked to the DD entry. The temporary list of DD entries is only used until the DFD is closed. Once the DFD is closed, any remaining entries on the temorary list that are not used by any other DFD element are either deleted or marked as UNUSED (depending on the value of DD DELETE).

When a DFD contains a large number of elements, it may be easier to re-draw the DFD rather than trying to move elements within the existing DFD in order to make the DFD more readable. The ERASE command has been designed to allow this sort or rationalisation of DFDs, without the user having to respecify all DFD element and DD entry connections.

The user will be required to confirm the execution of an ERASE command via a dialogue box.

VERSION

Each DFD has a 'VERSION' field, shown in the upper right hand corner of each DFD window. The version field is intended to be used by the user to keep track of how many revisions a DFD has undergone. The VERSION command allows the user to change the value of the VERSION field displayed on the DFD. A dialogue box will be used to display the current value of VERSION for the

DFD that is currently being viewed, and the user may then change the value. The VERSION field may contain any printable characters.

RENAME

This command allows the user to rename a DFD. If the current DFD is not the child DFD of any process, then it may be renamed to a new unique name (including a system default name). A DFD cannot be renamed with the name of an existing process or an unused process DD entry.

The user will be prompted for a new name via a dialogue box. A default DFD name will be presented (as discussed in the DFD ADD command), which the user may overtype if a name other than the default name is preferred.

DFD SIZE

The DFD SIZE command allows the user to change the number of A4 pages that form the DFD being viewed. Each DFD is, by default, one A4 page in size. If, however, a larger size is required, the user may specify that the (current) DFD should be any physically adjecent, even number of A4 pages. Some maximum limit must be placed on the size of such DFDs; this will be an implementation dependent limit.

A representation of the current size of the DFD will be displayed in a dialogue box. The user may then change the number of pages used by manipulating the graphical representation of the DFD size. The size and placement of any existing DFD elements is not affected by changing the size of the DFD.

The (current) DFD size can also be reduced to no less that one A4 page, providing that any pages no longer required are empty.

The positions of the scroll boxes within the DFD window scroll bars will change as the result of changing the size of a DFD.

SCROLL

The SCROLL command allows the user to view portions of a DFD that lie ouside the current viewing area. By default, the top left-most page (or top left-most portion of this page if the window size is less than A4 sized) of a DFD is displayed when the DFD is first displayed.

OBJECT Menu

Any of a number of operations may be performed on elements in the current DFD, however, as discussed in Appendix III, some operations are limited to certain types of DFD elements or to individual DFD elements when certain conditions are

met. These restrictions are identified for each command.

NAME

The NAME command allows the user to name, or rename, the selected DFD element. All DFD elements may be named.

DFD element names may consist of any sequence of printable characters, with the character '+' used to separate the individual elements of a compound data flow name. A maximum name length of characters will need to be imposed so that all names will fit within their appropriate DFD element shapes on the DFD. This maximum length will be implementation dependent.

There are four situations that may arise when the user names a DFD element:

(1) An unnamed data store, external entity or data flow is given the name of an existing DFD element of the same type;

(2) An unnamed data store, external entity or data flow is given the name of a DD entry of the same type that is marked as unused;

(3) An unnamed data flow is given the name of a DD entry that only exists in the definition field(s) of other data flow DD entries; or

(4) An unnamed DFD element is give the name of an existing DFD element or other DD entry of a different type than the selected DFD element. For example, a data store is given the name of an existing data element DD entry, or a process is given the name of an existing external entity.

In the first case, the user would be prompted (via a dialogue box) to specify whether the selected element should become a duplicate of the already existing identically named element. If the user does not want to create a duplicate of the existing DFD element, then a new unique name should be specified for the newly named DFD element.

If the current DFD element is a data store and is named as a duplicate of another existing data store, then the new data store will also be automatically numbered with the same number as the original data store (if the original data store has a number specified).

In the second case, user would be prompted, via a dialogue box, to specify whether the DFD element should be linked to the unused DD entry. If the user does not want to link the DFD element with the unused DD entry then the DFD element should either be given a new or name, or the unused DD entry should be deleted.

Although the user cannot create duplicates of a process, an unused DD entry may be re-used for a process given the same name as a previously deleted process.

In the third case, the user would be prompted, via a dialogue box, to confirm whether the named DD entry should be recorded as a data flow. If the user confirms the operation, then the new data flow becomes a component of all other data flows that include the new data flow name in a DD definition field.

In the fourth, and final case, an error dialogue box will be displayed and the user will be allowed to respecify the DFD element name or to cancel the naming operation.

MUSSAT automatically places data flow names on the appropriate data flow(s). The user may reposition these labels if desired. All name labels are horizontally oriented and are limited to a maximum line length and maximum number of lines so that all labels fit within a predefined rectangular area.

A named DFD element may be renamed to a new unique name, in which case the DD entry name and all occurrences of the element in all DFDs within the current project will be renamed. In the case of processes, the name of the immediately subordinate child DFD (if one exists) will also be changed.

A DFD element that is already named may not be renamed with the same name as any other DFD element or data item (regardless of whether the data item has been defined in a separate DD entry or whether the name exists only in the definition field of another DD entry).

To do so would allow the user to implicitly delete the current instance of the DFD element and create a duplicate of the second DFD element. The user should be forced to perform each operation separately using the existing DELETE and DUPLICATE commands (within the OBJECT MENU), so that the effects of the operations are quite clear.

In addition, a DFD element may not be renamed to a null string.

When a data flow is renamed the new name is placed in the same position on the DFD as the old name.

A dialogue box will be used to display the current name of the selected DFD element and the user may then specify a new name.

NUMBER

A selected processes or data store may be numbered (or renumbered) using the NUMBER command. The NUMBER command allows users to allocate numbers to data stores

and processes individually rather than automatically numbering elements using the AUTONUMBER command in the DFD menu.

If the current DFD element is a process, then the user may specify a (new), one or two digit number for the process. This number must be unique within the current DFD.

If the current DFD element is a data store, then the user may specify a new number or short character string as the D-number. The user may use the standard G & S numbers or some other numbering system incorporating any combination of printable characters (with the exception that the string cannot be all blanks or have leading blanks).

If the data store is named but not numbered, then the D-number specified by the user must be unique and the DD entry will be updated so that all occurrences of the data store bear the same D-number.

If a data store is not named and is given a D-number that is used by some other named data store then the user will be asked, via a dialogue box, whether to change the unnamed data store into a duplicate of the already existing data store with the same D-number. If the user changes the selected data store into a duplicate of the other data store, then the new data store

is linked to the appropriate DD entry and current DFD is updated so that the name and number of the data store is shown in the current DFD.

If the user does not wish to create a duplicate of the existing data store with the same number, then a new number should be specified or the data store should be left unnumbered.

A data store that is already numbered may be renumbered to any unused D-number. All occurrences of the selected data store within the project will be renumbered with the new number.

DUPLICATE

The DUPLICATE command allows the user to create duplicates of named data stores, external entities or data flows within the current DFD. To create a duplicate, the user selects the DFD element to be duplicated, activates the DUPLICATE command, and then specifies the position the duplicated DFD element should occupy.

The user specifies the position of the duplicated DFD element in the same way as creating a new DFD element. The ICON MENU section gives a more detailed explanation of how the user specifies the position of a new DFD element.

The resulting DFD element has the same name, argument (for a data flow), number (for data stores) and font (either bold or normal) as the original DFD element. The duplicated element is automatically linked into the existing DD entry. The new copy of the duplicated DFD element remains selected after the DUPLICATE operation is completed.

(UN)BOLD

The (UN)BOLD command allows the user to toggle the line width of a selected DFD element from bold to normal or vice versa. Any text labels, names,or numbers associated with the selected DFD element are not affected. Only the line width of DFD element shape is changed. The selected DFD element, all occurrences of the selected DFD element in the current DFD, all occurences of the selected DFD element in the current DFD and all child DFDs, or all instances of the selected DFD element within the project are redrawn with bold or unbolded outline.

ARGUMENT

The ARGUMENT command allows the user to specify the key used to retrieve data from a data store. A search argument may be specified for any one-ended, named data flow out of a data store. MUSSAT automatically places the label as near to the data store as possible (for each occurrence of the data flow) and appends a small

diamond (rather than a triangle as used in the G & S methodology) to the front of the label to indicate a search argument. As with data flow names, data flow search arguments may be moved by the user. Data flow search arguments are text strings and are only recorded in the DD as a text label associated with the relevant data flow.

As with the NAME command, a data flow argument may be reset to a null string.

A dialogue box will be used to display the current value of the data flow argument, which the user may then amend.

REDRAW

The REDRAW command allows the user to respecify the position of one or more of the component line segments and the start and end points of a selected data flow.

The REDRAW command operates similarly to the DRAW command with the exception being that a data flow cannot be removed from the DFD using REDRAW. Upon selection of the REDRAW command, the last segment of the selected data flow is erased and the the remaining data flow line segments and cursor are redrawn as if the user

were in the process of drawing a new data flow.

The user may repeatedly press the UNSELECT mouse button to delete segments of the data flow path, as if drawing a new data flow. Unlike drawing a new data flow, the user is not able to delete the data flow by deleting all line segments in the data flow. The REDRAW command is terminated either when a valid data flow path is specified or the DELETE mouse button is pressed.

When using the REDRAW command, the user may delete all line segments comprising the selected data flow in order to select a new start position for the data flow: either a different DFD element, or a different point on the perimiter of the originally selected DFD element. In either case, if the user has deleted the entire data flow and then, instead of selecting a new start point for the data flow, presses the UNSELECT mouse button again, the REDRAW command will be cancelled and the data flow will be redrawn as it was before the command was selected.

If the selected data flow is named or has an argument label specified, then these labels will be erased from the screen and redrawn once the REDRAW operation is completed.

---

[4]Although the relevant data flow icon is not highlighted as would be the case if the user were drawing a new data flow.

COMMENT

The COMMENT command allows the user to specify a text string to be displayed in the bottom, comment portion of a named process bubble. The current value of the comment section is displayed in a dialogue box. Once displayed, the user may then edit the comment field.

EXPLODE

The EXLODE command allows the user to create or view (and subsequently amend) a child DFD for any process within the DFD being viewed.

If the process selected as the argument to the EXPLODE command does not have a child DFD, then the user will be prompted as to whether a child DFD should be created. Then, if the user confirms the creation of a child DFD, an empty DFD, with the same name as the selected process will be displayed and may be subsequently edited. The child DFD will automatically be linked to the parent process, unlike the ADD command in the DFD menu)

If the user does not want to create a DFD for the selected process, then the command is terminated and the originally selected process remains selected.

If the process selected as the argument to the EXPLODE command has a child DFD, then this DFD will be displayed and may be subsequently manipulated as for

any other DFD.

VIEW DD

The VIEW DD command allows the user to view and, if desired, subsequently edit the DD entry of a selected DFD element. The selected DFD element must be named in order to retrieve the associated DD entry.

Unlike the VIEW command in the DD ENTRY menu, the user does not have to specify the name of the entry to retrieve; rather, the user need only select a DFD element and then activate the VIEW DD command.

The DD entry window becomes selected and the originally selected DFD element is unselected as the result of the VIEW DD command.

DELETE

The DELETE command allows the user to delete the selected instance, all instances of the selected DFD element in the current DFD, all instances of the selected DFD element in the current DFD and all child DFDs, or all instances of the selected DFD element in all DFDs within the curent project. If a process, external entity or data store is deleted, all connected data flows are also deleted.

The user will be prompted, via a dialogue box, to select which of the above-named types of delete to per-

form. Only relevant options will be presented. For example, if the user has selected a data flow to delete and two instances of the data flow exist, both in the DFD shown on the screen, then the user will be prompted to select one of the following options: all instances in the current DFD, or selected instance only.

If the instance(s) of the selected DFD element to be deleted are the only occurrence(s) of the element within the current project, then the DD entry for the DFD element will be either erased from DD, or marked as unused (depending on the value of the DD DELETE system option).

DD entries of data items (that is, data element and data structure entries) used in the definition field of the DD entry of a DFD element to be deleted (and all elements contained in these definition fields, etc.) will be similarly deleted or marked as unused if the DD entry is found to be used only by those occurrences of the current DFD element to be deleted.

Data element and data structure entries subordinate to data flows which are no longer required as the result of deleting the data flow, are also either deleted or marked as unused (as described above). If a data flow consisting of one or more data flows, then the deletion of the parent data flow does not result in child data flows being deleted.

Deleted elements are removed from the affected DFDs and the current DFD is redraw. No DFD element remains selected after this operation.

The user will be required to confirm deletion of a process that has one or more child DFDs.

LINK

The LINK command allows the user to link a DFD (and any child DFDs) that is not the child DFD of any other process, to the selected process within the DFD being viewed.

The user will be prompted to specify the DFD to be linked to the selected process via a dialogue box. If the specified DFD is already the child DFD of another process, then the user will be given the opportunity of specifying another DFD to link to the selected process. The selected process cannot already have an associated child DFD and the selected DFD cannot have an associated parent process. The parent process must have be named before the LINK operation is carried out.

The child DFD(s) will be renamed and possibly renumbered by MUSSAT if the child DFD number differs from the number of the specified parent process.

UNLINK

The UNLINK command allows the user to split a hierarchy

of DFDs into two hierarchies by removing the association between a selected process and its child DFD(s). All parent process and child DFD relationships within the disconnected hierarchy are retained.

The user will be prompted to specify a new name for the disconnected child DFD via a dialogue box, since a DFD and process may only have the same name if they exist as child DFD and parent process (respectively). As with other operations where the user is prompted to supply a DFD name, a default DFD name will automatically be displayed which the user may either accept or change.

MOVE

The MOVE command allows the user to move any selected DFD object, other than a data flow, to any unoccupied space in the current DFD. All data flows connected to an element that is moved are 'rubber-banded'. Data flows cannot be moved independently of the DFD element(s) they are connected to.

As with creating a new DFD element, an existing element can only be moved to an unoccupied position the DFD. Any data flows connected to a DFD element will be rubber-banded, but new turning points will not be added as a result of moving the connected DFD element. For example, PROCESS A in figure V.2a could be moved to the

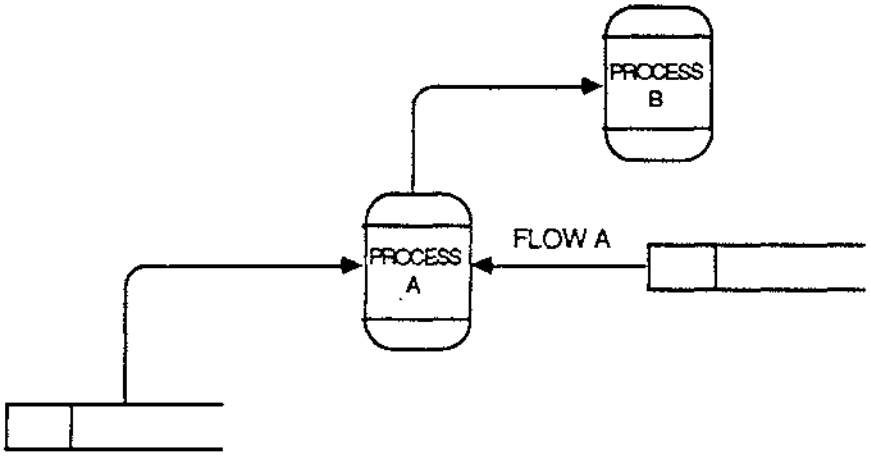


Figure IV.2a - Original position of PROCESS A.

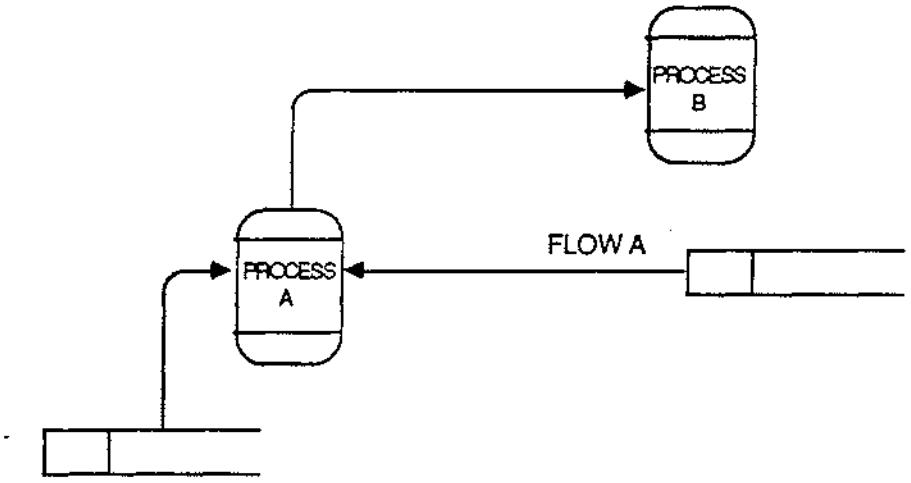


Figure IV.2b - Position of PROCESS A after having been moved.

left, as shown in figure IV.2b. However, PROCESS A could not be moved up or down since data flow FLOW A does not have a vertiacal line segment that can be rubber-banded.

The MOVE command does not appear in the OBJECT pull-down menu; MOVE is performed by dragging a selected element within the current viewing area using the SELECT mouse button.

MUSSAT will not let the user move a DFD element to a position where the selected object overlaps another object, or to a position where an external entity, process or data store becomes connected to a net flow into or out of the DFD.

Data flow name and search arguments may also be moved, however, placement of the data flow label is restricted to the immediate proximity of the associated data flow arc. This ensures that a data flow label remains next to its associated data flow and doesn't become 'lost' in the DFD. A data flow name or search argument label cannot be mvoed to a position occupied by another labe.

When a data flow is moved as the result of moving a connected process, data store or external entity, then all data flow labels are also moved, when necessary, in order to remain within the label space surrounding the data flow.

DD ENTRY Menu

Although MUSSAT maintains information about all DFD elements and data items within a MUSSAT project, the user is unable to view any information about unnamed DFD elements.

The user may view (and subsequently edit) a DD entry by activating one of the following commands:

```
VIEW;
NEXT;
PREVIOUS; or
DD VIEW.
```

The first three commands in the above list are DD ENTRY menu commands and are discussed below. The DD VIEW command is a command within the OBJECT menu and is discussed in the associated section.

VIEW

The VIEW command allows the user to retrieve any named DD entry in the project DD.

The user is prompted, via a dialogue box to specify the name of the DD entry to retrieve. The user may include the wildcard character ('*') at the end of a name, in which case the first entry found matching the partial string specified will be displayed. If the wildcard character is not specified MUSSAT attempts to find an entry with exactly the same name as specified. If a matching entry is not found then a dialogue box with a

message to the effect is displayed.

If no DD entry name is specified, then the first DD entry in alphabetic sequence is retrieved from the DD.

NEXT and PREVIOUS

The NEXT and PREVIOUS commands allow the user to retrieve the next highest and next lowest DD entries (respectively) in alphabetic sequence in relation to a selected DD entry currently displayed. These two commands are activated by selecting one of the two soft buttons, NEXT and PREVIOUS, at the bottom of the selected DD window.

In contrast to using the VIEW command, which causes a new DD Entry window to be displayed, the NEXT and PREVIOUS commands change the contents of the existing DD Entry window.

As with inactive options within the pull-down menus, the NEXT and PREVIOUS soft-buttons will be de-emphasised on the screen if the command is semantically invalid with respect to the contents of the selected DD entry window. For example, if the entry displayed in the selected DD Entry window is the first entry in the DD then the PREVIOUS soft button would be dimmed and selection of the PREVIOUS soft-button would not cause the contents of the DD Entry window to change.

CLOSE

The CLOSE command may be activated by either selecting the CLOSE command name in the pull-down menu, or by clicking in the close box in the corner of the selected DD entry window. Any selected DD entry window may be closed except when the entry is in edit mode.

CLOSE ALL

The CLOSE ALL command closes all DD entry windows, except, when open, the window of a DD entry being edited and the edit errors window.

DELETE

The DELETE command allows the user to delete individual DD entries marked as UNUSED from the project DD. The DD entry used as the argument to the DELETE operation will be deleted, regardless of the value of the DD DELETE system option so long as it is marked as UNUSED when the DELETE operation is activated. No other entries will be deleted, regardless of the value of DD DELETE.

The user will be required to confirm the execution of the DELETE command via a dialogue box. The DD entry will be removed from the display area and the data dictionary as a result of the DELETE operation.

EDIT

The user may edit the user maintained fields of a

selected DD entry using the EDIT command. An 'E' will be placed in the close box of the DD entry selected as the argument of the EDIT command, indicating the the entry is in edit mode and may not be closed. No other visible changes will be made to the DD entry window as a result of activating the EDIT command; all fields visible before EDIT was selected will remain visible, although the user will only be able to edit certain fields.

Only one entry at a time may be edited, although any number of DD windows may be opened, closed, moved and scrolled while the user is editing another entry. The entry being edited will retain the 'E' marker in the close box until the edit mode is exited, regardless of whether the DD entry remains selected.

If the user selects a DD entry to edit that has not yet been asigned a type[5], then the user will be required to select the type for the entry via a dialogue box before the entry may be edited. The user will be presented with a list of possible DD entry types that the selected entry may be given. This list will consist of the following types: data element, data

[5]A DD entry will have an unspecified type when the user has only included the DD entry name in the definition field of one or more other DD entries and the named DD entry has not been previously edited. See also the description of editing the DEFINITION field.

structure and data flow.[6]

The user may edit the following fields in a DD entry (although not all the fields listed below will be present in each DD entry):

ALIAS FOR

The DD element name in this field identifies the DD entry that contains the definition for the current DD entry, that is, it identifies the main entry of an alias DD entry.

If an alias name specified by the user is the name of an existing DD entry and the alias entry is of the same type as the current entry, then the current DD entry is linked to the main entry. The user cannot enter a definition in the definition field for an alias DD entry. The specified alias entry must already exist in the DD.

The main DD entry is updated so that the new alias is listed in the system maintained ALIAS OF field.

An alias cannot be specified for a DD entry with a non-empty definition field. Nor can the named alias be, itself, an alias of another DD entry.

---

[6]The data flow type can be specified for undefined components of a data flow entry, thereby creating a data flow DD entry without a data flow with the same name existing in any of the DFDs.

An alias field, once specified, may be reset to a null string.

NOTES, VOLUME, PROCESS/DATA FLOW TYPE, ORGANISATION

These are all text fields and have no associated semantics. These fields may be defined as any string of valid characters (the length of the string being implementation dependent) and once specified may be reset to a null string. A future enhancement may be to specify valid values or ranges of values (and to provide system checks for valid input) for the last three types of fields.

DEFINITION

Data element definitions are text only and may be edited in the same way as the above fields. A future enhancement may be to specify a language for the definition of data elements which may then be parsed.

Process definitions are predominantly text only with no associated semantics, however, there must be some way of identifying the names of sub-processes within the definition so that the correct DD links can be maintained. Identification of sub-processes is one of the tasks of the editor (and hence is not defined in this thesis). Nevertheless, a useful method may be to enclose

sub-process names in special character sequences such as * sub-process name * or ! sub-process name !.

Data structure, data stores and data flows are defined using the relational operators. Each element included in a definition must be of the appropriate type. For example, data structure and data store definitions may consist of the names of other data structures and data elements, data flow definitions may consist of the names of other data flow or data structure elements.

MUSSAT cannot immediately tell what type of DD entry to use for a specific data item contained in the definition of another data item since a DD definition may consist of more than one type of data item. For each element within a data item definition field MUSSAT either:

(1) links the existing DD entry with the same name (and a compatible type) to the current DD entry; or

(2) creates a new DD entry for the named item with type undefined.

Hence the user may also edit the type field of DD entries: by either specifying the entry type where

MUSSAT has automatically created the entry, or respecifying an existing entry type.

A definition may be reset to a null string, and hence returned to an undefined state, by erasing the definition field.

NAME

The name of any DD entry may be changed so long as the new name is unique and not used by any other DD entry (used or unused) or non-leaf-level process. If a renamed DD entry is used in the definition of one or more other DD entries, then these entries will also reflect the name change. The name of a DD entry may not be reset to null.

Other edit state operations

In addition to being able to edit the DD entry fields listed above, there are three further operations available to the user whilst editing a DD entry. These operations are described below:

RESTORE

The RESTORE command allows the user to replace the DD entry being edited with the most recently saved version of the entry, that is, the DD entry as it was immediately after the last successful SAVE or END EDITS command. If the user is editing the DD entry for the first time within the current MUSSAT

session and has not successfully issued the SAVE or END EDITS command during the current session, then the DD entry is replaced with the DD entry as it was imediately prior to the EDIT command being selected.

The RESTORE command allows the user to abandon edits carried out on a DD entry and hence return the DD entry to a consistent state, from which the END EDIT command can be successfully executed. Since the user cannot end an edit session while the DD entry being edited contains errors, this comand gives the user the option of abandoning the edits that gave rise to an error condition rather than attempting to correct the errors during the current edit session.

The user will be prompted to confirm the execution of the RESTORE command via a dialogue box.

SAVE EDITS

The SAVE EDITS command allows the user to commit the changes made to the DD entry being edited to the project DD. Upon selecting this command MUS-SAT checks the DD entry being edited for any semantic or syntactical errors introduced as a result of the changes the user has made to the DD entry.

If no errors are found, then the DD is updated, where required, to reflect the changes made to the DD entry. MUSSAT may need to update other visible DD entries or a DFD (if one is opened) as a result of the edits.

Other DD entry windows may be removed from the display area by MUSSAT as a result of the SAVE EDITS operation.

If, as a result of executing the SAVE EDITS command, errors are found to exist in the DD entry being edited, then the DD is not updated and an error dialogue box containing the message "Error(s) found in edited DD entry" (or similar) is displayed.

Once the user clicks the 'OK' button in the dialogue box, then another window is opened in which is displayed the error messages associated with the errors found in the edited DD entry. The user may move and scroll the edit errors window as with other DD entry windows, however, once the window is closed the user is unable to re-open it other than by selecting the SAVE EDITS command again.

END EDIT

The END EDIT command allows the user to finish editing and exit out of edit mode. This command

performs the same functions as the SAVE EDITS command except in the case where no errors are found in the DD entry being edited. In this case, after the DD is updated the edit session for the selected DD entry is terminated. The user cannot end an edit session while errors remain in the DD entry being edited.

SCROLL

The SCROLL command allows the user to view portions of a DD entry that not visible within the view shown in the DD window. A DD entry window is scrolled using the scroll bar mechanism introduced in section 5.2.1.5.

DFD REPORTS Menu

MUSSAT is able to produce four types of reports based on the contents of a (set of) DFDs. When a DFD name is required as a parameter for the report, the default value will be the name of the DFD specified in the TOP DFD system option.

These DFD reports are:

PRINTED DFD(S)

A printed copy of a (set of) DFDs, or all DFDs in the current project may be printed. The user is able to specify the starting DFD (by name) and the number of additional levels of DFDs to print. The user must specify a starting DFD name if TOP DFD does not have a

value. All subordinate DFDs are printed if the number of additional levels is not specified.

DFD STATISTICS

A DFD statistics report lists, for each DFD in a specified DFD hierarchy:

DFD name;

DFD number;

Date created;

Date last ammended;

the number of unique data flows;
data stores;
external entities;
processes in the DFD; and

prints a message indicating whether any unnamed DFD elements were ignored in the report.

The user may specify the starting DFD and the number of additional levels to be included in the report. If neither the starting DFD nor the number of levels is specified, all DFDs in the project, including any unlinked DFDs, are included in the report.

This report provides a means of determining: the number of DFDs in the DFD hierarchy, if any DFDs are unlinked and the level to which a DFD has been expanded, without having to display each DFD in the hierarchy.

UNNAMED ELEMENTS

DFD elements that have not been named by the user are

ignored in all other system generated reports (with the exception of data flows to or from data stores in a DFD BALANCING REPORT). This report lists the number of unnamed data flows, data stores, processes and external entities in each DFD within a specified range of DFDs. Since unnamed data flows to or from data stores may be valid, the number of these elements found is listed separately from other unnamed data flows.

As with the DFD statistics report, the user may specify a starting DFD and the number of additional levels of DFDs to include in the report. If neither the starting DFD name nor the number of levels are specified, then all DFDs in the project are included.

Since most unnamed DFD elements are ignored in all but one other system generated report, this report provides a way of finding out, in general terms, which DFD elements MUSSAT considers to be incompletely specified.

BALANCE DFD(S)

A DFD balancing report may be produced for any (sub) hierarchy of DFDs in the current project. The following information is produced for each DFD included in the report:

parent process name, number;

flows into and out of the parent process;

child DFD name and number; and

net flows into and out of the child DFD.

MUSSAT correctly identifies missing data flows that are shown as entering the parent process but not included in the child DFD. Data flows that are shown as parallel decompositions of flows into or out of the parent process, and missing components from these flows are also correctly identified.

The user may specify the name of the starting DFD and the number of additional levels of DFDs to include in the report. If TOP DFD is not specified then the user must specify the name of the starting DFD. If the number of levels is not specified, then all levels in the hierarchy are included in this report.

This report helps the analyst identify which DFDs may contain balancing errors.

UNDEFINED ELEMENTS

This report prints the names of all named DFD elements within a specified hierarchy of DFDs that have not been defined in the DD. A DD entry is undefined if the definition field is empty (other user defined fields may have been defined). Names are sorted alphabetically within DFD.

The purpose of this report is to allow the analyst to identify those DFD elements, within a given range of DFDs, that have not yet been defined or have had definitions erased. Undefined elements indicate that the DFD is not complete, either beause the analyst has been unable to, or has chosen not to, define all elements in the DFD.

DFD DD ENTRIES

This report contains DD entries corresponding to the DFD elements shown in a specified set of DFDs. The user may specify the starting DFD and the number of additional levels of DFDs to include in the report. As with the DFD based reports, if the starting DFD name is not specified then the TOP DFD is assumed: the starting DFD name must be specified if TOP DFD is undefined. If the number of additional levels is not specified then all subordinate DFDs are included. The user may specify the format (either full or summary) for the DD entries included in the report.

The DD describes, in greater detail, the elements contained within the DFDs of a system model and hence the information contained in the DD is essential to a detailed understanding of a system model. This report allows the user to produce the relevant DD entries to more fully describe a set of DFDs.

DD REPORTS Menu

For all DD based reports the user may choose between a full or summary format for the report. The option values are the same as those for the DD DISPLAY command discussed at the beginning of this section.

PRINTED DD

The user may request an alphabetic listing of (a subset of) the contents of the project DD. MUSSAT allows the user to specify the starting and ending entry names. The starting or ending entry name may include the wild-card character '*' at the end of the name. The wild-card character matches any character string, including a null string. For example, the user may specify that the ending entry of a report is "CUST*" in which case all DD entries starting with (and including) "CUST" will be included in the report.

If either (or both) the starting or ending entries are specified without a wildcard character and no DD entry is found with the specified name(s), then the user is informed that no match was found and may respecify one or both of the entry names. If the starting or ending entries are not specified then MUSSAT assumes the starting or ending entry to be the first or last entry (respectively) in the DD.

UNUSED ENTRIES

The user may request an alphabetic listing of (a subset of) all entries in the project DD marked as unused. As with the DD contents report, the user may specify the starting and ending entry names and the entry format (full or summary).

Even though the full format may be specified, the only system maintained information that may exist for an unused DD entry would be the COMPONENT OF field for data flow, data structure and data element entries. Hence there would not be much (if any) difference between the full and summary report formats for an UNUSED DD ENTRIES REPORT. The ability of the user to choose between output formats has been included in this report so that the parameters the user must specify for report generation are the same for all reports.

This report allows the produce a listing of all or some unused DD entries in order to decide which ones to keep and which ones to delete. This report also allows the user to see which DD entries would be deleted if the PURGE DD command was activated.

MAIN AND SUB ENTRIES

This report allows user to specify the name of an entry and MUSSAT then produces a listing of the named entry and all subordinate DD entries. The user may also

specify the entry format (full or summary) of the DD entries included in the report. This report is useful in identifying all component data items within a data flow, data structure or data store.

Appendix V

SSA Model of MUSSAT

1. The Data Model Diagram

Each of the entities in the Data Model Diagram is numbered. These numbers are used in the associated DD entries to provide a cross-refenence between logical entities in the Data Model Diagram and data structures in the DD.

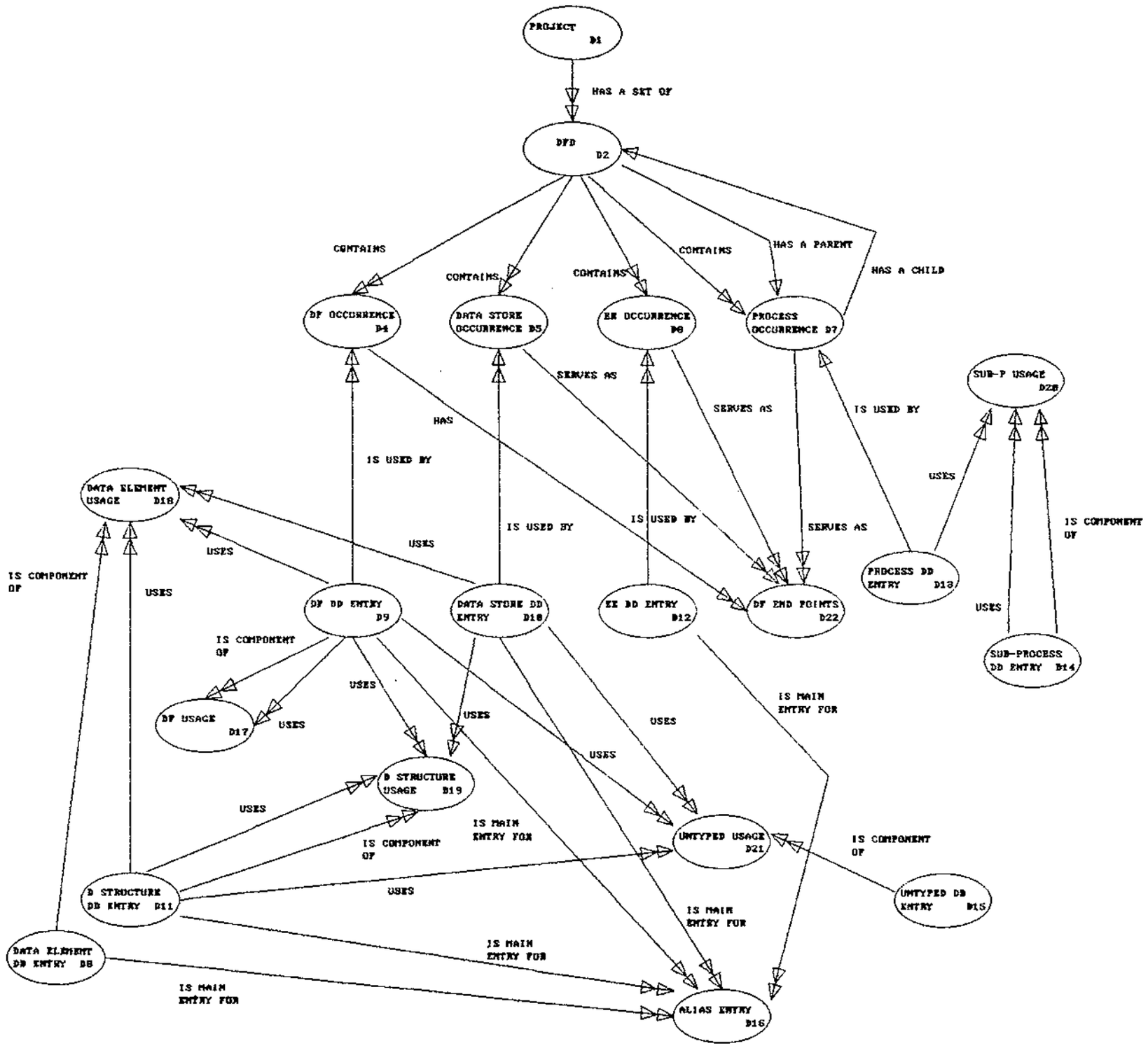
The software tool, EXCELERATOR, was used to draw both the Data Model Diagram and the DFDs. Due to name length limitations of EXCELERATOR some of the entity names in the Data Model Diagram had to be abbreviated and hence are not the same as the name used in the DD. Nevertheless, the entity number is the same in both the Data Model Diagram and the DD.

When appropriate, the logical entity number has been included in the NOTES section of data flow DD entries to identify which Data Model Diagram entity the stored or retrieved information is associated with.

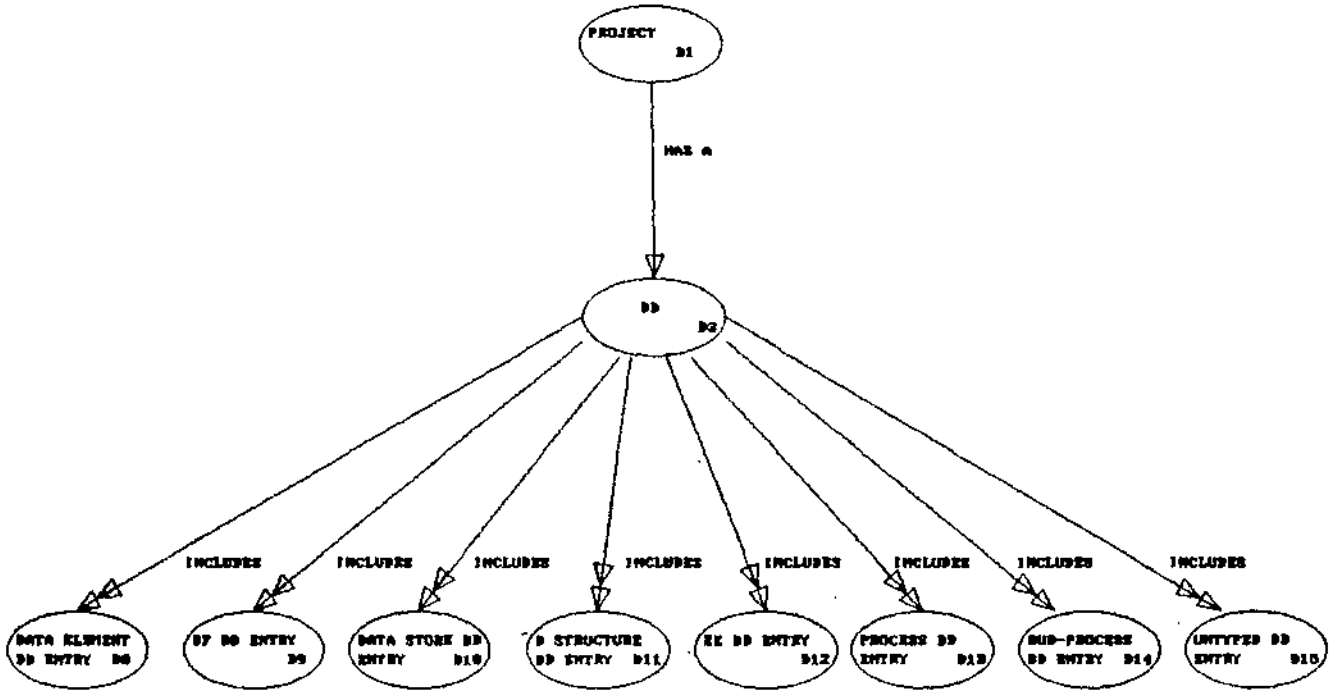
Although the Data Model Diagram is not part of the Structured Systems Analysis methodology, it was used to show the data entities and relationships required in the MUSSAT database. Neither the Demarco or Gane & Sarson variations of SSA include techniques suitable for modelling complex data

structures. DeMarco's Data Structure Diagrams are only useful for simple data structures and Gane & Sarson's Data Immediate Access Diagrams are not useful as, nor were they intended to be used as a data design tool.

The Data Model Diagram is split into two diagrams since the entire diagram is too large to fit on one page. The first diagram shows all but one of the entities required in the MUSSAT database. The second diagram shows the entity 'DD' and the associated relationships to other MUSSAT entities. Those entities shown in the first diagram that are related to 'DD' are also shown in the second diagram.



DATA MODEL DIAGRAM OF THE MUSSAT DATABASE



DATA MODEL DIAGRAM OF THE MUSEUM DATABASE: ENTITY D2

2. The Data Flow Diagrams

The DFDs shown in this section conform to the MUSSAT SSA rules discussed in Chapter 4 and summarised in Appendix II. In addition to these rules, several additional data flow naming conventions were used in the DFDs in an attempt to make the DFDs easier to understand. The additional conventions adopted are as follows:

(1) Data flows into a data store that contain new information that does not already exist in the data store contain "NEW" as the first word in the data flow name.

(2) Data flows into a data store that contain information that was retrieved, updated and replaced in the data store contain "UPDATED" as the first word in the data flow name.

(3) Data flows into a data store that contain information that already exists and is to be deleted from the data store contain "DELETED" as the first word in the data flow name.

(4) Data flows from a data store are named with a description of the contents of the data flow.

Retrieval keys for data flows out of data stores are not shown since EXCELERATOR does not suppport either the definition of display of retrieval keys.

Minor data outputs are not shown in the DFDs. These minor data outputs comprise error data flows and changes to the viewing area of the VDU.

Error flows are not shown because they tend to clutter up a DFD. As discussed in Appendix III, dialogue boxes are used in MUSSAT to inform the user of any input errors. Where appropriate, the use of error or prompt dialogue boxes is included in the relevant process or sub-process definition rather than showing the dialogue boxes as data flows out of a process.

Changes to the viewing area, as seen by the user are not shown as data flows because manipulation of the MUSSAT project by the user is considered to be an intermediate step necessary to produce the MUSSAT report; only the MUSSAT reports are shown as net flows out of MUSSAT. Also, it is difficult to construct data structures that accurately and concisely represent what is shown on the screen.

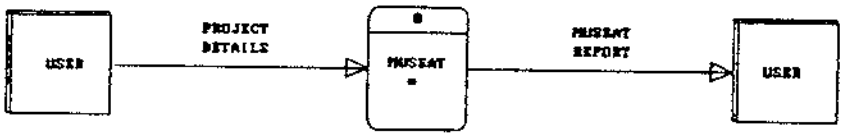
The SSA model of MUSSAT also does not show the interactive graphical operations necessary to create a visual representation of the data stored in the MUSSAT database for two reasons:

The operations required to create the graphical representation of an object are implementation dependant; and

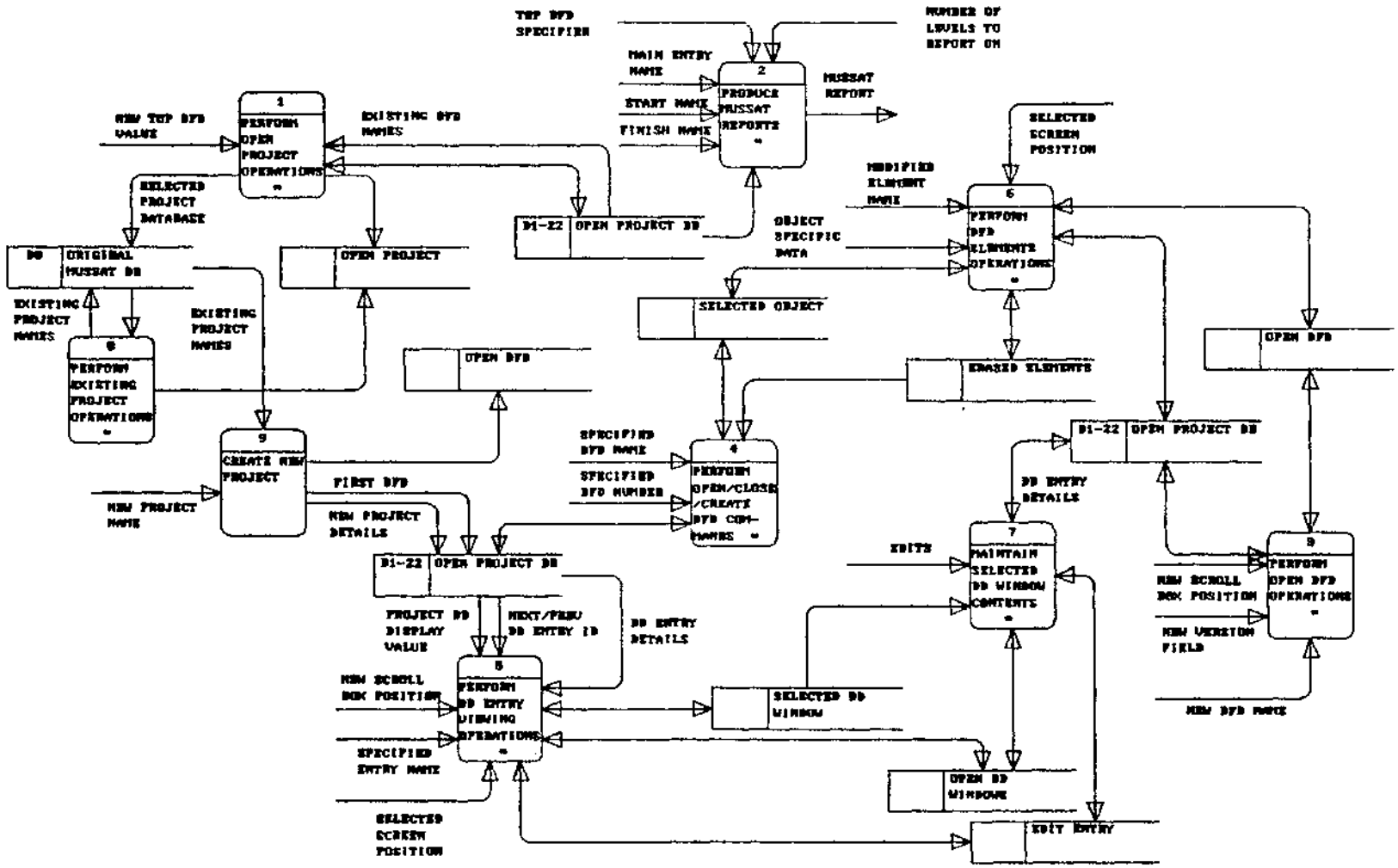
even if graphical operations could be described in a high level language it would not be appropriate to include such details in a logical model.

In addition, no attempt has been made in the DD to specify the application data structures that will be required to implement the MUSSAT user interface. Graphical application data structures are complex and implementation dependent. The way in which a graphical model is represented and manipulated in an application is affected by the graphics and data management facilities available in the operating environment of the application.

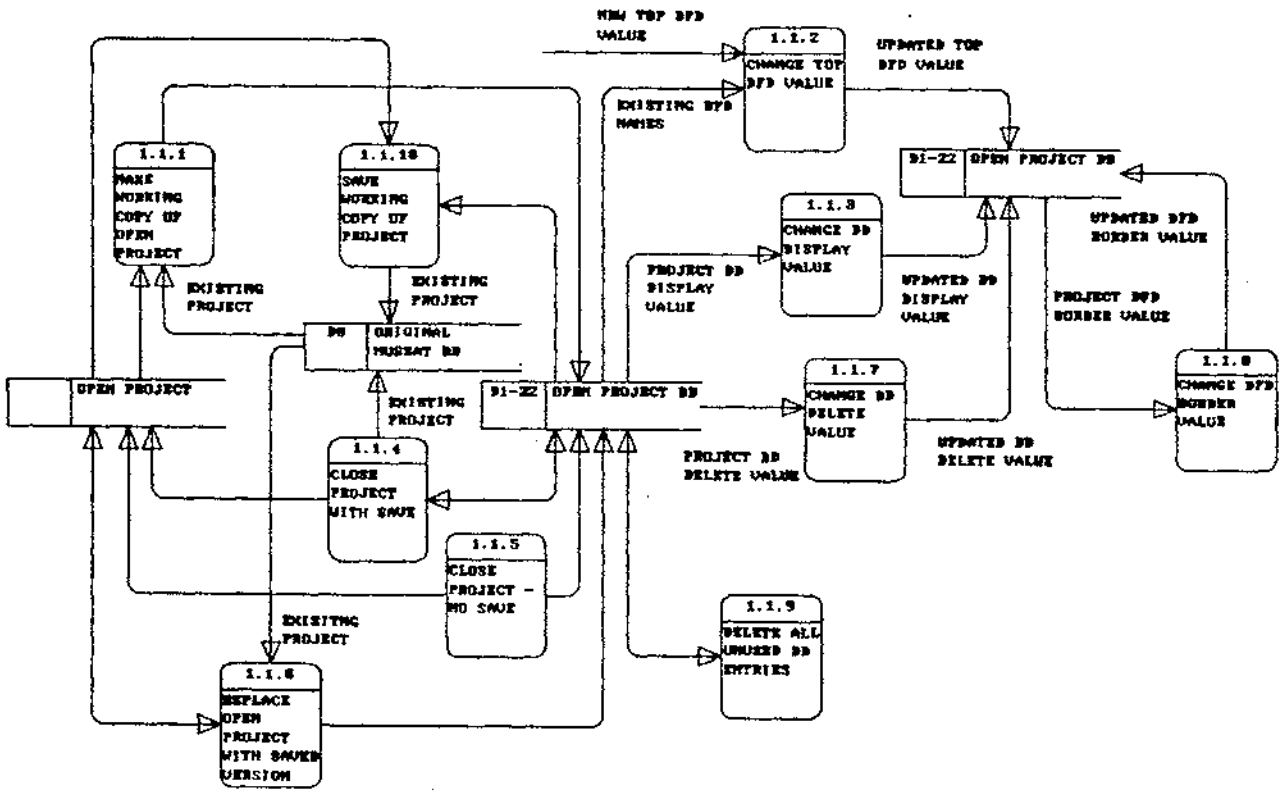
It is possible that the MUSSAT database could be modified so that both the analysis and graphical information could be stored together in the MUSSAT database, in which case a Relational Database Management System would manage all MUSSAT data ([FOL82] PP. 359-362). Alternatively, the graphical model could be represented in a separate data structure, which the application program would be responsible for managing. The choice of which representation to use cannot be made until a proposed operating environment is chosen.



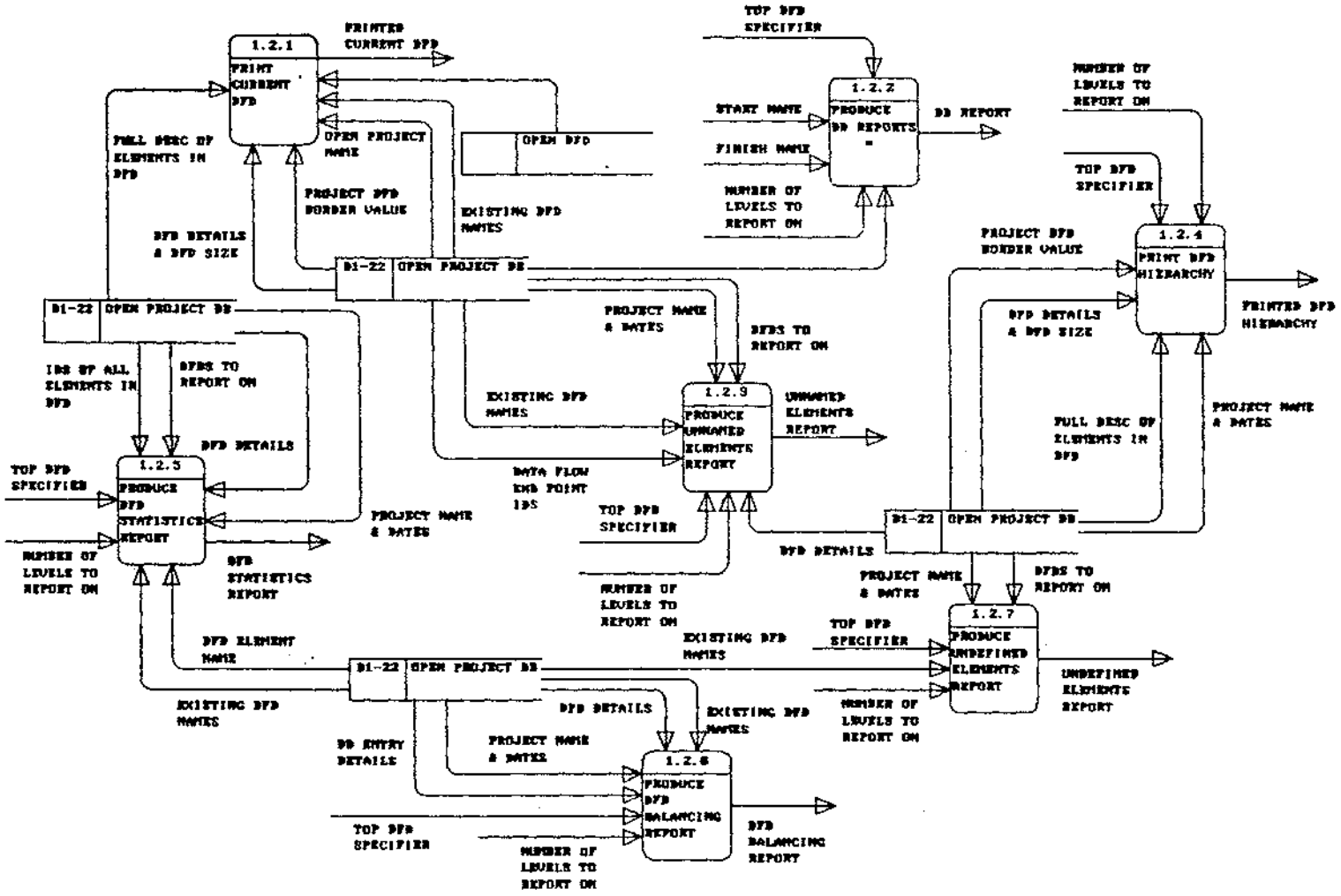
DFD 0: CONTEXT DFD


TOP DFD SPECIFIER
NUMBER OF LEVELS TO REPORT ON
MAIN ENTRY NAME
START NAME
FINISH NAME
2
PRODUCE MUSSAT REPORTS
MUSSAT REPORT
1
PERFORM OPEN PROJECT OPERATIONS
NEW TOP DFD VALUE
EXISTING DFD NAMES
SELECTED PROJECT DATABASE
D1-22 OPEN PROJECT DB
DB ORIGINAL MUSSAT DB
OPEN PROJECT
EXISTING PROJECT NAMES
EXISTING PROJECT NAMES
8
PERFORM EXISTING PROJECT OPERATIONS
OPEN DFD
SELECTED SCREEN POSITION
MODIFIED ELEMENT NAME
6
PERFORM DFD ELEMENTS OPERATIONS
OBJECT SPECIFIC DATA
SELECTED OBJECT
ERASED ELEMENTS
OPEN DFD
9
CREATE NEW PROJECT
NEW PROJECT NAME
FIRST DFD
NEW PROJECT DETAILS
SPECIFIED DFD NAME
SPECIFIED DFD NUMBER
4
PERFORM OPEN/CLOSE /CREATE DFD COPY-NAMES
D1-22 OPEN PROJECT DB
DB ENTRY DETAILS
D1-22 OPEN PROJECT DB
PROJECT DB DISPLAY VALUE
NEXT/PREV DB ENTRY ID
DB ENTRY DETAILS
EDITS
7
MAINTAIN SELECTED DB WINDOW CONTENTS
NEW SCROLL BOX POSITION
NEW VERSION FIELD
9
PERFORM OPEN DFD OPERATIONS
NEW DFD NAME
NEW SCROLL BOX POSITION
5
PERFORM DB ENTRY VIEWING OPERATIONS
SELECTED DB WINDOW
SPECIFIED ENTRY NAME
OPEN DB WINDOWS
SELECTED SCREEN POSITION
EDIT ENTRY

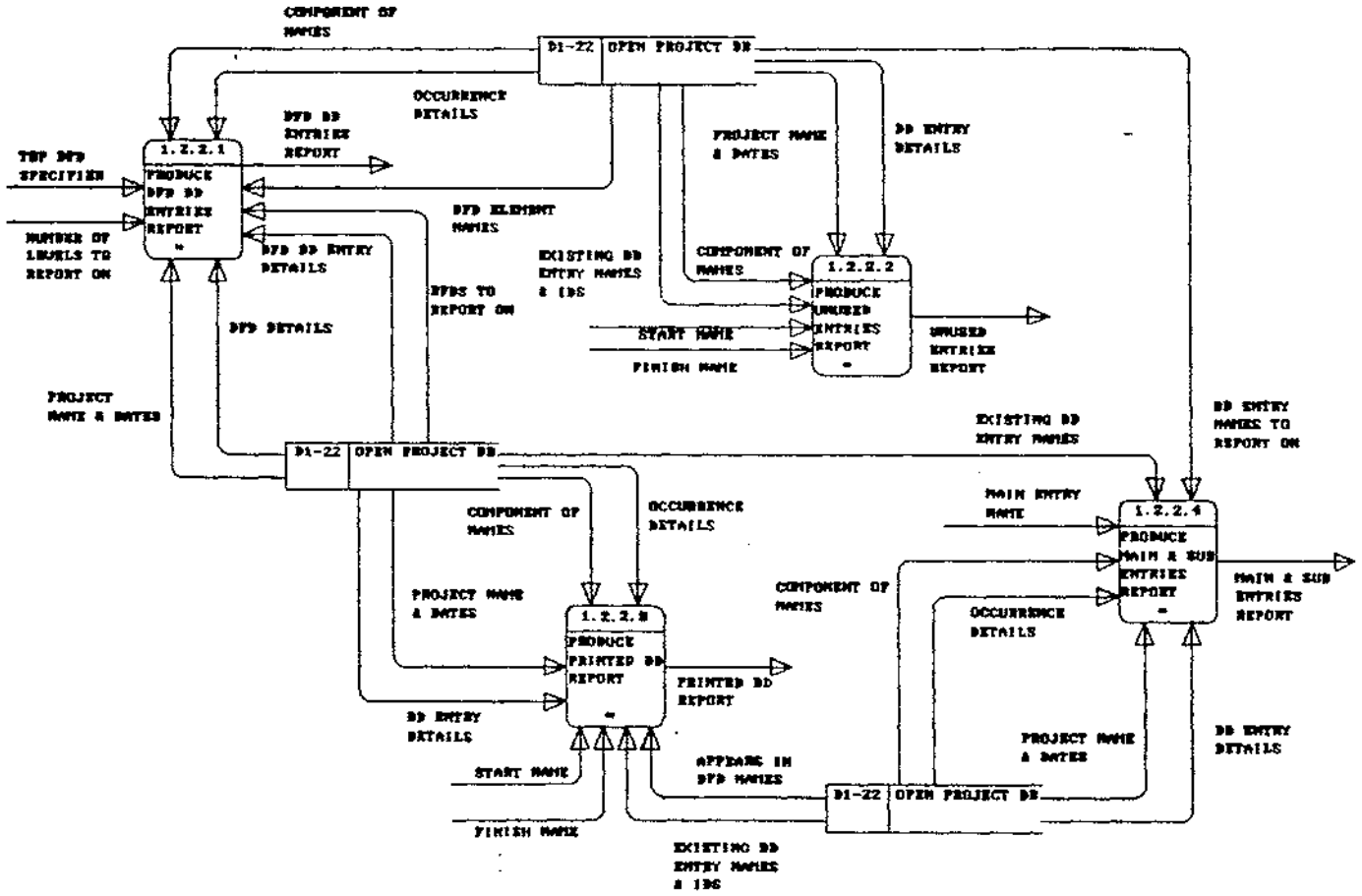
DFD 1: MUSSAT



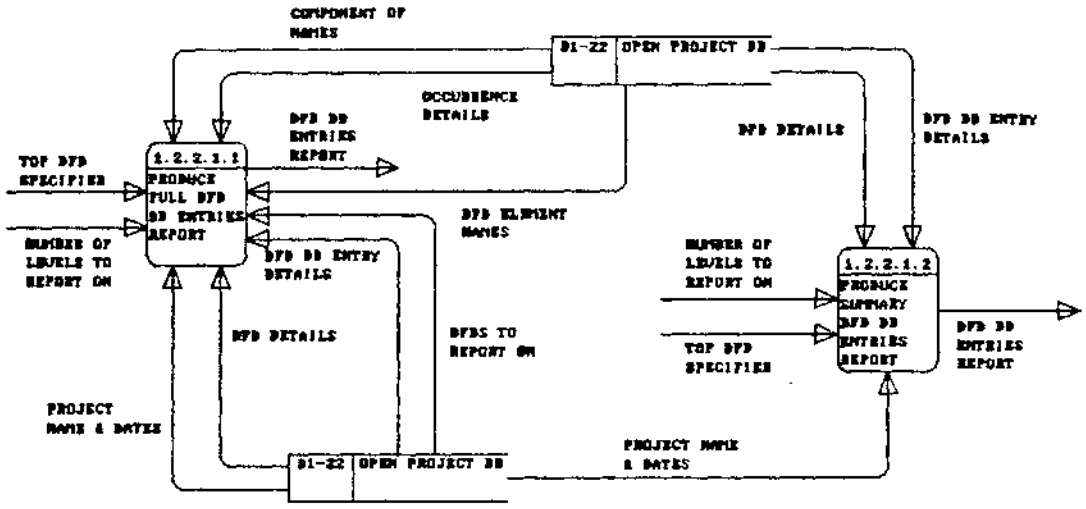
DFD 1.1: PERFORM OPEN PROJECT OPERATIONS



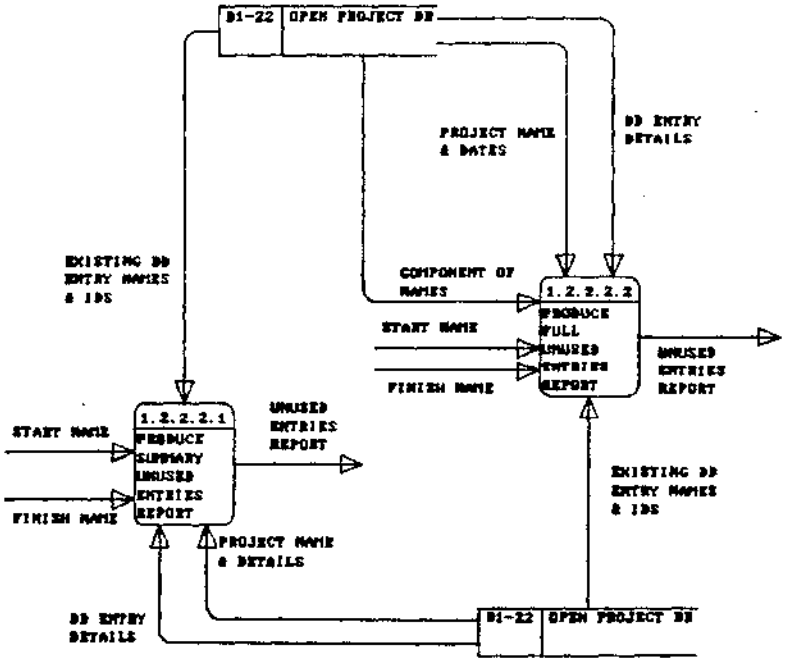
DFD 1.2: PRODUCE PRESENT REPORTS



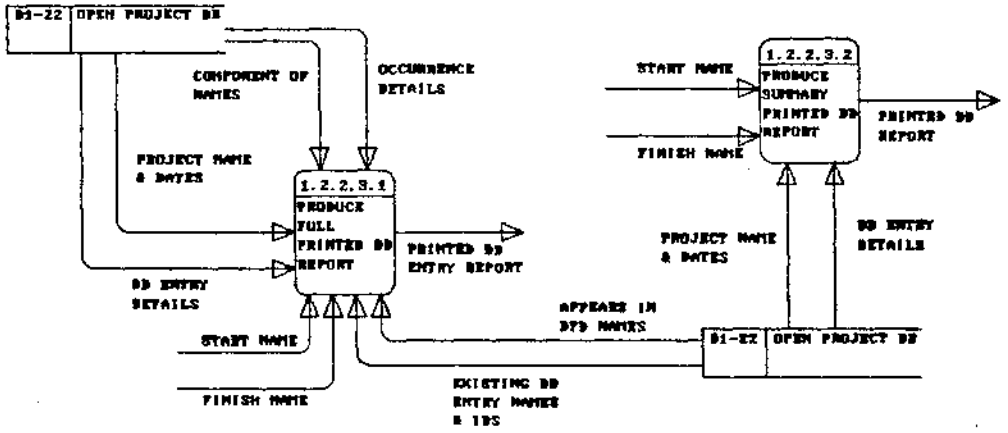
DFD 1.2.2: PRODUCE DD REPORTS



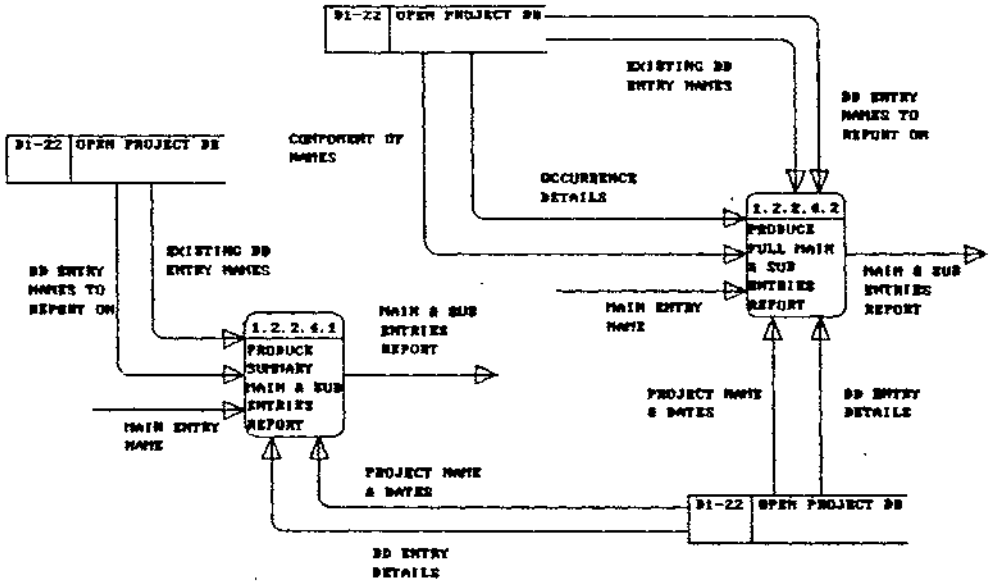
DFD 1.2.2.1: PRODUCE DFD DB ENTRIES REPORT



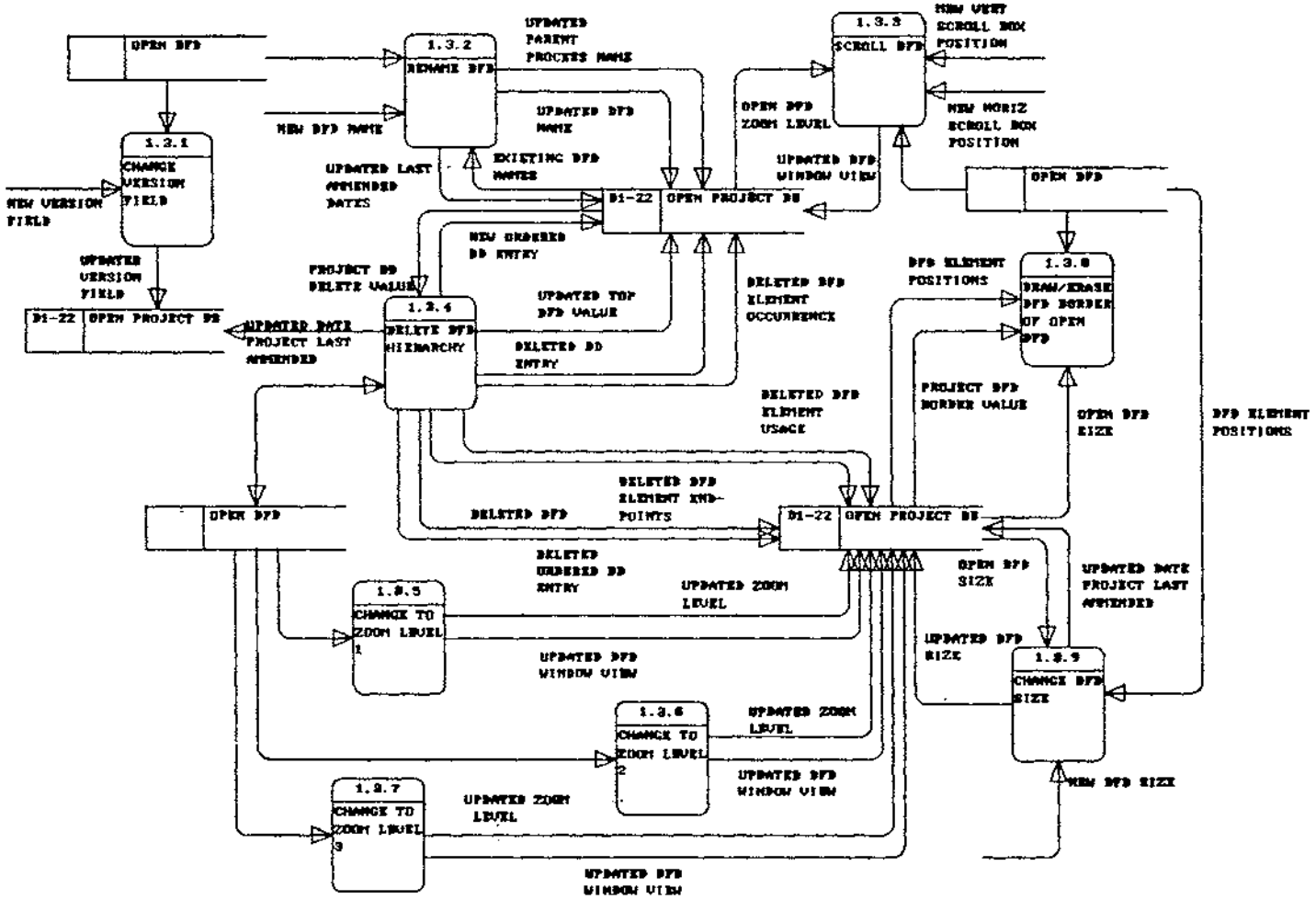
DFD 1.2.2.2: PRODUCE UNUSED ENTRIES REPORT



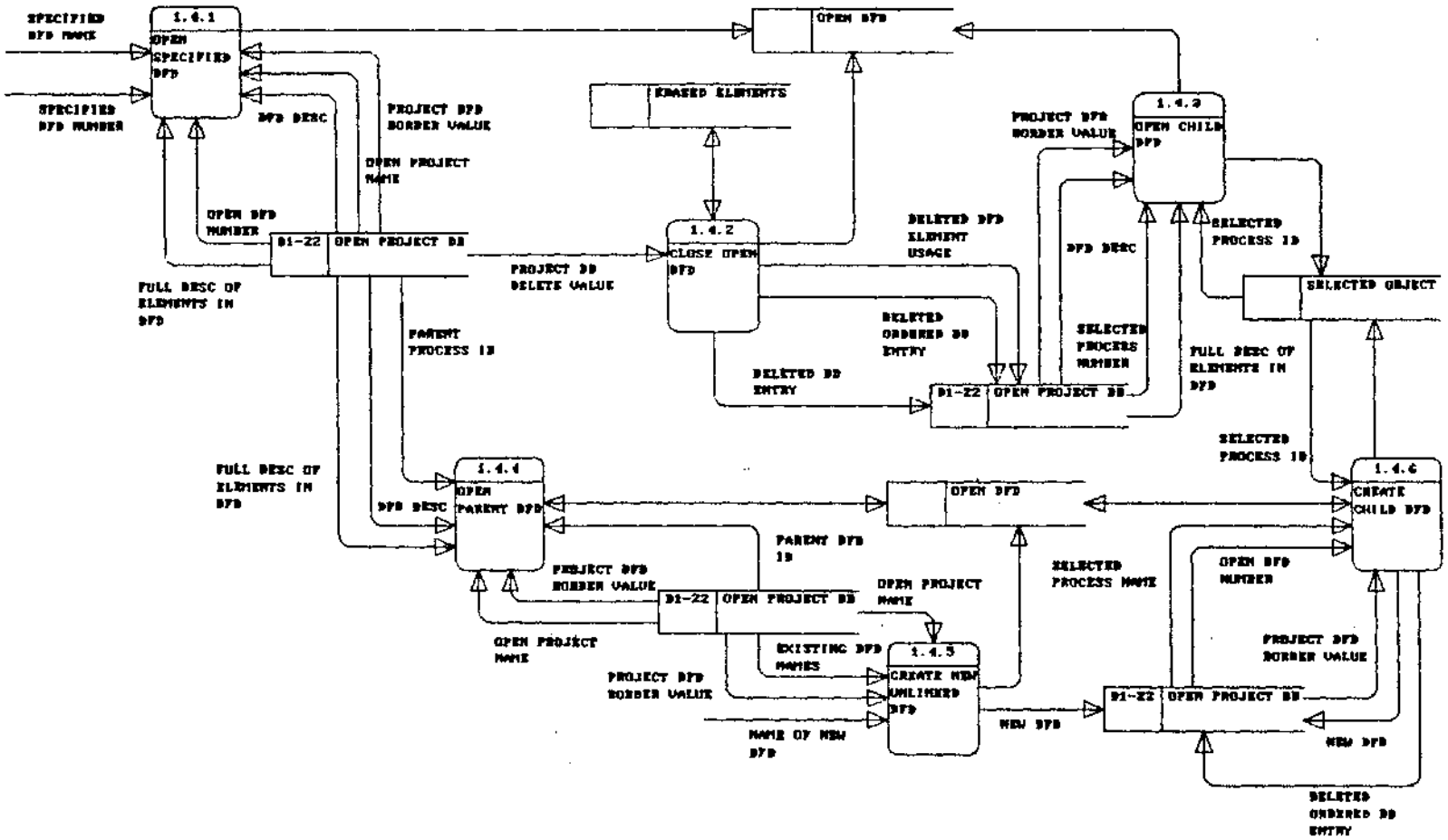
DFD 1.2.2.3: PRODUCE PRINTED DD REPORT



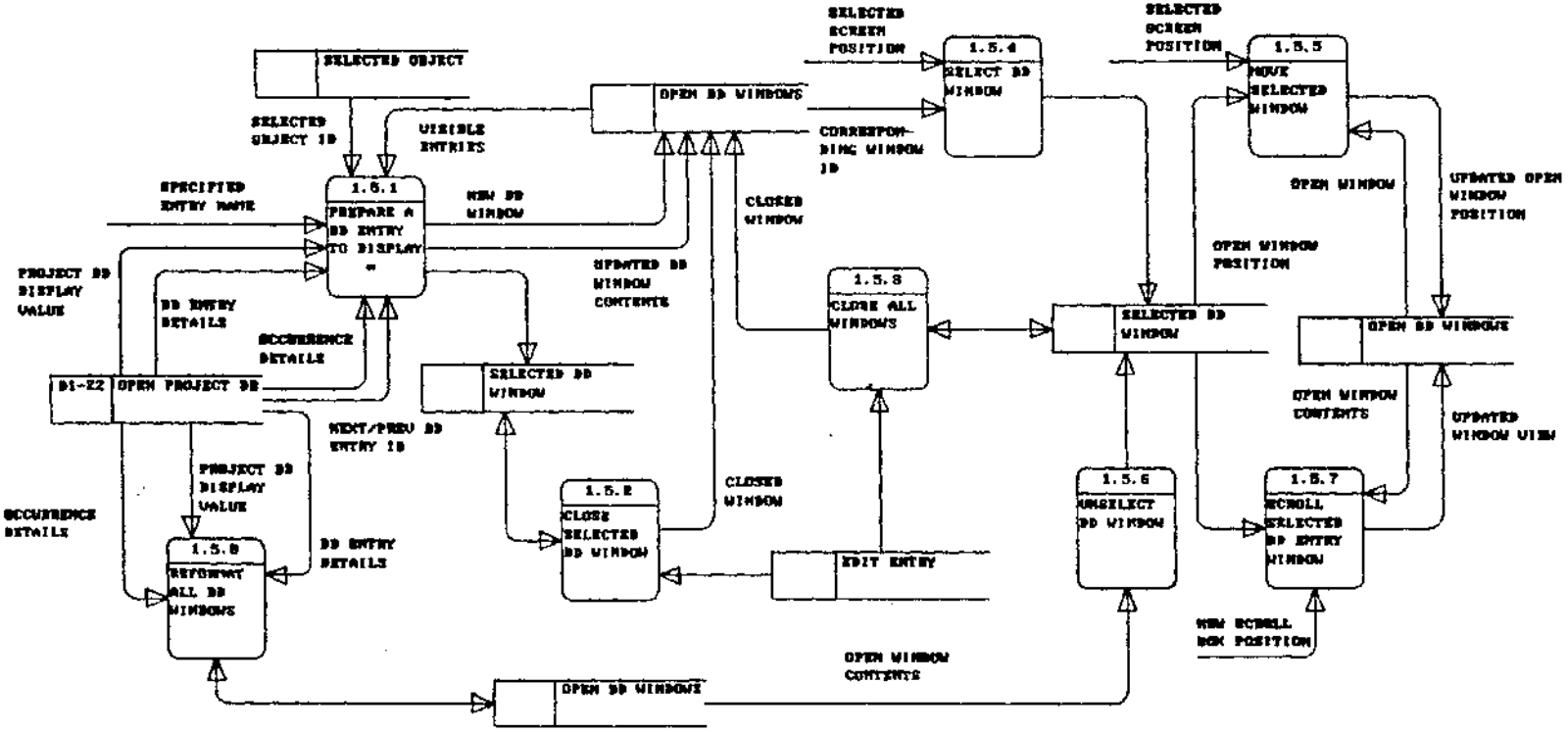
DFD 1.2.2.4: PRODUCE MAIN & SUB ENTRIES REPORT



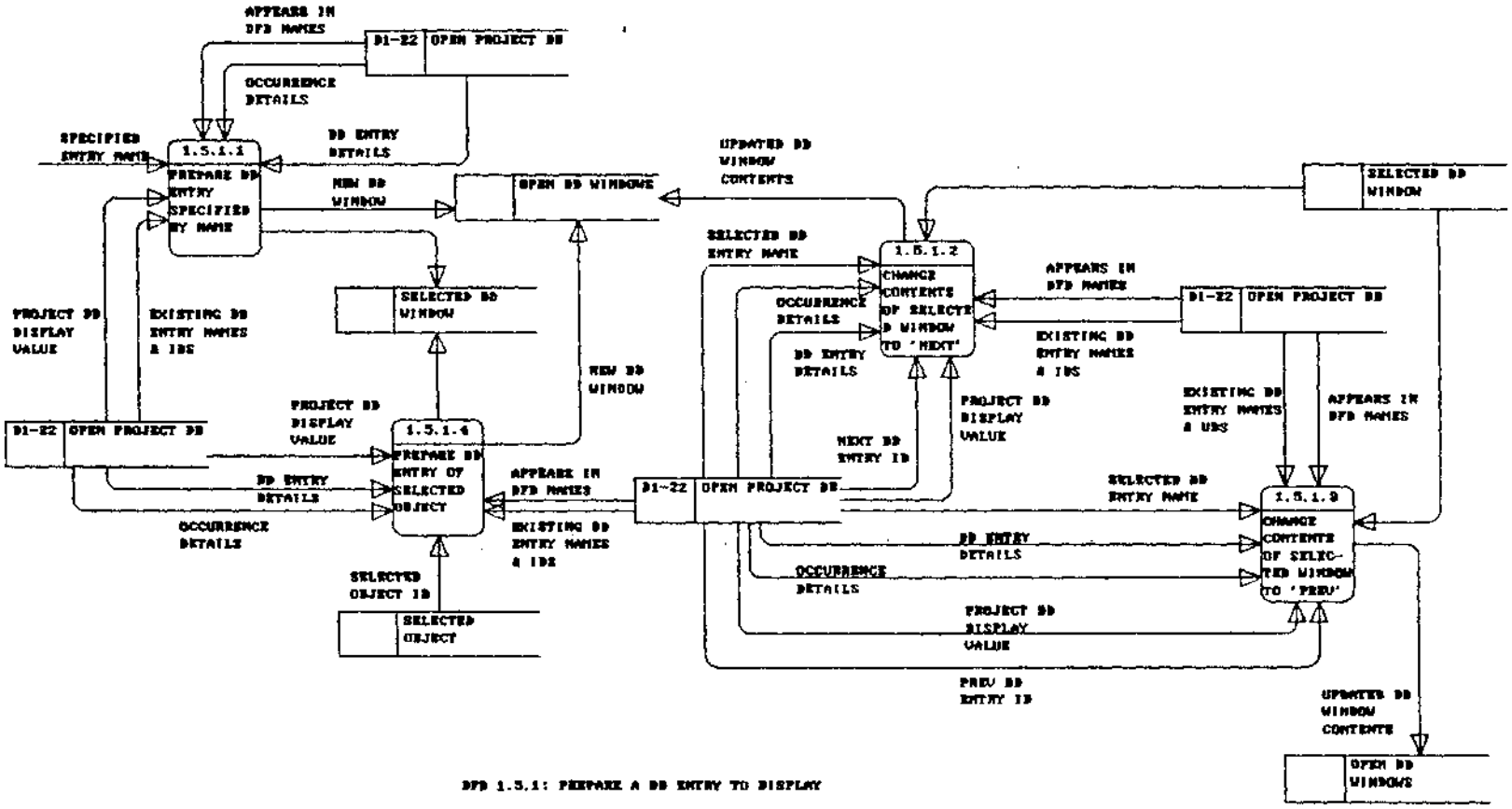
DFD 1.3: PERFORM OPEN DFD OPERATIONS



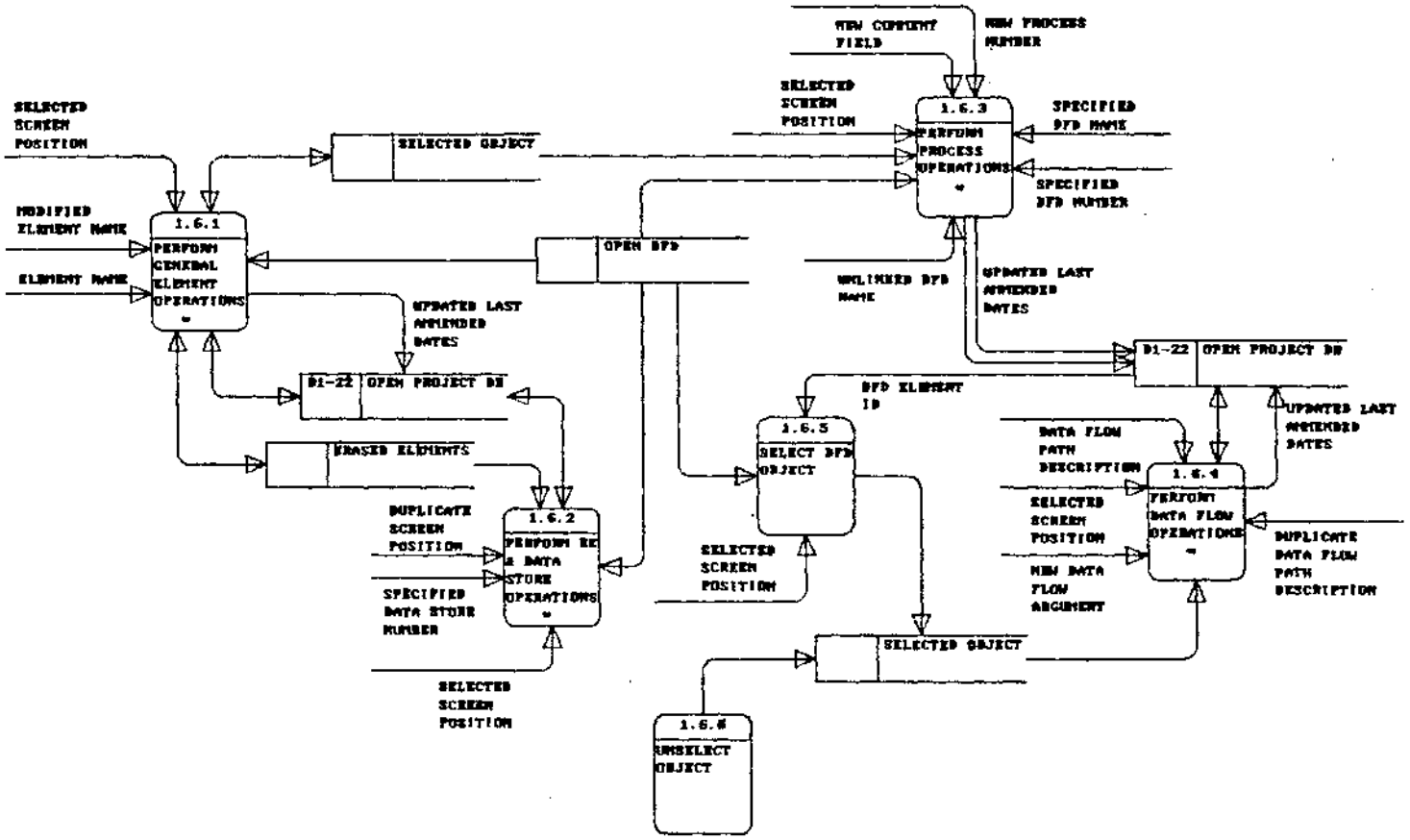
DFD 1.4: PERFORM OPEN/CLOSE/CREATE DFD COMMANDS



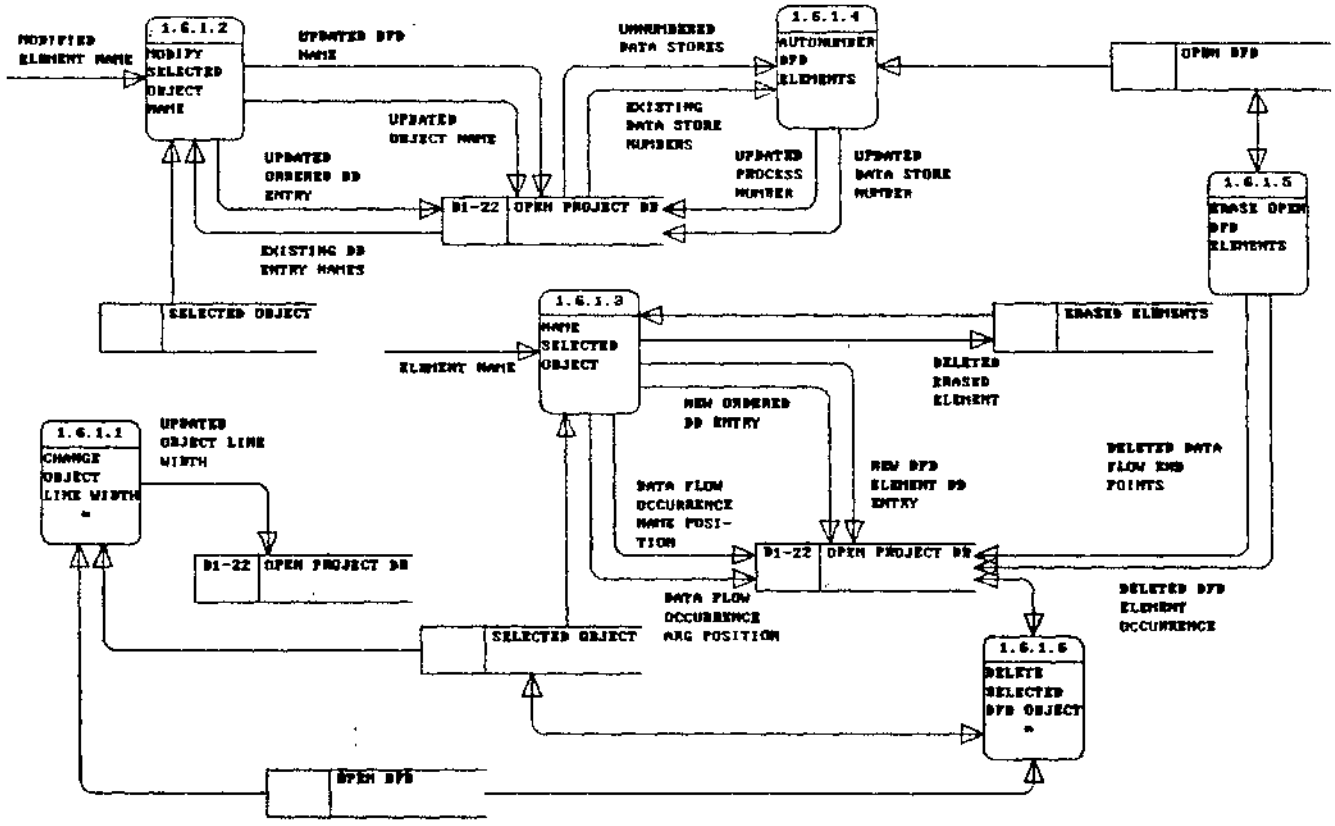
DFD 1.5: PERFORM DD ENTRY VIEWING OPERATIONS



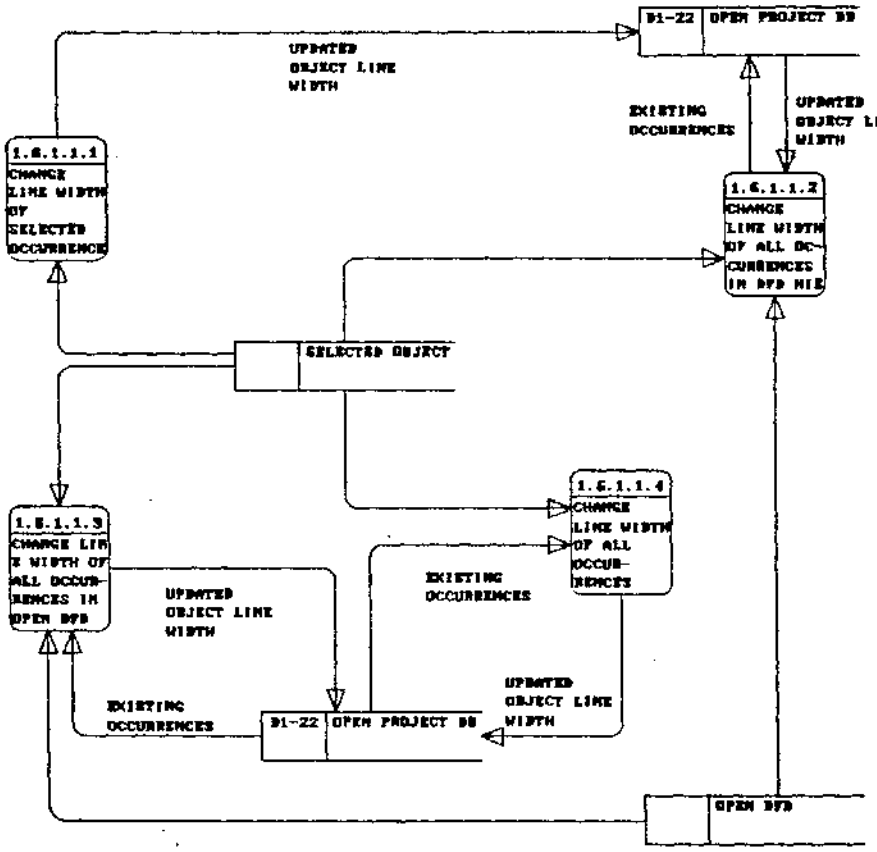
DFD 1.5.1: PREPARE A DB ENTRY TO DISPLAY



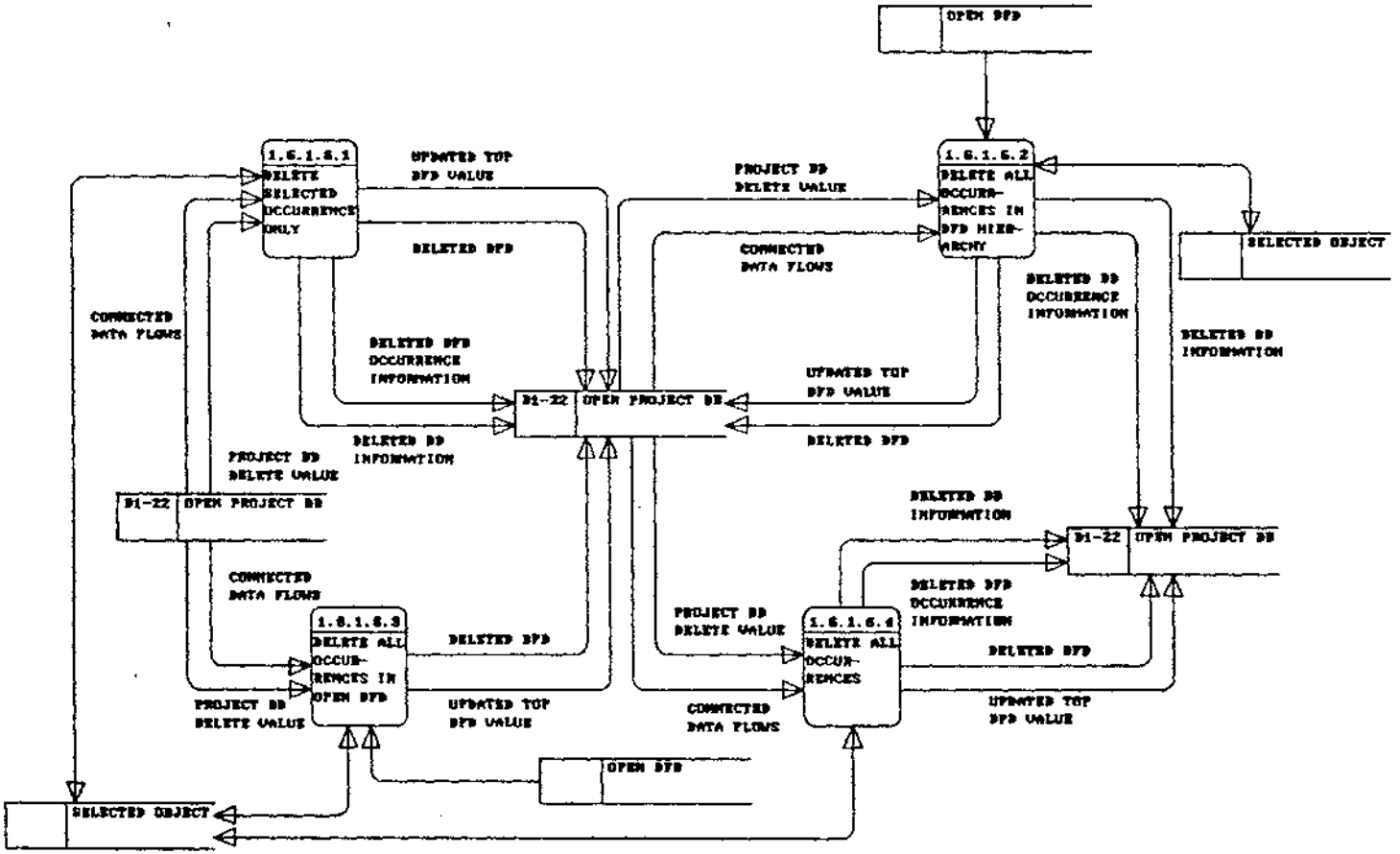
DFD 1.6: PERFORM DFD ELEMENTS OPERATIONS



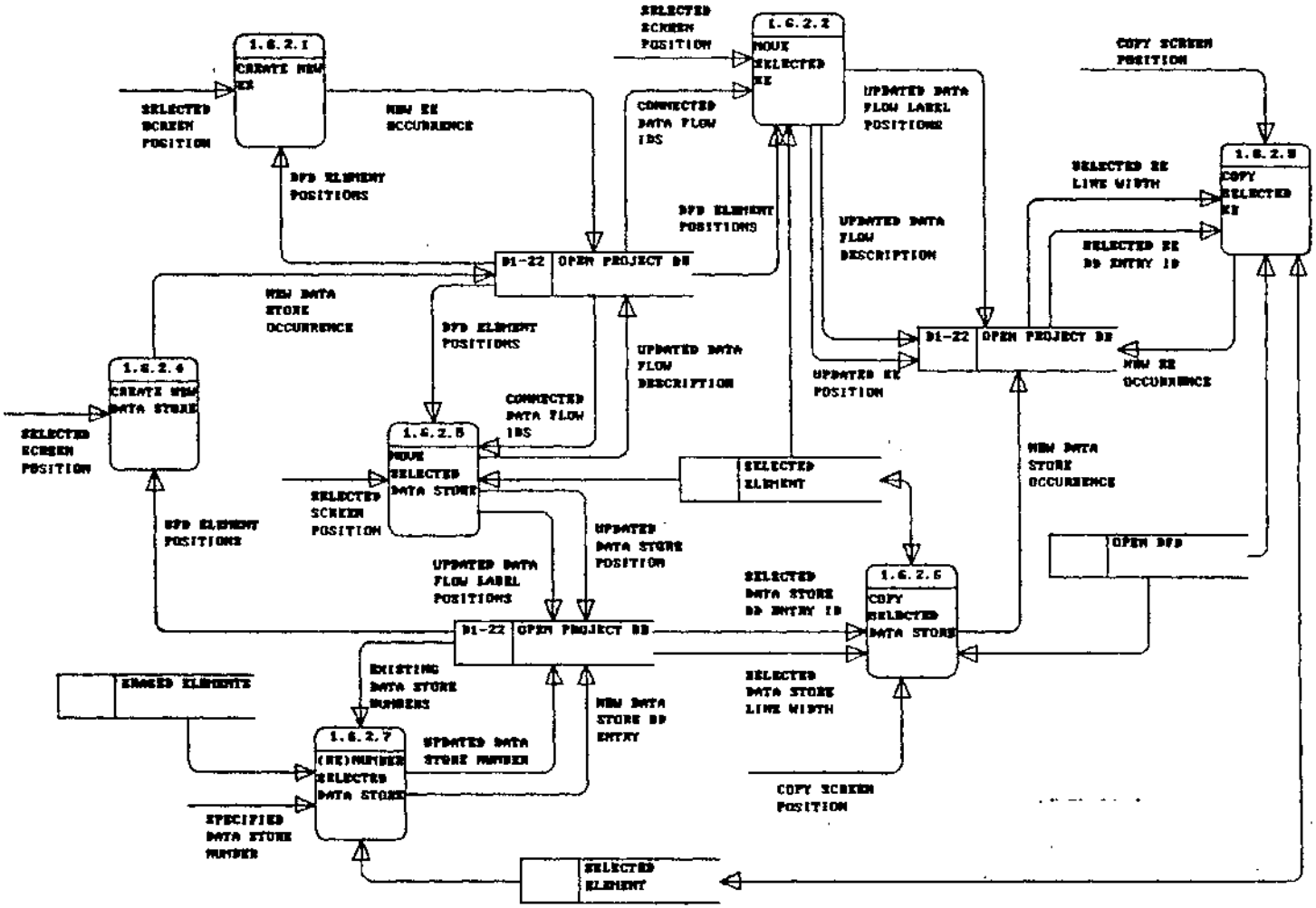
DFD 1.6.1: PERFORM GENERAL ELEMENT OPERATIONS



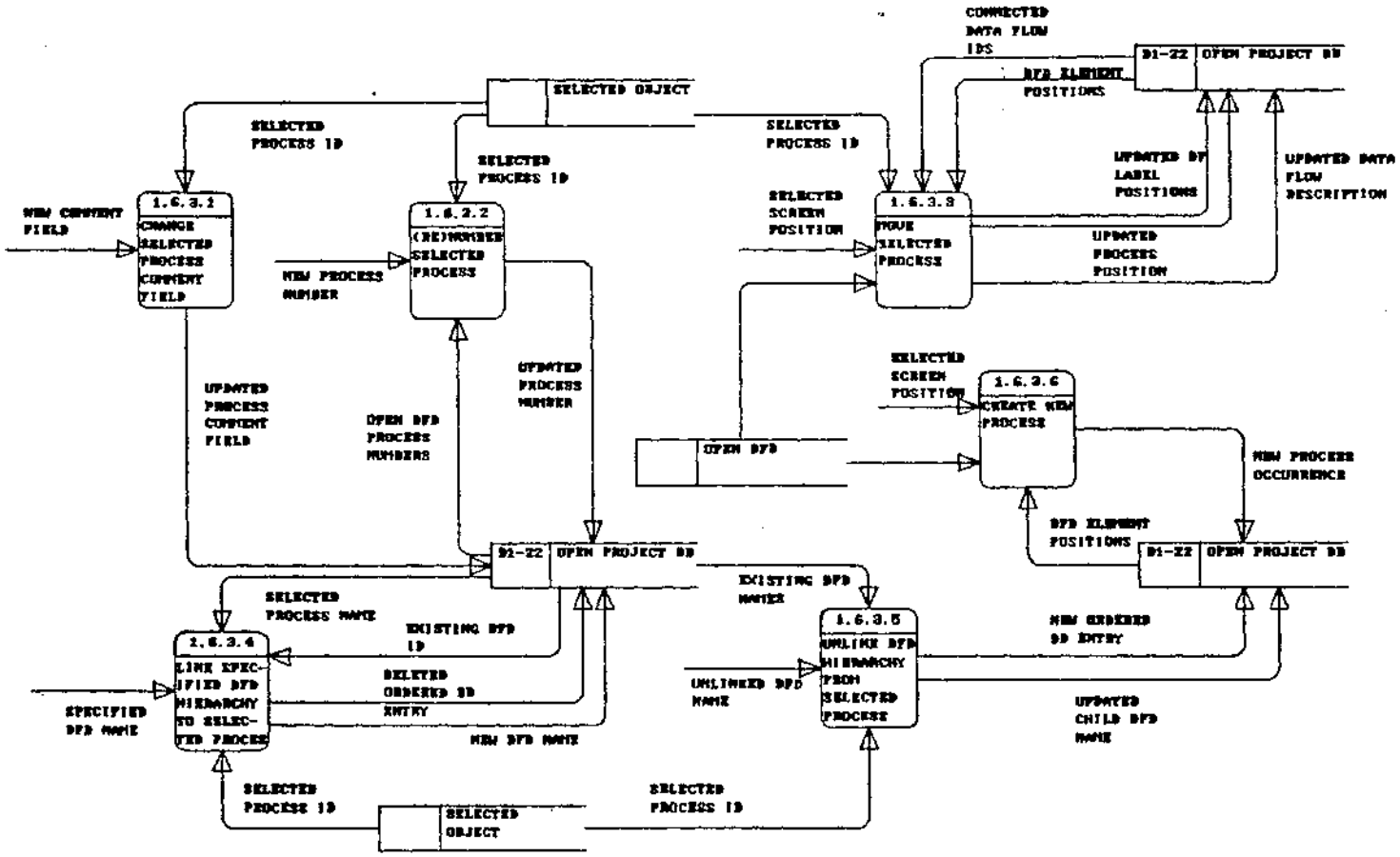
DFD 1.6.1.1: CHANGE OBJECT LINE WIDTH



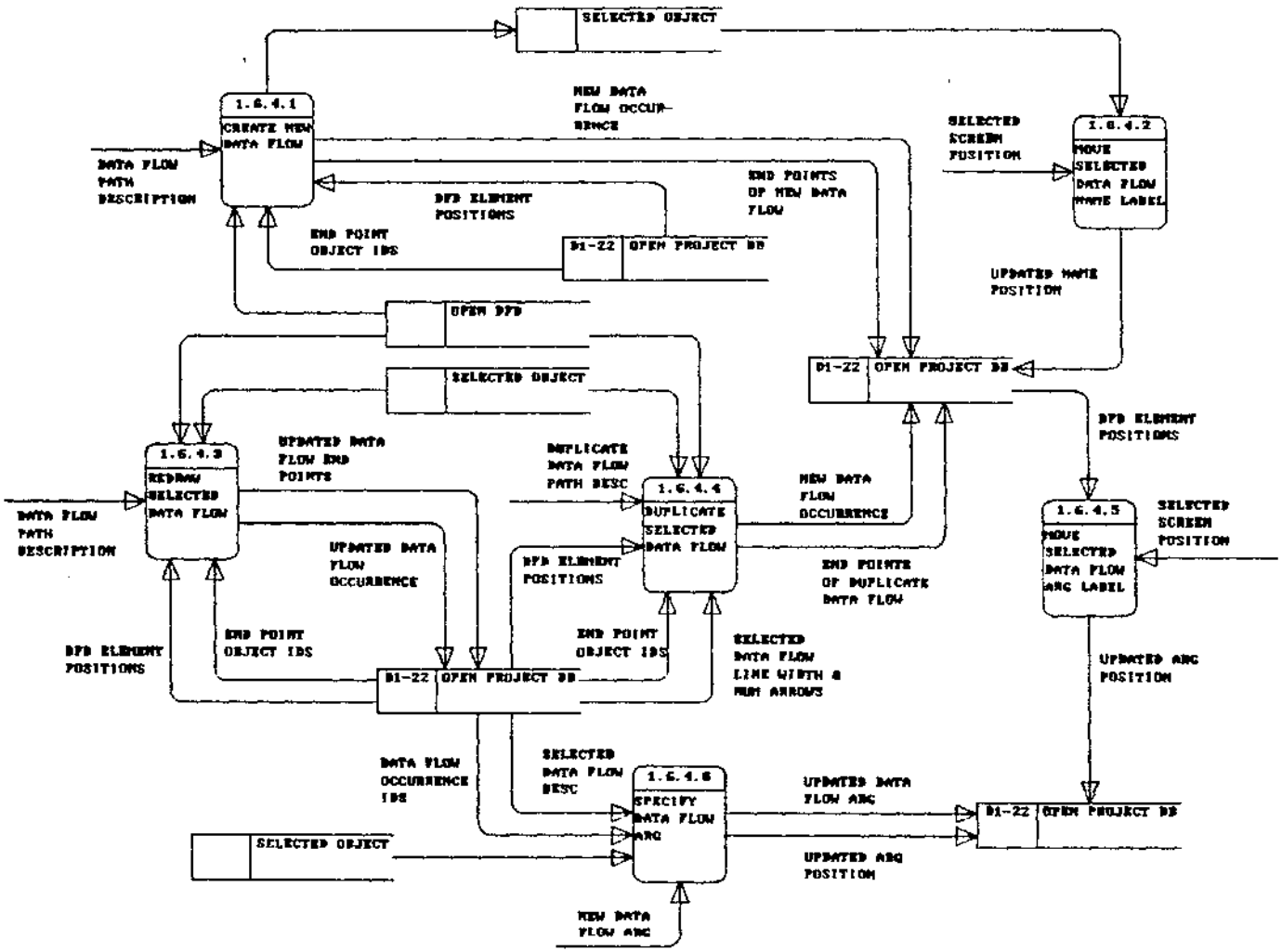
DFD 1.6.1.6: DELETE SELECTED DFD OBJECT



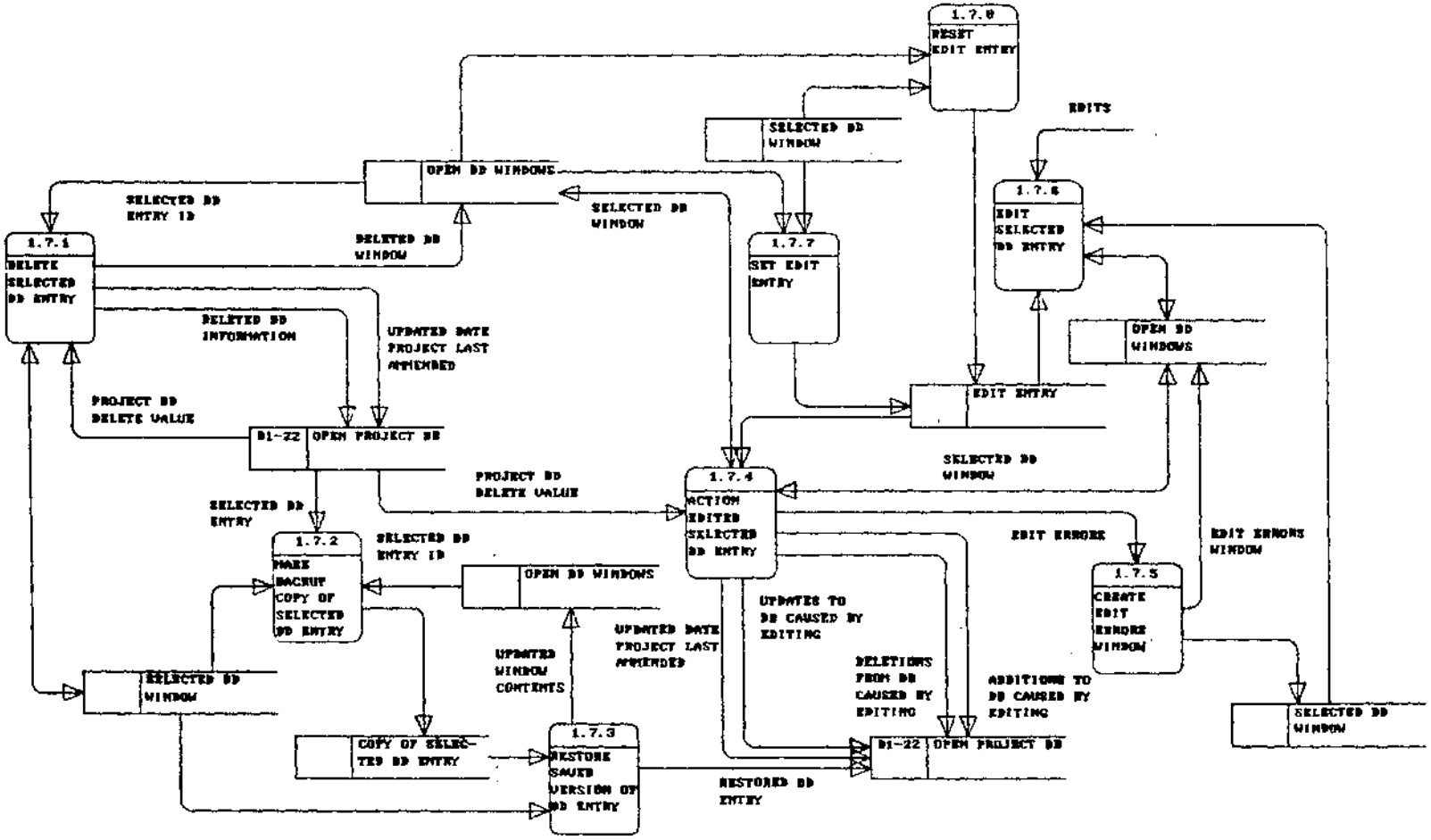
DFD 1.6.2: PERFORM RE & DATA STORE OPERATIONS



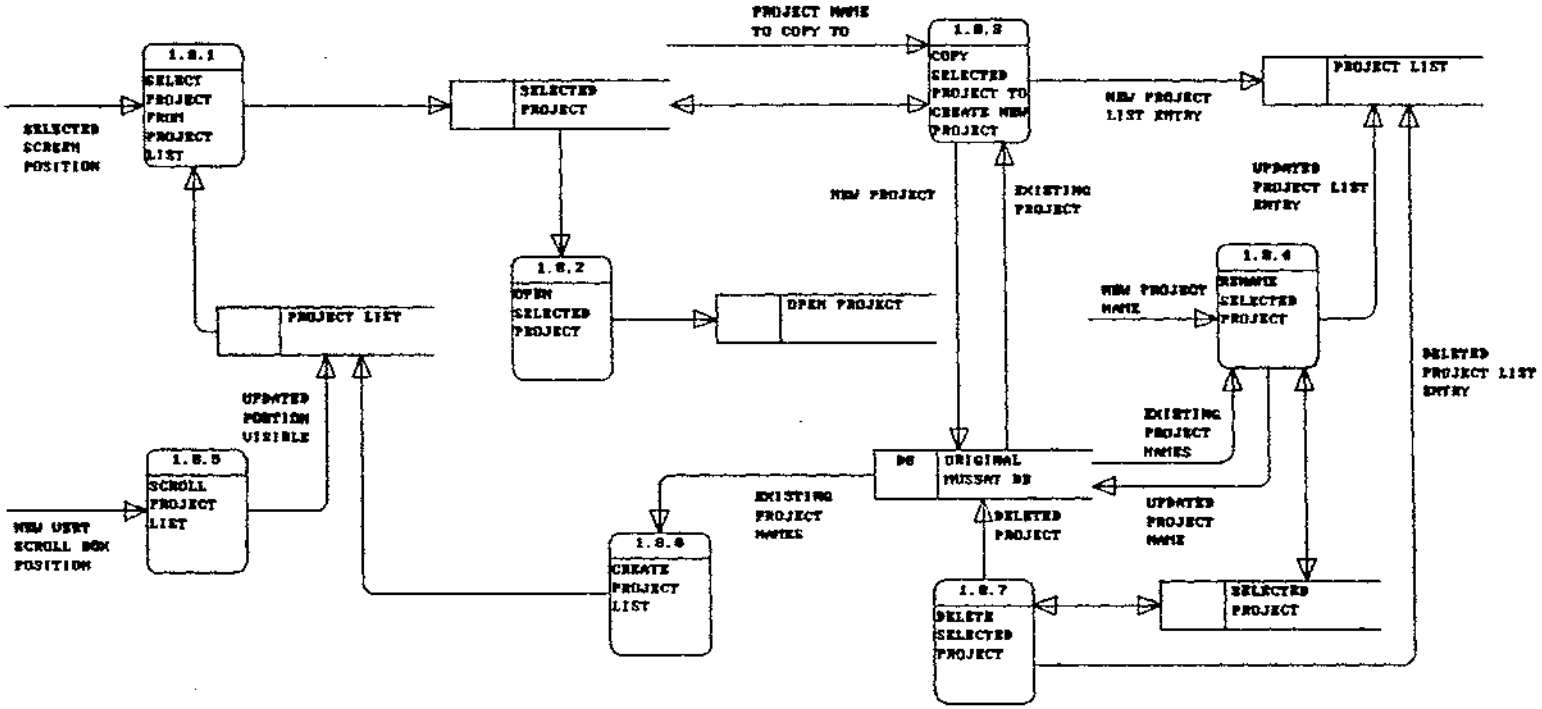
DFD 1.6.3: PERFORM PROCESS OPERATIONS



DFD 1.6.4: PERFORM DATA FLOW OPERATIONS



DFD 1.7: MAINTAIN SELECTED DB WINDOW CONTENTS



DFD 1.0: PERFORM EXISTING PROJECT OPERATIONS

3. The MUSSAT Data Dictionary

The data dictionary in Appendix V has been divided into three sub-sections:

(1) Definitions of the MUSSAT database.

(2) Definitions of data flows and all other data stores.

(3) Definitions of all processes and sub-processes.

Names of all data flows, data elements, data structures, processes, data stores and external entities are shown in upper case. Names appearing in parenthesis are the names of data flows into or out of the process being described in the DD entry. For example, if a process definition contains the statement:

get names of all matching DD ENTRY (MATCHED NAMES)

then a data flow named MATCHED NAMES will appear as a data flow into the process in the relevant DFDs and contains the data described in the preceeding statement.

3.1. Database Definitions

Name: ALIAS ENTRY

Type: DATA STRUCTURE
Alias:
Composition:

MAIN ENTRY DD ID + ALIASED FROM DD ID

Notes:

D16.

A cross reference of DD entries and where they are aliased from. The MAIN ENTRY DD ID identifies the entry that serves as the definition of the entry identified by ALIASED FROM DD ID. Hence, the DD entry identified by ALIASED FROM DD ID is an alias of MAIN ENTRY DD ID.

Name: ALIASED FROM DD ID

Type: DATA STRUCTURE
Alias:
Composition:

DD KEY

Notes:

Key of DD entry that is defined as an alias of another entry - i.e. the entry specified by ALIASED FROM DD ID does not have a DEFN field of its own, but uses the definition of another DD entry.

Name: ARG POSITION

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Description of where to draw an argument label for a particular data flow occurrence.

Name: COMPONENT DATA ELEMENT DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

DD KEY

Notes:

Key of data element DD entry that is a used in the definition field of another data flow, data store or data structure DD entry.

Name: COMPONENT DATA FLOW DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

DD KEY

Notes:

Key of data flow DD entry that is used in the definition field of another data flow DD entry.

Name: COMPONENT DATA STRUCTURE DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Key of data structure DD entry that is used in the definition field of data flow, data store or data structure entry.

Name: COMPONENT SUB PROCESS DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Key of sub process DD entry that is used in the definition of a another process or sub process entry.

Name: DATA ELEMENT DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + DATA ELEMENT NAME + DATA ELEMENT DEFN + NOTES

Notes:

D8.

Contains a unique key (DD ENTRY ID) and the information that is stored for a data element DD entry.

Name: DATA ELEMENT DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string describing contents of a data element.

Name: DATA ELEMENT NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: DATA ELEMENT USAGE

Type: DATA STORE
Alias:
Composition:

MAIN ENTRY DD ID + COMPONENT DATA ELEMENT DD ENTRY ID

Notes:

D18.

A cross reference of which DD entries use which data element definitions. The MAIN ENTRY DD ID field identifies the DD entry that uses the data element identified by COMPONENT DATA ELEMENT DD ENTRY ID.

Name: DATA FLOW DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + DATA FLOW NAME + DATA FLOW TYPE + VOLUME + DATA FLOW DEFN + ARG NAME + NOTES

Notes:

D9.

Contains a unique key (DD ENTRY ID) and all information stored for a data flow DD entry.

Name: DATA FLOW DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string describing contents of a data flow.

Name: DATA FLOW DESC

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Description of how to draw a specific data flow occurrence. Must include a description of each of the data flow segments and the position of turning points.

Name: DATA FLOW END POINTS

Type: DATA STORE
Alias:
Composition:

DATA FLOW ID + SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

D22.

Cross reference of the DFD elements that a data flow connects.

Name: DATA FLOW ID

Type: DATA STRUCTURE
Alias:
Composition:

DFD ELEMENT KEY

Notes:

Key identifying a particular occurence of a data flow.

Name: DATA FLOW NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: DATA FLOW OCCURRENCE

Type: DATA STRUCTURE
Alias:
Composition:

DATA FLOW ID + DATA FLOW DESC + NAME POSITION +
ARG POSITION + LINE WIDTH + NUM ARROWS

Notes:

All information required in order to draw a specific data flow occurrence.

Name: DATA FLOW OCCURRENCES

Type: DATA STORE
Alias:
Composition:

{DATA FLOW OCCURRENCE}

Notes:

D4.

Description of each data flow shown in a DFD.

Name: DATA FLOW TYPE

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: DATA FLOW USAGE

Type: DATA STORE
Alias:
Composition:

MAIN ENTRY DD ID + COMPONENT DATA FLOW DD ENTRY ID

Notes:

D17.

A cross reference of which data flows are components of other data flows. The MAIN ENTRY DD ID field identifies the parent data flow and the COMPONENT DATA FLOW DD ENTRY ID identifies the child data flow.

Name: DATA STORE DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + DATA STORE NAME + DATA STORE NUMBER + ORGANISATION + DATA STORE DEFN + NOTES

Notes:

D10.

A unique key (DD ENTRY ID) and the information stored for a data store DD entry.

Name: DATA STORE DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string describing the contents of a data store.

Name: DATA STORE ID

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Key identifying a particular data store occurrence.

Name: DATA STORE NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: DATA STORE NUMBER

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: DATA STORE OCCURRENCE

Type: DATA STRUCTURE
Alias:
Composition:

DATA STORE ID + POSITION + LINE WIDTH

Notes:

Description of how a data store occurrence should be drawn in a DFD.

Name: DATA STORE OCCURRENCES

Type: DATA STORE
Alias:
Composition:

{DATA STORE OCCURRENCE}

Notes:

D5.

Description of all occurrences of a particular data store.

Name: DATA STRUCTURE DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + DATA STRUCTURE NAME + VOLUME +
DATA STRUCTURE DEFN + NOTES

Notes:

D11.

Name: DATA STRUCTURE DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string describing contents of a data structure.

Name: DATA STRUCTURE NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string

Name: DATA STRUCTURE USAGE

Type: DATA STORE
Alias:
Composition:

MAIN ENTRY DD ID + COMPONENT DATA STRUCTURE DD ENTRY ID

Notes:

D19.

A cross reference of which DD entries use which data element definitions. The MAIN ENTRY DD ID field identifies the DD entry that uses the data element identified by COMPONENT DATA STRUCTURE DD ENTRY ID.

Name: DATE DFD CREATED

Type: DATA STRUCTURE
Alias:
Composition:

DATE/TIMESTAMP

Notes:

System maintained field describing when a DFD was added to a MUSSAT project.

Name: DATE DFD LAST AMMENDED

Type: DATA STRUCTURE
Alias:
Composition:

DATE/TIMESTAMP
Notes:

System maintained field describing when a particular DFD was updated as the result of the user perforing an operation that affects the physical appearance of the associated DFD. For example, naming or moving an element.

Name: DATE PROJECT CREATED

Type: DATA STRUCTURE
Alias:
Composition:

DATE/TIMESTAMP

Notes:

System maintained field indicating when a project was created by a user.

Name: DATE PROJECT LAST AMMENDED

Type: DATA STRUCTURE
Alias:
Composition:

DATE/TIMESTAMP

Notes:

System maintained field indicating when a project was last ammended by a user. Updated as for DATE DFD LAST AMMENDED and also when a DD entry is deleted or changed by editing.

Name: DATE/TIMESTAMP

Type: DATA STRUCTURE
Alias:
Composition:

DAY + MONTH + YEAR + TIME

Notes:

Name: DD

Type: DATA STORE
Alias:
Composition:

{DD ENTRY ID + DD ENTRY NAME}

Notes:

D3.

Name and keys of all entries in D8 - D14, except for processes with child DFDs. Sorted in alphabetic order on DD ENTRY NAME.

Name: DD DELETE VALUE

Type : DATA ELEMENT
Alias:
Values and Meanings:

DELETE = delete all DD entry once it becomes
         unused
SAVE   = save unused DD entries
Notes:

Name: DD DISPLAY VALUE

Type : DATA ELEMENT
Alias:
Values and Meanings:

FULL =   show all DD information, including
          details of all DFD element instances
SUMMARY = SHOW only summary information
Notes:

Name: DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

DD KEY

Notes:

Unique identifier of a DD entry.

Name: DD KEY

Type : DATA ELEMENT
Alias:
Values and Meanings:

Notes:

A key used to uniquely identify each DD entry (in data stores D8-D18).

Name: DD ENTRY NAME

Type: DATA STRUCTURE
Alias:
Composition:

[DATA ELEMENT NAME | DATA FLOW NAME | DATA STORE NAME |
DATA STRUCTURE NAME | EE NAME | PROCESS NAME |
SUB PROCESS NAME]

Notes:

Name: DESTN OBJECT ID

Type: DATA STRUCTURE
Alias:
Composition:

DFD ELEMENT KEY

Notes:

Key of DFD element occurrence that serves as the destination of a specific data flow occurrence.

Name: DFD

Type: DATA STORE
Alias:
Composition:

{DFD DESCRIPTION}

Notes:

D2.

A description of each DFD in the database.

Name: DFD BORDER VALUE

Type : DATA ELEMENT
Alias:
Values and Meanings:

BORDER = draw DFD border
NO BORDER = do not draw DFD border
Notes:

Name: DFD ELEMENT KEY

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Key used to uniquely identify each DFD element occurrence.
Key must be unique within data stores D4-D7

Name: DFD DESCRIPTION

Type: DATA STRUCTURE
Alias:
Composition:

DFD ID + DFD NAME + VERSION FIELD + DATE DFD CREATED +
DATE DFD LAST AMMENDED + DFD SIZE +
SAVED DFD WINDOW VIEW +
VIEWING SCALE

Notes:

Information that describes a DFD. Even though the DFD NAME can be derived from the parent process, it is included here since not all DFDs will have a parent process

---

Name: DFD ID

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Numeric key that uniquely identifies each DFD.

Name: DFD NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string. If the DFD is the child DFD of a process then the DFD and process names will be the same.

Name: DFD SIZE

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

A description of how many A4 pages a DFD consists of, and how these pages are arranged, e.g. three pages across and two down or four pages across.

Name: EE DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + EE NAME + NOTES

Notes:

D12.

Name: EE NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: EE OCCURRENCE

Type: DATA STRUCTURE
Alias:
Composition:

EE ID + POSITION + LINE WIDTH

Notes:

Name: EE OCCURRENCES

Type: DATA STORE
Alias:
Composition:

{EE OCCURRENCE}

Notes:

D6.

Name: LINE WIDTH

Type : DATA ELEMENT
Alias:
Values and Meanings:

BOLD or NORMAL

Notes:

Describes the width of the line that should be used to draw a specific DFD element occurrence.

Name: MAIN ENTRY DD ID

Type: DATA STRUCTURE
Alias:
Composition:

DD KEY

Notes:

Key of DD entry that is aliased from another DD entry, i.e. the DD entry identified by MAIN ENTRY DD ID serves as the definition of the aliased DD entry.

Name: NAME POSITION

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Description of where a data flow name label should be drawn for a particular data flow occurrence.

Name: NOTES

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: NUM ARROWS

Type : DATA ELEMENT
Alias:
Values and Meanings:

1 = single headed data flow
2 = double headed data flow

Notes:

Used to determine type of data flow.

Name: ORGANISATION

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: ORIGINAL MUSSAT DB

Type: DATA STORE
Alias:
Composition:

{PROJECT DATABASE}

Notes:

D0.

Description of all projects stored in MUSSAT.

Name: POSITION

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Description of a point at which to draw a DFD element within a DFD.

Name: PROCESS DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + PROCESS NAME + PROCESS NUMBER + COMMENT + PROCESS DEFN + NOTES

Notes:

D13.

If the process is exploded into a lower level DFD, then the PROCESS DEFN field will be empty.

Name: PROCESS DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined textual description of a process - including the names of any sub processes used.

Name: PROCESS ID

Type: DATA STRUCTURE
Alias:
Composition:

DFD ELEMENT KEY

Notes:

Name: PROCESS NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined string.

Name: PROCESS NUMBER

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

The full number of a process. Only the last digit is ever displayed in a process bubble.

Name: PROCESS OCCURRENCE

Type: DATA STRUCTURE
Alias:
Composition:

PROCESS ID + POSITION + LINE WIDTH

Notes:

Name: PROCESS OCCURRENCES

Type: DATA STORE
Alias:
Composition:

{PROCESS OCCURRENCE}

Notes:

D7.

Name: PROJECT

Type: DATA STORE
Alias:
Composition:

PROJECT NAME + DATE PROJECT CREATED +
DATE PROJECT LAST AMMENDED + DD DISPLAY VALUE +
DD DELETE VALUE + TOP DFD VALUE + DFD BORDER VALUE

Notes:

D1.

Attributes of a project, including the project parameter values.

Name: PROJECT DATABASE

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

All information contained in the MUSSAT database (data stores D1-D22) for a particular project.

Name: PROJECT NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: SAVED DFD WINDOW VIEW

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

A description of which portion of a particular DFD to display.

Name: SOURCE OBJECT ID

Type: DATA STRUCTURE
Alias:
Composition:

DFD ELEMENT KEY

Notes:

Key identifying the DFD element occurrence that serves as the source object for a particular data flow occurrence.

Name: SUB PROCESS DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + SUB PROCESS NAME + SUB PROCESS DEFN + NOTES

Notes:

D14.

Name: SUB PROCESS DEFN

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text description of a sub process, including the names of any sub processes it may use.

Name: SUB PROCESS NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: SUB PROCESS USAGE

Type: DATA STORE
Alias:
Composition:

MAIN ENTRY DD ID + COMPONENT SUB PROCESS DD ENTRY ID

Notes:

D20.

Name: TOP DFD VALUE

Type: DATA STRUCTURE
Alias:
Composition:

DFD ID

Notes:

Name: UNTYPED DD ENTRY

Type: DATA STORE
Alias:
Composition:

DD ENTRY ID + UNTYPED NAME

Notes:

D15.

Temporary DD entry for an element named in another DD entry definition and for which the user has not yet specified what type the entry should be.

Name: UNTYPED NAME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string appearing in the definition field of another DD entry.

Name: VERSION FIELD

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

Name: VIEWING SCALE

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

A description of the zoom level at which to display a particular DFD.

Name: VOLUME

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

User defined text string.

3.2. Data Item and Temporary Data Store Definitions

Name: ADDITIONS TO DB CAUSED BY EDITING

Type: DATA FLOW
Alias:
Composition:

[NEW UNTYPED DD ENTRY | NEW DD ENTRY USAGE COMPONENT |
NEW ALIAS ENTRY]

Notes:

Name: APPEARS IN DFD NAME

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

Names of the DFD in which an occurrence of the specified DFD element appears.

Name: CLOSED WINDOW

Type: DATA FLOW
Alias:
Composition:

Notes:

Deleted DD or EDIT ERRORS WINDOW

Name: COMPONENT OF NAMES

Type: DATA FLOW
Alias:
Composition:

Notes:

Names of all items which use the selected component. D17-D21.

Name: CONNECTED DATA FLOW IDS

Type: DATA FLOW
Alias:
Composition:

{DATA FLOW ID}
Notes:

Data flow occurrence IDs of those data flows connected to the selected DFD, EE PROCESS or DATA STORE. D22.

Name: COPY OF SELECTED DD ENTRY

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

[DATA STRUCTURE DD ENTRY | DATA FLOW DD ENTRY |
EE DD ENTRY | DATA STORE DD ENTRY |
PROCESS DD ENTRY | UNTYPED DD ENTRY |
SUB PROCESS DD ENTRY]
Notes:

Name: CORRESPONDING WINDOW ID

Type: DATA FLOW
Alias:
Composition:

WINDOW ID

Notes:

Name: DATA FLOW OCCURRENCE DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Name of source and destination elements of data flow occurrence.

Name: DATA FLOW OCCURRENCE IDS

Type: DATA FLOW
Alias:
Composition:

{DATA FLOW ID}

Notes:

D4.

Name: DATA FLOW OCCURRENCE NAME POSITION

Type: DATA FLOW
Alias:
Composition:

NAME POSITION

Notes:

Calculated position of data flow name label.

Name: DATA FLOW DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

DATA FLOW NAME + DATA FLOW TYPE + VOLUME +
DATA FLOW DEFN + NOTES + MAIN ENTRY NAME +
ARG NAME
Notes:

Name: DATA STORE DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

DATA STORE NAME + DATA STORE NUMBER +
ORGANISATION + DATA STORE DEFN + NOTES +
MAIN ENTRY NAME
Notes:

Name: DATA STORE OCCURRENCES IN OPEN DFD

Type: DATA FLOW
Alias:
Composition:

{DATA STORE ID}
Notes:

IDs of all occurrences of a particular data store in the OPEN DFD. D5.

Name: DD ENTRY DETAILS

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW DETAILS | DATA STORE DETAILS |
EE DETAILS | PROCESS DETAILS |
DATA STRUCTURE DETAILS | UNTYPED DETAILS |
SUB PROCESS DETAILS]

Notes:

Name: DD ENTRY NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME | DATA STORE NAME | DATA STRUCTURE NAME |
DATA ELEMENT NAME | UNTYPED NAME]

Notes:

Name: DD ENTRY NAMES TO REPORT ON

Type: DATA FLOW
Alias:
Composition:

{DD ENTRY NAME}

Notes:

Name: DD REPORT

Type: DATA FLOW
Alias:
Composition:

[MAIN & SUB ENTRIES REPORT | UNUSED ENTRIES REPORT |
DFD DD ENTRIES REPORT | PRINTED DD REPORT]

Notes:

Name: DELETED ALIAS ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

A deleted D16 - ALIAS ENTRY row.

Name: DELETED DATA FLOW END POINTS

Type: DATA FLOW
Alias:
Composition:

Notes:

A deleted D22 - DATA FLOW END POINTS row.

Name: DELETED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted data structure, data flow, EE, data store, utyped or sub process DD entry. D9-D15.

Name: DD WINDOW

Type: DATA STRUCTURE
Alias:
Composition:

WINDOW ID + WINDOW POSITION + DD ENTRY ID +
FORMATTED DD ENTRY TEXT + DD WINDOW VIEW

Notes:

Name: DD WINDOW VIEW

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Describes the portion of the underlying DD ENTRY that the associated DD WINDOW displays

---

Name: DELETED DD INFORMATION

Type: DATA FLOW
Alias:
Composition:

DELETED ORDERED DD ENTRY + DELETED DD ENTRY +
DELETED USAGE OF DD ENTRY COMPONENT

Notes:

Name: DELETED DD WINDOW

Type: DATA FLOW
Alias:
Composition:

Notes:

Window deleted as a result of DD entry being deleted.

Name: DELETED DFD

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted DFD. D2.

Name: DELETED DFD ELEMENT OCCURRENCE

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted process, data flow, EE or data store occurrence. D4-D7.

Name: DELETED DFD ELEMENT END POINTS

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted DF END POINTS row. D22.

Name: DELETED DFD ELEMENT USAGE

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted usage row. D16-D21.

Name: DELETED DFD OCCURRENCE INFORMATION

Type: DATA FLOW
Alias:
Composition:

DELETED DATA FLOW END POINTS +
DELETED DFD ELEMENT OCCURRENCE

Notes:

Name: DELETED ERASED ELEMENTS NAME

Type: DATA FLOW
Alias:
Composition:

Notes:

A deleted row in ERASED ELEMENTS

Name: DELETED ORDERED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

One deleted DD row. D3.

Name: DELETED PROJECT

Type: DATA FLOW
Alias:
Composition:

Notes:

All information for one project deleted (i.e. one PROJECT DATABASE deleted). DO.

Name: DELETED PROJECT LIST ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

A row to be deleted row PROJECT LIST.

Name: DATA FLOW END POINT IDS

Type: DATA FLOW
Alias:
Composition:

SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

Name: DATA FLOW PATH DESCRIPTION

Type: DATA FLOW
Alias:
Composition:

DATA FLOW DESC

Notes:

Name: DATA FLOW OCCURRENCE ARG POSITION

Type: DATA FLOW
Alias:
Composition:

ARG POSITION
Notes:

D4.

Name: DELETED USAGE OF DD ENTRY COMPONENT

Type: DATA FLOW
Alias:
Composition:

Notes:

A usage row no longer required as a result of USED item being deleted from DD entry being edited. D16-21.

Name: DELETIONS FROM DB CAUSED BY EDITING

Type: DATA FLOW
Alias:
Composition:

[DELETED USAGE OF DD ENTRY COMPONENT |
 DELETED DD ENTRY INFORMATION |
 DELETED ALIAS ENTRY]

Notes:

Name: DFD BALANCING REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: DFD DD ENTRIES REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: DFD DD ENTRY DETAILS

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW DETAILS | DATA STORE DETAILS |
EE DETAILS | PROCESS DETAILS]

Notes:

Name: DFD DESC

Type: DATA FLOW
Alias:
Composition:

DFD ID + DFD NAME + VERSION FIELD + DFD SIZE +
DFD WINDOW VIEW + VIEWING SCALE

Notes:

Name: DFD DETAILS

Type: DATA FLOW
Alias:
Composition:

DFD ID + DFD NAME + VERSION FIELD +
DATE DFD CREATED + DATE DFD LAST AMMENDED + DFD NUMBER

Notes:

D2.

Name: DFD DETAILS + DFD SIZE

Type: DATA FLOW
Alias:
Composition:

DF DID + DFD NAME + VERSION FIELD +
DATE DFD CREATED + DATE DFD LAST AMMENDED + DFD SIZE

Notes:

Name: DFD ELEMENT DESC

Type: DATA STRUCTURE
Alias:
Composition:

[DFD NAME + DATA FLOW OCCURRENCE | DATA STORE NAME +
DATA STORE NUMBER + DATA STORE OCCURRENCE | EE NAME +
EE OCCURRENCE + PROCESS NAME + PROCESS NUMBER +
PROCESS OCCURRENCE]

Notes:

All the information necessary to draw a DFD element. D4-D7, D9, D10, D12, D13.

Name: DFD ELEMENT ID

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW ID | DATA STORE ID | EE ID |
PROCESS ID]

Notes:

Name: DFD ELEMENT NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME | PROCESS NAME | EE NAME | DATA STORE NAME]

Notes:

Name: DFD ELEMENT NAMES

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT NAME}

Notes:

Name: DFD ELEMENT POSITION

Type: DATA STRUCTURE
Alias:
Composition:

[POSITION | DATA FLOW DESC + NAME POSITION +
ARG POSITION]

Notes:

Name: DFD ELEMENT POSITIONS

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT POSITION}

Notes:

Description of the space each object withing the OPEN DFD occupies. D4-D7.

Name: DFD STATISTICS REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: DFD WINDOW VIEW

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

A description of which portion of a particular DFD to dispaly.

Name: DFDS TO REPORT ON

Type: DATA FLOW
Alias:
Composition:

{DFD ID}

Notes:

IDs of all DFDs to include in a report.

Name: DUPLICATE DATA FLOW PATH DESCRIPTION

Type: DATA FLOW
Alias:
Composition:

DATA FLOW DESC

Notes:

D4.

Name: DUPLICATE SCREEN POSITION

Type: DATA FLOW
Alias:
Composition:

Notes

Representation of a selectead point within the OPEN DFD window - the position which the user has selected as the point at which a duplicate of a DFD element should be drawn.

Name: EDIT ENTRY

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

WINDOW ID

Notes:

Name: EDIT ERRORS

Type: DATA FLOW
Alias:
Composition:

{EDIT ERROR}

Notes:

All errors found in the DD ENTRY being edited.

Name: EDIT ERRORS WINDOW

Type: DATA STRUCTURE
Alias:
Composition:

WINDOW ID + WINDOW POSITION + ERRORS LISTING =
EDIT ERRORS WINDOW VIEW

Notes:

Name: EDIT ERRORS WINDOW VIEW

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Description of the portion of the EDIT ERRORS body which is currently visible in the EDIT ERRORS WINDOW.

Name: EDITS

Type: DATA FLOW
Alias:
Composition:

Notes:

All user specified changes to DD ENTRY being edited.

Name: EE DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

EE NAME + NOTES + MAIN ENTRY NAME

Notes:

Name: ELEMENT NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME | DATA STORE NAME |
EE NAME | PROCESS NAME]

Notes:

Name: END POINT OBJECT IDS

Type: DATA FLOW
Alias:
Composition:

SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

Name: END POINTS OF DUPLICATED DATA FLOW

Type: DATA FLOW
Alias:
Composition:

DATA FLOW ID + SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

A new D22 row.

Name: END POINTS OF NEW DATA FLOW

Type: DATA FLOW
Alias:
Composition:

DATA FLOW ID + SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

A new D22 row.

Name: ERASED ELEMENTS

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

{DD ENTRY ID}
Notes:

A list of the DD IDs of all elements in a DFD removed by the ERASE command.

Name: EXISTING DATA STORE NUMBERS

Type: DATA FLOW
Alias:
Composition:

{DATA STORE NUMBER}

Notes:

D10.

Name: EXISTING DD ENTRY NAMES

Type: DATA FLOW
Alias:
Composition:

{DD ENTRY NAME}

Notes:

D3.

Name: EXISTING DD ENTRY NAMES & IDS

Type: DATA FLOW
Alias:
Composition:

{DD ENTRY ID + DD ENTRY NAME}

Notes:

D3.

Name: EXISTING DFD ID

Type: DATA FLOW
Alias:
Composition:

DFD ID

Notes:

D2.

Name: EXISTING DFD NAMES

Type: DATA FLOW
Alias:
Composition:

{DFD NAME}

Notes:

D2.

Name: EXISTING OCCURRENCES

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT ID}

Notes:

D4-D7.

Name: EXISTING PROJECT

Type: DATA FLOW
Alias:
Composition:

PROJECT DATABASE

Notes:

D0.

Name: EXISTING PROJECT NAMES

Type: DATA FLOW
Alias:
Composition:

{PROJECT NAME}
Notes:

D0.

Name: FINISH NAME

Type: DATA FLOW
Alias:
Composition:

(DD ENTRY NAME + (WILDCARD CHARACTER))

Notes:

User specified text string representing last DD entry to include in report.

Name: FIRST DFD

Type: DATA FLOW
Alias:
Composition:

DFD DESCRIPTION
Notes:

Level 0 DFD created by default when a new PROJECT is created.

Name: FULL DESC OF ELEMENTS IN DFD

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT DESCRIPTION}

Notes:

D4-D7, D9, D10, D12, D13.

Name: IDS OF ALL ELEMENT IN DFD

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT ID}

Notes:

D4-D7.

Name: KEY ONLY DESC OF ELEMENTS IN DFD

Type: DATA FLOW
Alias:
Composition:

{SIMPLE DFD ELEMENT DESC}

Notes:

Key only values of each DFD element in a specific DFD. D4-D7, D9, D10, D12, D13.

Name: MAIN & SUB ENTRIES REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: MAIN ENTRY NAME

Type: DATA FLOW
Alias:
Composition:

DD ENTRY NAME

Notes:

Name: MODIFIED ELEMENT NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME| DATA STORE NAME | EE NAME | PROCESS NAME]
Notes:

Name: MUSSAT REPORT

Type: DATA FLOW
Alias:
Composition:

[PRINTED CURRENT DFD | UNNAMED ELEMENTS REPORT |
DFD BALANCING REPORT | DD REPORT | PRINTED DFD HIERARCHY |
DFD STATISTICS REPORT]

Notes:

See report mock-ups for report contents.

Name: NAME OF EXISTING DFD

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

Name: NAME OF NEW PROJECT

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME

Notes:

User specified name of a new project.

Name: NEW ALIAS ENTRY

Type: DATA FLOW
Alias:
Composition:

MAIN ENTRY DD ID + ALIASED FROM DD ID

Notes:

Name: NEW COMMENT FIELD

Type: DATA FLOW
Alias:
Composition:

COMMENT

Notes:

User specified value for process COMMENT field.

Name: NEW DATA FLOW ARGUMENT

Type: DATA FLOW
Alias:
Composition:

ARG NAME

Notes:

User specified value for data flow ARG NAME.

Name: NEW DATA STORE DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

A D10 row for an existing named and/or numbered DATA STORE OCCURRENCE (D5).

Name: NEW DATA STORE OCCURRENCE

Type: DATA FLOW
Alias:
Composition:

DATA STORE OCCURRENCE

Notes:

A new DATA STORE OCCURRENCE. D5.

Name: NEW DD ENTRY

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID + DD ENTRY NAME

Notes:

New DD row. D3.

Name: NEW DD ENTRY USAGE COMPONENT

Type: DATA FLOW
Alias:
Composition:

Notes:

A row in one of the USAGE tables created as a result of the component name being added to the DD ENTRY being edited. D16-D21.

Name: NEW DD WINDOW

Type: DATA FLOW
Alias:
Composition:

DD WINDOW

Notes:

Name: NEW DATA FLOW OCCURRENCE

Type: DATA FLOW
Alias:
Composition:

DATA FLOW OCCURRENCE

Notes:

A new DATA FLOW OCCURRENCE. D4.

Name: NEW DFD

Type: DATA FLOW
Alias:
Composition:

DFD DESCRIPTION

Notes:

Description of a new DFD. D2.

Name: NEW DFD ELEMENT DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

A new DFD ELEMENT with default values. D9, D10, D12, D13.

Name: NEW DFD NAME

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

User specified new name for existing DFD.

Name: NEW DFD SIZE

Type: DATA FLOW
Alias:
Composition:

DFD SIZE

Notes:

User specified description of number and layout of pages in a DFD.

Name: NEW EE OCCURRENCE

Type: DATA FLOW
Alias:
Composition:

EE OCCURRENCE

Notes:

A new EE OCCURRENCE. d6.

Name: NEW HORIZ SCROLL BOX POSITION

Type: DATA FLOW
Alias:
Composition:

SCREEN POSITION

Notes:

Name: NEW OBJECT NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME | DATA STORE NAME | EE NAME | PROCESS NAME]

Notes:

D8-14, D3.

Name: NEW ORDERED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID + DD ENTRY NAME

Notes:

A new row in D3 - DD.

Name: NEW PROCESS NUMBER

Type: DATA FLOW
Alias:
Composition:

PROCESS NUMBER

Notes:

Updated PROCESS NUMBER - user specified last digit.

Name: NEW PROCESS OCCURRENCE

Type: DATA FLOW
Alias:
Composition:

PROCESS OCCURRENCE

Notes:

A new PROCESS OCCURRENCE with line width set do default.
D7.

Name: NEW PROJECT

Type: DATA FLOW
Alias:
Composition:

PROJECT DATABASE

Notes:

D0.

Name: NEW PROJECT DETAILS

Type: DATA FLOW
Alias:
Composition:

PROJECT DESCRIPTION

Notes:

D1.

Name: NEW PROJECT LIST ENTRY

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME + LIST POSITION

Notes:

A new row in PROJECT LIST.

Name: NEW PROJECT NAME

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME

Notes:

New user specified name for an existing project.

Name: NEW SCROLL BOX POSITION

Type: DATA FLOW
Alias:
Composition:

[NEW HORIZ SCROLL BOX POSITION |
 NEW VERT SCROLL BOX POSITION]

Notes:

Name: NEW TOP DFD VALUE

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

Name: NEW UNTYPED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID + UNTYPED NAME

Notes:

Name: NEW VERSION FIELD

Type: DATA FLOW
Alias:
Composition:

VERSION FIELD

Notes:

Name: NEW VERT SCROLL BOX POSITION

Type: DATA FLOW
Alias:
Composition:

SCREEN POSITION

Notes:

Name: NEXT DD ENTRY ID

Type: DATA STRUCTURE
Alias:
Composition:

DD ENTRY ID

Notes:

Name: NEXT/PREV DD ENTRY ID

Type: DATA FLOW
Alias:
Composition:

[NEXT DD ENTRY ID | PREV DD ENTRY ID]
Notes:

Name: NUMBER OF DFD LEVELS TO REPORT ON

Type: DATA FLOW
Alias:
Composition:

(POSITIVE INTEGER)
Notes:

Null value indicates that all levels are to be reported included, a value of 0 indicates that only selected DFD is to be included.

Name: OBJECT SPECIFIC DATA

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW PATH DESCRIPTION | DUPLICATE SCREEN POSITION |
DUPLICATE DATA FLOW PATH DESCRIPTION | ELEMENT NAME |
NEW DATA FLOW ARGUMENT | NEW COMMENT FIELD |
NEW PROCESS NUMBER | SPECIFIED DATA STORE NUMBER |
SPECIFIED DFD NAME | SPECIFIED DFD NUMBER | UNLINKED DFD NAME]

Notes:

Name: OCCURRENCE DETAILS

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW OCCURRENCE DETAILS | OTHER OCCURRENCE DETAILS]

Notes:

D2, D4-7, D9, D1, D12, D13, D22.

Name: OPEN DD WINDOWS

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

{DD WINDOW} + {EDIT ERRORS WINDOW}1
Notes:

Name: OPEN DFD

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

DFD ID

Notes:

Name: OPEN DFD NUMBER

Type: DATA FLOW
Alias:
Composition:

PROCESS NUMBER

Notes:

D13.

Name: OPEN DFD PROCESS NUMBERS

Type: DATA FLOW
Alias:
Composition:

{PROCESS NUMBER}

Notes:

D13.

Name: OPEN DFD SIZE

Type: DATA FLOW
Alias:
Composition:

DFD SIZE

Notes:

D2.

Name: OPEN DFD ZOOM LEVEL

Type: DATA FLOW
Alias:
Composition:

VIEWING SCALE

Notes:

D2.

Name: OPEN PROJECT

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

PROJECT NAME

Notes:

Name: OPEN PROJECT NAME

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME

Notes:

D1.

Name: OPEN WINDOW

Type: DATA FLOW
Alias:
Composition:

DD WINDOW

Notes:

Name: OPEN WINDOW CONTENTS

Type: DATA FLOW
Alias:
Composition:

[FORMATTED DD ENTRY TEXT | ERRORS LISTING]

Notes:

Name: OPEN WINDOW POSITION

Type: DATA FLOW
Alias:
Composition:

WINDOW POSITION

Notes:

Name: OTHER OCCURRENCE DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

Notes:

Names of all data flows in and out of a PROCESS, DATA STORE or EE OCCURRENCE.

Name: PARENT DFD ID

Type: DATA FLOW
Alias:
Composition:

DFD ID

Notes:

D2.

Name: PARENT DFD NUMBER

Type: DATA FLOW
Alias:
Composition:

DFD NUMBER

Notes:

D2.

Name: PARENT PROCESS ID

Type: DATA FLOW
Alias:
Composition:

PROCESS ID

Notes:

D7.

Name: PARENT PROCESS NUMBER

Type: DATA FLOW
Alias:
Composition:

PROCESS NUMBER

Notes:

Name: PREV DD ENTRY ID

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID

Notes:

Name: PRINTED CURRENT DFD

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: PRINTED DD REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: PRINTED DFD HIERARCHY

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: PROCESS DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

PROCESS NAME + PROCESS NUMBER + COMMENT + PROCESS DEFN + NOTES
Notes:

Name: PROJECT DD DELETE VALUE

Type: DATA FLOW
Alias:
Composition:

DD DELETE VALUE

Notes:

D1.

Name: PROJECT DD DISPLAY VALUE

Type: DATA FLOW
Alias:
Composition:

DD DISPLAY VALUE

Notes:

D1.

Name: PROJECT DETAILS

Type: DATA FLOW
Alias:
Composition:

{PROJECT INPUT DATUM}

Notes:

Name: PROJECT DFD BORDER VALUE

Type: DATA FLOW
Alias:
Composition:

DFD BORDER VALUE

Notes:

D1.

Name: PROJECT INPUT DATUM

Type: DATA FLOW
Alias:
Composition:

[EDITS | FINISH NAME | MAIN ENTRY NAME | MODIFIED ELEMENT NAME |
NEW DFD NAME | NEW PROJECT NAME | NEW SCROLL BOX POSITION |
NEW TOP DFD VALUE | OBJECT SPECIFIC DATA |
SELECTED SCREEN POSITION | SPECIFIED DFD NAME |
SPECIFIED DFD NUMBER | SPECIFIED ENTRY NAME | START NAME |
TOP DFD SPECIFIER]

Notes:

Name: PROJECT LIST

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

{PROJECT NAME + LIST POSITION} + PORTION VISIBLE
Notes:

Name:PROJECT NAME & DATES

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME + DATE PROJECT CREATED +
DATE PROJECT LAST AMMENDED

Notes:

D1.

Name: PROJECT NAME TO COPY TO

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME

Notes:

User specified PROJECT NAME use in COPY command.

Name: RESTORED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

Saved version of entry as at last SAVE EDITS or prior to edit session. D9-15.

Name: SELECTED DATA FLOW DESC

Type: DATA FLOW
Alias:
Composition:

DATA FLOW DESC

Notes:

D4.

Name: SELECTED DATA FLOW LINE WIDTH & NUM ARROWS

Type: DATA FLOW
Alias:
Composition:

LINE WIDTH + NUM ARROWS

Notes:

D4.

Name: SELECTED DATA FLOW NAME

Type: DATA FLOW
Alias:
Composition:

DATA FLOW NAME

Notes:

D9.

Name: SELECTED DATA STORE DD ENTRY ID

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID

Notes:

D10.

Name: SELECTED DATA STORE LINE WIDTH

Type: DATA FLOW
Alias:
Composition:

LINE WIDTH

Notes:

D5.

Name: SELECTED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

D9, D10, D12-15.

Name: SELECTED DD ENTRY ID

Type: DATA FLOW
Alias:
Composition:

Notes:

Name: SELECTED DD ENTRY NAME

Type: DATA FLOW
Alias:
Composition:

[DATA ELEMENT NAME | DATA FLOW NAME | DATA STORE NAME |
DATA STRUCTURE NAME | EE NAME | PROCESS NAME |
SUB PROCESS NAME | UNTYPED NAME]

Notes:

D8-D15.

Name: SELECTED DD WINDOW

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

WINDOW ID

Notes:

Name: SELECTED EE DD ENTRY ID

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID

Notes:

D12.

Name: SELECTED EE LINE WIDTH

Type: DATA FLOW
Alias:
Composition:

LINE WIDTH

Notes:

D6.

Name: SELECTED EE NAME

Type: DATA FLOW
Alias:
Composition:

EE NAME

Notes:

D12.

Name: SELECTED OBJECT

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

OBJECT TYPE + [DATA FLOW ID | DATA STORE ID | EE ID |
PROCESS ID]
Notes:

OBJECT TYPE will indicate whether the selected object is a: PROCESS, EE DATA FLOW, DATA FLOW NAME LABEL, DATA FLOW ARG LABEL, or DATA STORE.

Name: SELECTED OBJECT ID

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW ID | DATA STORE ID | EE ID | PROCESS ID]

Notes:

Name: SELECTED PROCESS ID

Type: DATA FLOW
Alias:
Composition:

PROCESS ID

Notes:

Name: SELECTED PROCESS NAME

Type: DATA FLOW
Alias:
Composition:

PROCESS NAME

Notes:

D13.

Name: SELECTED PROCESS NUMBER

Type: DATA FLOW
Alias:
Composition:

PROCESS NUMBER

Notes:

D13.

Name: SELECTED PROJECT

Type: DATA STORE
Number:
Alias:
Organisation:
Composition:

PROJECT NAME

Notes:

A PROJECT NAME selected from the lilst of existing projects.

Name: SELECTED PROJECT DATABASE

Type: DATA FLOW
Alias:
Composition:

PROJECT DATABASE

Notes:

Name: SELECTED SCREEN POSITION

Type: DATA FLOW
Alias:
Composition:

Notes:

Description of the selected point within the OPEN DFD WINDOW. Used to determine which DFD element was selected, or to specify the position that a new DFD ELEMENT or label will occupy.

Name: SELECTED WINDOW POSITION

Type: DATA FLOW
Alias:
Composition:

WINDOW POSITION

Notes:

Used to determine where to show 'E' marker for edit entry.

Name: SIMPLE DFD ELEMENT DESC

Type: DATA STRUCTURE
Alias:
Composition:

[DATA FLOW ID + DATA FLOW NAME |
DATA STORE ID + DATA STORE NAME |
EE ID + EE NAME | PROCESS ID + PROCESS NAME]

Notes:

Name: SPECIFIED DATA STORE NUMBER

Type: DATA FLOW
Alias:
Composition:

DATA STORE NUMBER

Notes:

User specified number for a data store - need not be unique.

Name: SPECIFIED DFD NAME

Type: DATA FLOW
Alias:
Composition:

(DFD NAME)

Notes:

User specified.

Name: SPECIFIED DFD NUMBER

Type: DATA FLOW
Alias:
Composition:

(DFD NUMBER)
Notes:

User specified.

Name: SPECIFIED ENTRY NAME

Type: DATA FLOW
Alias:
Composition:

DD ENTRY NAME

Notes:

User specified.

Name: START NAME

Type: DATA FLOW
Alias:
Composition:

(DD ENTRY NAME + (WILDCARD CHARACTER))

Notes:

Name: SUB PROCESS DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

SUB PROCESS NAME + SUB PROCESS DEFN + NOTES

Notes:

Name: TOP DFD DETAILS

Type: DATA FLOW
Alias: DFD DETAILS
Composition:

Notes:

Name: TOP DFD SPECIFIER

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

Name: UNDEFINED DFD ELEMENT NAMES

Type: DATA FLOW
Alias:
Composition:

{DFD ELEMENT NAMES}

Notes:

Name: UNLINKED DFD NAME

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

Name: UNNAMED ELEMENTS REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: UNNUMBERED DATA STORES

Type: DATA FLOW
Alias:
Composition:

DD ENTRY ID

Notes:

D10.

Name: UNNUSED ENTRIES REPORT

Type: DATA FLOW
Alias:
Composition:

Notes:

As per report mock-up.

Name: UNTYPED DETAILS

Type: DATA STRUCTURE
Alias:
Composition:

UNTYPED NAME

Notes:

Name: UPDATED ALIAS ENTRY

Type: DATA FLOW
Alias:
Composition:

MAIN ENTRY DD ID + ALIASED FROM DD ID

Notes:

Name: UPDATED ARG POSITION

Type: DATA FLOW
Alias:
Composition:

ARG POSITION

Notes:

D9.

Name: UPDATED DATA FLOW ARG

Type: DATA FLOW
Alias:
Composition:

ARG NAME

Notes:

D9.

Name: UPDATED DATA FLOW END POINTS

Type: DATA FLOW
Alias:
Composition:

DATA FLOW ID + SOURCE OBJECT ID + DESTN OBJECT ID

Notes:

Updated D22 row for redrawn DATA FLOW OCCURRENCE.

Name: UPDATED DD WINDOW VIEW

Type: DATA FLOW
Alias:
Composition:

[DD WINDOW VIEW | EDIT ERRORS WINDOW VIEW]
Notes:

Name: UPDATED CHILD DFD NAME

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

D2.

Name: UPDATED DATA STORE NUMBER

Type: DATA FLOW
Alias:
Composition:

Notes:

D10.

Name: UPDATED DATA STORE POSITION

Type: DATA FLOW
Alias:
Composition:

POSITION

Notes:

D5.

Name: UPDATED DATA DFD LAST AMMENDED

Type: DATA FLOW
Alias:
Composition:

DATE DFD LAST AMMENDED

Notes:

D2.

Name: UPDATED DATA FLOW DESC

Type: DATA FLOW
Alias:
Composition:

DATA FLOW DESC + NAME POSITION + ARG POSITION

Notes:

D4.

Name: UPDATED DATA FLOW LABEL POSITIONS

Type: DATA FLOW
Alias:
Composition:

[NAME POSITION | ARG POSITION |
 NAME POSITION + ARG POSITION]

Notes:

D4.

Name: UPDATED DD DELETE VALUE

Type: DATA FLOW
Alias:
Composition:

DD DELETE VALUE

Notes:

D1.

Name: UPDATED DD DISPLAY VALUE

Type: DATA FLOW
Alias:
Composition:

DD DISPLAY VALUE

Notes:

D1.

Name: UPDATED DD WINDOW CONTENTS

Type: DATA FLOW
Alias:
Composition:

FORMATTED DD ENTRY TEXT

Notes:

Name: UPDATED DFD BORDER VALUE

Type: DATA FLOW
Alias:
Composition:

DFD BORDER VALUE

Notes:

D1.

Name: UPDATED DFD NAME

Type: DATA FLOW
Alias:
Composition:

DFD NAME

Notes:

D2.

Name: UPDATED DFD WINDOW VIEW

Type: DATA FLOW
Alias:
Composition:

DFD WINDOW VIEW

Notes:

D2.

Name: UPDATED DFD SIZE

Type: DATA FLOW
Alias:
Composition:

DFD SIZE

Notes:

D2.

Name: UPDATED EE POSITION

Type: DATA FLOW
Alias:
Composition:

POSITION

Notes:

D6.

Name: UPDATED LAST AMMENDED DATES

Type: DATA FLOW
Alias:
Composition:

DATA DFD LAST AMMENDED + (DATA PROJECT LAST AMMENDED)

Notes:

Name: UPDATED NAME POSITION

Type: DATA FLOW
Alias:
Composition:

NAME POSITION

Notes:

D4.

Name: UPDATED OBJECT LINE WIDTH

Type: DATA FLOW
Alias:
Composition:

LINE WIDTH

Notes:

D4-D7.

Name: UPDATED OBJECT NAME

Type: DATA FLOW
Alias:
Composition:

[DATA FLOW NAME | DATA STORE NAME | EE NAME | PROCESS NAME]
Notes:

D3, D9, D10, D12, D13.

Name: UPDATED OPEN WINDOW

Type: DATA FLOW
Alias:
Composition:

WINDOW POSITION

Notes:

Name: UPDATED ORDERED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

DD ENTRY NAME

Notes:

D3.

Name: UPDATED PARENT PROCESS NAME

Type: DATA FLOW
Alias:
Composition:

PROCESS NAME

Notes:

D13.

Name: UPDATED PORTION VISIBLE

Type: DATA FLOW
Alias:
Composition:

PORTION VISIBLE

Notes:

Name: UPDATED PROCESS COMMENT

Type: DATA FLOW
Alias:
Composition:

COMMENT

Notes:

D13.

Name: UPDATED PROCESS NUMBER

Type: DATA FLOW
Alias:
Composition:

PROCESS NUMBER

Notes:

D13.

Name: UPDATED PROCESS POSITION

Type: DATA FLOW
Alias:
Composition:

POSITION

Notes:

D7.

Name: UPDATED PROJECT

Type: DATA FLOW
Alias:
Composition:

PROJECT DATABASE

Notes:

D0.

Name: UPDATED PROJECT LIST ENTRY

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME + LIST POSITION

Notes:

Name: UPDATED PROJECT NAME

Type: DATA FLOW
Alias:
Composition:

PROJECT NAME

Notes:

D1.

Name: UPDATED SELECTED DD ENTRY

Type: DATA FLOW
Alias:
Composition:

Notes:

Updated version fo DD entryas a result of edits - updated version is error-free. D9-D15.

Name: UPDATED TOP DFD VALUE

Type: DATA FLOW
Alias:
Composition:

TOP DFD VALUE

Notes:

D1.

Name: UPDATED VERSIION FIELD

Type: DATA FLOW
Alias:
Composition:

VERSION FIELD

Notes:

D2.

Name: UPDATED WINDOW VIEW

Type: DATA FLOW
Alias:
Composition:

[EDIT ERRORS WINDOW VIEW | DD WINDOW VIEW]
Notes:

Name: UPDATED ZOOM LEVEL

Type: DATA FLOW
Alias:
Composition:

VIEWING SCALE

Notes:

D2.

Name: UPDATES TO DB CAUSED BY EDITING

Type: DATA FLOW
Alias:
Composition:

[UPDATED SELECTED DD ENTRY | UPDATED ALIAS ENTRY]

Notes:

Name: USER

Type: EXTERNAL ENTITY
Alias:

Notes:

Name: VISIBLE ENTRIES

Type: DATA FLOW
Alias:
Composition:

{DD ENTRY ID}

Notes:

DD ENTRY IDs of all DD entries currently visible.

3.3. Process and Sub Process Definitions

```
Name: ACTION EDITED ALIAS
Type: SUB PROCESS
Description:

if new alias field is empty
  then
    delete existing alias entry (DELETED ALIAS ENTRY)
    if DD ENTRY associated with alias entry is not
        used by any DFD ELEMENT, or any other DD ENTRY,
        or as a MAIN ENTRY for any DD ENTRY
      then
        if PROJECT DD DELETE VALUE = DELETE
          then
            *DELETE DD ENTRY* for alias entry
        end if
    end if
  else
    if there is not an existing alias entry for
         DD ENTRY
       then
         get DD ENTRY ID for DD ENTRY named in alias
             field
         create an alias entry  (NEW ALIAS ENTRY)
       else
         get DD ENTRY ID for DD ENTRY named in alias
             field
         update existing alias entry (UPDATED ALIAS
             ENTRY)
         if DD ENTRY associated with original alias
             is not used by any DFD ELEMENT, DD ENTRY
             or as the MAIN ENTRY for a DD ENTRY
           then
             if PROJECT DD DELETE VALUE = DELETE
               then
                 *DELETE DD ENTRY* for old alias
             end if
         end if
    end if
end if

Notes:
```

```
Name: ACTION EDITED DEFN FIELD
Type: SUB PROCESS
Description:

for each DD ENTRY name that appears in new version and
not in the old version of the DD ENTRY, do:
  if the named entry does not exists
    then
      create a NEW UNTYPED DD ENTRY for component name
  end if
  create a NEW DD ENTRY USAGE COMPONENT linking
    edited DD ENTRY and component

for each DD ENTRY name that appeared in the old version
and not in the new version of the DD ENTRY, do:
  delete usage of component (DELETED USAGE OF DD ENTRY COMPONENT)
  if DD ENTRY associated with component is not used by any
      DFD ELEMENT, or DD ENTRY, or as the MAIN ENTRY of another
      DD ENTRY
    then
      if PROJECT DD DELETE VALUE = DELETE
        then
          *DELETE DD ENTRY* of component
      end if
  end if
Notes:
```

```
Name: ACTION EDITED SELECTED DD ENTRY

Type: PROCESS
Number: 1.7.4
Description:

check edited version of DD ENTRY for errors
if any errors are found
  then
    display error dialogue box
  else
  *UPDATE DB TO REFLECT EDITS*
  update DATE PROJECT LAST AMMENDED
end if

Notes:
```

Name: AUTONUMBER DATA STORES

Type: SUB-PROCESS
Description:

```
find DD ENTRY IDs of all unnumbered data stores in
    OPEN DFD (UNNUMBERED DATA STORES)
for each of these DD ENTRY IDs do:
  calculate next data store number to allocate
     (using EXISTING DATA STORE NUMBERS)
  set UPDATED DATA STORE NUMBER to calculated
      number
```

Notes:

I.E. will only autonumber named datastores.

Name: AUTONUMBER DFD ELEMENTS

Type: PROCESS
Number: 1.6.1.4
Description:

```
*AUTONUMBER DATA STORES*
*AUTONUMBER PROCESSES*
if one or more DFD elements were numbered
  then
    update DATA PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
end if
```

Notes:

Name: AUTONUMBER PROCESSES

Type: SUB-PROCESS
Description:

working top to bottom, and left to right across DFD,
(re)number the last digit of each PROCESS NUMBER
(UPDATED PROCESS NUMBER) starting from '1'.

Notes:

Name: BALANCE FLOWS INTO PARENT PROCESS WITH NET FLOWS
INTO DFD

Type: SUB-PROCESS
Description:

get names of all data flows flowing into parent process
get name of all net flows into DFD
new flows into a DFD are found by:
  retrieving names of all data flows in DFD with no
      source object
  retrieving names of all data flows in DFD flowing from
      a data store that serves as a source oject for a data flow
      into the parent process where the data store is named
      if such a data flow is unnamed then use the name of the
        connected data store.
for each data flow into the parent process do:
  search for a matching data flow or data flow component in
      net flows into child DFD
  a match exists if:
    a data flow with the same name exists in net flows into
        the DFD, or
    one or more net flows into the DFD are sub-data flows, or
        sub-data flows of a sub-data flow of a data flow into the
        parent process.

Notes:

Name: BALANCE FLOWS OUT OF PARENT PROCESS WITH NET FLOWS
OUT OF DFD

Type: SUB-PROCESS
Description:

get names of all data flows flowing out of parent process
get name of all net flows out of DFD
new flows out of a DFD are found by:
  retrieving names of all data flows in DFD with no
    destination object
  retrieving names of all data flows in DFD flowing to
    a data store that serves as a destination oject for a data fl
      out of the parent process where the data store is named
    if such a data flow is unnamed then use the name of the
      connected data store.
for each data flow out of the parent process do:
  search for a matching data flow or data flow component in
    net flows out of child DFD
  a match exists if:
    a data flow with the same name exists in net flows out of
      the DFD, or
    one or more net flows into the DFD are sub-data flows, or
      sub-data flows of a sub-data flow of a data flow out of the
      parent process.

Notes:

Name: CHANGE CONTENTS OF SELECTED WINDOW TO 'NEXT' ENTRY

Type: PROCESS
Number: 1.5.1.2
Description:

```
get SELECTED DD ENTRY NAME for DD ENTRY ID in
    DD WINDOW indicated by SELECTED DD WINDOW
get NEXT DD ENTRY ID for DD ENTRY alphabetically
    next after SELECTED DD ENTRY NAME
if PROJECT DD DISPLAY VALUE = FULL
  then
    *RETRIEVE FULL DD ENTRY DETAILS*
  else
    *RETRIEVE SUMMARY DD ENTRY DETAILS*
end if
replace existing FORMATTED DD ENTRY TEXT with
   retrieved UPDATED DD WINDOW CONTENTS
redisplay SELECTED DD WINDOW
```

Notes:

Name: CHANGE CONTENTS OF SELECTED WINDOW TO 'PREV' ENTRY

Type: PROCESS
Number: 1.5.1.3
Description:

```
get SELECTED DD ENTRY NAME for DD ENTRY ID in
    DD WINDOW indicated by SELECTED DD WINDOW
get NEXT DD ENTRY ID for DD ENTRY alphabetically
    preceeding SELECTED DD ENTRY NAME
if PROJECT DD DISPLAY VALUE = FULL
  then
    *RETRIEVE FULL DD ENTRY DETAILS*
  else
    *RETRIEVE SUMMARY DD ENTRY DETAILS*
end if
replace existing FORMATTED DD ENTRY TEXT with
   retrieved UPDATED DD WINDOW CONTENTS
redisplay SELECTED DD WINDOW
```

Notes:

Name: CHANGE DD DELETE VALUE

Type: PROCESS
Number: 1.1.7
Description:

update PROJECT DD DELETE VALUE by setting the value of
    UPDATED DD DELETE VALUE to the opposite value
    contained in PROJECT DD DELETE VALUE

Notes:

Name: CHANGE DD DISPLAY VALUE

Type: PROCESS
Number: 1.1.3
Description:

update PROJECT DD DISPLAY VALUE by setting
    the value of UPDATED DD DISPLAY VALUE to
    the opposite value contained in PROJECT
    DD DISPLAY VALUE

Notes:

Name: CHANGE DFD BORDER VALUE

Type: PROCESS
Number: 1.1.8
Description:

update DFD BORDER VALUE by setting
    the value of UPDATED DFD BORDER VALUE to
    the opposite value contained in PROJECT
    DFD BORDER VALUE

Notes:

Name: CHANGE DFD SIZE

Type: PROCESS
Number: 1.3.9
Description:

update DFD SIZE of DFD INDICATED by OPEN DFD to
  NEW DFD SIZE (UPDATED DFD SIZE)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Assumes that the user is unable to specify an invalid DFD size.

Name: CHANGE LINE WIDTH OF ALL OCCURRENCES

Type: PROCESS
Number: 1.6.1.1.4
Description:

get DFD ELEMENT IDs of all occurrences of SELECTED OBJECT
    (EXISTING OCCURRENCES)
for each occurrence do:
  set LINE WIDTH to opposite value (UPDATED OBJECT LINE WIDTH)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: CHANGE LINE WIDTH OF ALL OCCURRENCES IN DFD HIERARCHY

Type: PROCESS
Number: 1.6.1.1.2
Description:

get DFD ELEMENT IDs of all occurrences of SELECTED OBJECT in
    DFD hierarchy starting at OPEN DFD (EXISTING OCCURRENCES)
for each of the occurrences do:
  set LINE WIDTH to opposite value (UPDATED OBJECT LINE WIDTH)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: CHANGE LINE WIDTH OF ALL OCCURRENCES IN OPEN DFD

Type: PROCESS
Number: 1.6.1.1.3
Description:

get DFD ELEMENT IDs of all occurrences of SELECTED OBJECT
    in OPEN DFD (EXISTING OCCURRENCES)
for each occurrence do:
  set LINE WIDTH to opposite value (UPDATED OBJECT LINE WIDTH)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: CHANGE LINE WIDTH OF SELECTED OCCURENCE

Type: PROCESS
Number: 1.6.1.1.1
Description:

set LINE WIDTH of SELECTED OBJECT to opposite value
    (UPDATED OBJECT LINE WIDTH)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

That is, toggle value from NORMAL to BOLD or vice versa.

Name: CHANGE SELECTED PROCESS COMMENT FIELD

Type: PROCESS
Number: 1.6.3.1
Description:

update COMMENT of SELECTED OBJECT by setting UPDATED
    PROCESS COMMENT to NEW COMMENT FIELD value
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: CHANGE TO ZOOM LEVEL 1

Type: PROCESS
Number: 1.3.5
Description:

update VIEWING SCALE by setting UPDATED ZOOM LEVEL TO
    level 1 value
update DFD WINDOW VIEW by recalculating portion of DFD
    visible with new VIEWING SCALE

Notes:

Name: CHANGE TO ZOOM LEVEL 2

Type: PROCESS
Number: 1.3.6
Description:

update VIEWING SCALE by setting UPDATED ZOOM LEVEL TO
    level 2 value
update DFD WINDOW VIEW by recalculating portion of DFD
    visible with new VIEWING SCALE

Notes:

Name: CHANGE TO ZOOM LEVEL 3

Type: PROCESS
Number: 1.3.7
Description:

update VIEWING SCALE by setting UPDATED ZOOM LEVEL TO
    level 3 value
update DFD WINDOW VIEW by recalculating portion of DFD
    visible with new VIEWING SCALE

Notes:

Name: CHANGE TOP DFD VALUE

Type: PROCESS
Number: 1.1.2
Description:

```
if NEW TOP DFD VALUE does not exist in EXISTING DFD NAMES
  then
    display error dialogue box
  else
    set PROJECT TOP DFD VALUE to NEW TOP DFD VALUE
        (UPDATED TOP DFD VALUE)
end if
```

Notes:

Name: CHANGE VERSION FIELD

Type: PROCESS
Number: 1.3.1
Description:

```
update VERSION FIELD of OPEN DFD by setting value of
    UPDATED VERSION FIELD to NEW VERSION FIELD
update DATA PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: CLOSE ALL WINDOWS

Type: PROCESS
Number: 1.5.3
Description:

```
for all OPEN DD WINDOWS except the window indicated by
    EDIT ENTRY do:
  erase window
  delete window from OPEN DD WINDOWS (CLOSED WINDOW)
  if window indicated by SELECTED DD WINDOW is not
      window indicated by EDIT ENTRY
    then
      reset SELECTED DD WINDOW
  end if
```

Notes:

Name: CLOSE OPEN DFD

Type: PROCESS
Number: 1.4.2
Description:

```
erase DFD from viewing area
if PROJECT DD DELETE VALUE = DELETE
  then
    for each DD ENTRY ID in ERASED ELEMENTS do:
      if DD ENTRY indicated by DD ENTRY ID is not
          used by any DFD ELEMENT, DD ENTRY or as
          a MAIN ENTRY for another DD ENTRY
        then
          *DELETE DD ENTRY*
      end if
end if
reset ERASED ELEMENTS
reset OPEN DFD
```

Notes:

Name: COPY SELECTED PROJECT TO CREATE NEW PROJECT

Type: PROCESS
Number: 1.8.3
Description:

copy project indicated by SELECTED PROJECT to create a
    new project with PROJECT NAME = PROJECT NAME TO COPY TO
insert PROJECT NAME TO COPY TO into PROJECT LIST in
    alphabetic order
set SELECTED PROJECT to new PROJECT NAME TO COPY TO
highlight new PROJECT NAME in PROJECT LIST
Notes:

Name: CREATE CHILD DFD

Type: PROCESS
Number: 1.4.6
Description:

erase currently visible DFD indicated by OPEN DFD
create NEW DFD with DFD NAME = SELECTED PROCESS NAME
    indicated by SELECTED PROCESS ID
link NEW DFD to parent process
delete DD row for selected process (DELETED ORDERED ENTRY)
set OPEN DFD to DFD ID of NEW DFD
reset SELECTED OBJECT
DISPLAY NEW DFD indicated by OPEN DFD

Notes:

Name: CLOSE PROJECT - NO SAVE

Type: PROCESS
Number: 1.1.5
Description:

reset OPEN PROJECT DB
reset OPEN PROJECT
display PROJECT LIST

Notes:

Name: CLOSE PROJECT - WITH SAVE

Type: PROCESS
Number: 1.1.4
Description:

set DATE PROJECT LAST AMMENDED for OPEN PROJECT to current DATE/TI
copy OPEN PROJECT DB to ORIGINAL MUSSAT DB replacing original
    project details of OPEN PROJECT (UPDATED PROJECT)
reset OPEN PROJECT DB
reset OPEN PROJECT
display PROJECT LIST

Notes:

Name: CLOSE SELECTED DD WINDOW

Type: PROCESS
Number: 1.5.2
Description:

erase SELECTED DD WINDOW
delete SELECTED DD WINDOW from OPEN DD WINDOWS
    (CLOSED WINDOW)
reset SELECTED DD WINDOW

Notes:

Name: CREATE EDIT ERRORS WINDOW

Type: PROCESS
Number: 1.7.5
Description:

format EDIT ERRORS for display
create EDIT ERRORS WINDOW
set SELECTED DD WINDOW to EDIT ERRORS WINDOW ID
display SELECTED DD WINDOW at front of all windows
    with highlighted window border

Notes:

Name: CREATE NEW DATA FLOW

Type: PROCESS
Number: 1.6.4.1
Description:

determine objects at end points of DF PATH DESCRIPTION
    (END POINT OBJECT IDS) using DFD ELEMENT POSITIONS
create NEW DATA FLOW OCCURRENCE setting DATA FLOW PATH
    DESC to DATA FLOW PATH DESCRIPTION, LINE WIDTH to
    default, NUM ARROWS to 1 or 2 depending on icon used
line NEW DATA FLOW OCCURRENCT to OPEN DFD
create END POINTS OF NEW DATA FLOW using END POINT OBJECT
    IDs
set SELECTED OBJECT to DFD ELEMENT ID of NEW DATA FLOW
    OCCURRENCE

Notes:

Name: CREATE NEW DATA STORE

Type: PROCESS
Number: 1.6.2.4
Description:

```
if there is space for a data store at SELECTED POSITION
  then
    create NEW DATA STORE OCCURRENCE with POSITION set to
        SELECTED POSITION and LINE WIDTH sete to default
  else
    sound audible alarm
end if
```

Notes:

Name: CREATE NEW DD WINDOW

Type: SUB PROCESS
Description:

```
if PROJECT DD DISPLAY VALUE = FULL
  then
    *RETRIEVE FULL DD ENTRY DETAILS FOR DISPLAY* for DD ENTRY
  else
    *RETRIEVE SUMMARY DD ENTRY DETAILS FOR DISPLAY* for DD ENTRY
end if
select a position at which to dislay a new DD WINDOW
create NEW DD WINDOW with WINDOW POSITION set to selected
    position, FORMATTED DD ENTRY TEXT set to retrieved details,
    DD WINDOW VIEW set to top of DD ENTRY
set SELECTED DD WINDOW to WINDOW ID of NEW DD WINDOW
display SELECTED DD WINDOW at fron with highlighted window frame
```

Notes:

```
Name: CREATE NEW EE

Type: PROCESS
Number: 1.6.2.1
Description:

if there is room for an EE at SELECTED POSITION
  then
    create NEW EE OCCURRENCE with position set to
        SELECTED POSITION and LINE WIDTH set to
        default
  else
    sound audible alarm
end if

Notes:
```

```
Name: CREATE NEW PROJECT

Type: PROCESS
Number: 1.9
Description:

if a project already exist with PROJECT NAME = NEW
   PROJECT NAME
  then
    display error dialogue box
  else
    create NEW PROJECT DETAILS, setting DD DISPLAY VALUE,
        DD DELETE VALUE and DFD BORDER VALUE to default values,
        and setting DATE PROJECT CREATED TO current
        DATE/TIMESTAMP
    set OPEN PROJECT to NEW PROJECT NAME
    create FIRST DFD for project setting DFD NAME to 'DFD 0',
        DATE DFD CREATED to current DATE/TIMESTAMP, DFD SIZE,
        DFD WINDOW VIEW and VIEWING SCALE to default values
    set  OPEN DFD, and TOP DFD VALUE to DFD ID of FIRST DFD
    *DISPLAY NEW DFD* indicated by OPEN DFD
end if
Notes:
```

Name: CREATE NEW UNLINKED DFD

Type: PROCESS
Number: 1.4.5
Description:

```
if NAME OF NEW DFD exists in EXISTING DFD NAMES
  then
    display error dialogue box
  else
    create NEW DFD, setting DFD NAME to NAME OF NEW DFD
        and all other fields to default values
    set OPEN DFD to DFD ID of NEW DFD
*DISPLAY NEW DFD* indicated by OPEN DFD
end if
```

Notes:

Name: CREATE PROJECT LIST

Type: PROCESS
Number: 1.8.6
Description:

```
get all EXISTING PROJECT NAMES
sort into alphabetic list
create a PROJECT LIST entry for each PROJECT NAME,
    in alphabetic order
set PORTION VISIBLE to top of list
display PROJECT LIST
initialise SELECTED PROJECT to none selected
```

Notes:

Name: DELETE ALL ASSOCIATED DATA FLOW END POINTS

Type: SUB-PROCESS
Description:

```
if the DFD ELEMENT OCCURRENCE is a data flow
  then
    delete all DATA FLOW END POINT rows where
        DATA FLOW ID = DFD ELEMENT ID
  else
    for each row in DATA FLOW END POINTS where
        either SOURCE OBJECT ID or DESTN OBJECT ID =
        DFD ELEMENT ID do:
      delete DFD ELEMENT OCCURRENCE indicated by DATA FLOW ID
      if DD ENTRY indicated by DATA FLOW ID is not
          used by any dfd element, DD ENTRY, or as the
          MAIN ENTRY for a DD ENTRY
        then
          *DELETE DD ENTRY* for DD ENTRY indicated by
              DATA FLOW ID
      end if
       delete DATA FLOW END POINTS row
    end for
end if
```

Notes:

Deletes unused DD entries of source and object DFD elements.

Name: DELETE ALL ASSOCIATED DATA FLOW END POINTS ONLY

Type: SUB-PROCESS
Description:

```
if the DFD ELEMENT OCCURRENCE is a data flow
  then  (* delete connections to source and destn objects *)
    for each  DATA FLOW END POINT row where
        DATA FLOW ID = DFD ELEMENT ID do:
      delete DATA FLOW END POINT row (DELETED DFD ELEMENT
        END POINTS)
    end for
  else (* delete all connections and connected data flow
          occurrences *)
    for each DATA FLOW END POINT row where
        either SOURCE OBJECT ID or DESTN OBJECT ID =
        DFD ELEMENT ID do:  (* for all connected data flows *)
      delete DFD ELEMENT OCCURRENCE  indicated by DATA FLOW ID
          (DELETED DFD ELEMENT OCCURRENCE)
      delete DATA FLOW END POINT row (DELETED DFD ELEMENT
        END POINTS)
```

Notes:

Doesn't delete unused entries of source or destination objects.

Name: DELETE ALL OCCURRENCES

Type: PROCESS
Number: 1.6.1.6.4
Description:

```
find DFD ELEMENT IDs of all occurrences of SELECTED OBJECT
    (EXISTING OCCURRENCES)
*DELETE EXISTING OCCURRENCES*
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: DELETE ALL OCCURRENCES IN DFD HIERARCHY

Type: PROCESS
Number: 1.6.1.6.2
Description:

find DFD ELEMENT IDs of all occurrences of SELECTED OBJECT
  in DFD hierarchy starting at OPEN DFD (EXISTING OCCURRENCES)
DELETE EXISTING OCCURRENCES
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: DELETE ALL OCCURRENCES IN OPEN DFD

Type: PROCESS
Number: 1.6.1.6.3
Description:

find DFD ELEMENT IDs of all occurrences of SELECTED OBJECT
    in OPEN DFD (EXISTING OCCURRENCES)
DELETE EXISTING OCCURRENCES
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: DELETE ALL UNUSED DD ENTRIES

Type: PROCESS
Number: 1.1.9
Description:

```
for each DD ENTRY (D8-D14) do:
  if DD ENTRY is not used by any DFD ELEMENT occurrence,
      DD ENTRY, or as a MAIN ENTRY for a DD ENTRY
    then
      *DELETE DD ENTRY*
  end if
  if one or more entries are deleted
    then
      update DATE PROJECT LAST AMMENDED
  end if
end for
```

Notes:

Name: DELETE CHILD DFD HIERARCHY

Type: SUB-PROCESS
Description:

```
for each DFD that is part of the DFD hierarchy
    with the root being the DFD that is the
    child DFD of the specified process do:
  *DELETE DFD ELEMENTS*
  if DFD ID = TOP DFD VALUE
    then
      reset TOP DFD VALUE by setting UPDATED TOP DFD VALUE
          to blank
  end if
  delete DFD DESCRIPTION for DFD (DELETED DFD)
```

Notes:

Name: DELETE DD ENTRY

Type: SUB-PROCESS
Description:

```
if DD ENTRY ID appears as an ALIASED FROM DD ID
    in ALIAS ENTRY
  then  (* uses another entry for defn field *)
    delete ALIAS ENTRY row
  else
    *DELETE DD ENTRY HIERARCHY*
```

Notes:

Even if unused, the MAIN ENTRY for an aliased entry is not deleted.

Name: DELETE DD ENTRY HIERARCHY

Type: SUB-PROCESS
Description:

```
for each DD ENTRY associated with this DD ENTRY do:
    (* i.e. each DD ENTRY ID appearing as MAIN ENTRY DD ID in
       D17-D21 where component DD ID = current DD ENTRY ID *)
  delete usage row
  if DD ENTRY indicated by MAIN ENTRY DD ID is not used by any
      DFD ELEMENT, DD ENTRY or as MAIN ENTRY for another DD ENTRY
    then
      *DELETE DD ENTRY* indicated by MAIN ENTRY DD ID
  end if
end for
delete DD row for DD ENTRY (DELETED ORDERED DD ENTRY)
delete DD entry (DELETED DD ENTRY)
```

Notes:

Name: DELETE DFD ELEMENTS

Type: SUB-PROCESS
Description:

```
for each DFD element in the DFD do:
    *DELETE CONNECTED DATA FLOW OCCURENCES*
    delete the DFD ELEMENT occurrence
        (DELETED DFD ELEMENT OCCURRENCE)
  if PROJECT DD DELETE VALUE = DELETE
    then
      if the DD ENTRY indicated by the DFD ELEMENT ID is not
           used by any other DFD ELEMENT, DD ENTRY or as a
           MAIN ENTRY for another DD ENTRY
         then
           *DELETE DD ENTRY*
      end if
    else
      delete the DFD ELEMENT occurrence
          (DELETED DFD ELEMENT OCCURRENCE)
  end if
end for
```

Notes:

Name: DELETE DFD HIERARCHY

Type: PROCESS
Number: 1.3.4
Description:

```
for each DFD that is part of the DFD hierarchy with the root
    DFD being the DFD specified in DFD, do:
  *DELETE DFD ELEMENTS*
  if DFD ID = TOP DFD VALUE
    then
      reset TOP DFD VALUE by setting UPDATED TOP DFD VALUE
          to blank
  end if
  delete DFD DESCRIPTION for DFD (DELETED DFD)
  if root DFD has a parent process
    then
      create a new DD entry for parent process (NEW ORDERED DD ENTI
  end if
end for
reset OPEN DFD
update DATE PROJECT LAST AMMENDED
```

Notes:

Name: DELETE EXISTING OCCURRENCES

Type: SUB-PROCESS
Description:

```
if PROJECT DD DELETE VALUE = DELETE
  then
    for each DFD ELEMENT ID do:
       *DELETE ALL ASSOCIATED DATA FLOW END POINTS*
       delete DFD ELEMENT occurrence
          (DELETED DFD ELEMENT OCCURRENCE)
       if DD ENTRY associated with DFD ELEMENT ID is not
            used by an other DFD ELEMENT occurrence, DD ENTRY,
            or as MAIN ENTRY for another DD ENTRY
          then
            *DELETE DD ENTRY*
       end if
    end for
  else
    for each DFD ELEMENT ID do:
       *DELETE ALL ASSOCIATED DATA FOW END POINTS ONLY*
       delete DFD ELEMENT occurrence (DELETED DFD ELEMENT OCCURREN(
    end for
end if
```

Notes:

Name: DELETE ONE DD ENTRY

Type: SUB-PROCESS
Description:

```
if DD entry appears as an ALIASED FROM DD ID entry
    in ALIAS ENTRY
  then
    delete ALIAS ENTRY row
  else
    for each DD ENTRY ID used by selected
        DD ENTRY (D17-D21) do:
      delete usage row (DELETED USAGE OF DD ENTRY COMPONENT)
    end for
end if
delete DD row (DELETED ORDERED DD ENTRY)
delete DD ENTRY occurrence  (DELETED DD ENTRY)
update DATE PROJECT LAST AMMENDED to current data/time
    (UPDATED DATA PROJECT LAST AMMENDED)
```

Notes:

Name: DELETE SELECTED DD ENTRY

Type: PROCESS
Number: 1.7.1
Description:

```
*DELETE ONE DD ENTRY* of entry indicated by SELECTED DD ENTRY ID
delete DD WINDOW indicated by SELECTED DD WINDOW
    (DELETED DD WINDOW)
reset SELECTED DD WINDOW
update DATE PROJECT LAST AMMENDED
```

Notes:

Name: DELETE SELECTED OCCURRENCE ONLY

Type: PROCESS
Number: 1.6.1.6.1
Description:

```
if PROJECT DD DELETE VALUE = DELETE
  then
     *DELETE ALL ASSOCIATED DATA FLOW END POINTS*
     delete DFD ELEMENT occurrence
        (DELETED DFD ELEMENT OCCURRENCE)
     if DD ENTRY associated with DFD ELEMENT ID is not
          used by an other DFD ELEMENT occurrence, DD ENTRY,
          or as MAIN ENTRY for another DD ENTRY
        then
          *DELETE DD ENTRY*
      end if
  else
    for each DFD ELEMENT ID do:
      *DELETE ALL ASSOCIATED DATA FOW END POINTS ONLY*
      delete DFD ELEMENT occurrence (DELETED DFD ELEMENT OCCURRENCI
    end for
end if
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: DELETE SELECTED PROJECT

Type: PROCESS
Number: 1.8.7
Description:

```
delete PROJECT NAME indicated by SELECTED PROJECT from
    PROJECT LIST
delete project indicated by SELECTED PROJECT (DELETED PROJECT)
reset SELECTED PROJECT
redisplay PROJECT LIST
```

Notes:

Name: DISPLAY DFD

Type: SUB-PROCESS
Description:

```
use OPEN DFD DESC to create DFD title line and to determine
    window and viewing scale
use OPEN PROJECT NAME to create bottom title on DFD
use FULL DESC OF ELEMENTS IN DFD to drall all DFD elements
if DFD has a parent process
  then
    get DFD NUMBER from PROCESS DETAILS to display
end if
```

Notes:

Name: DRAW/ERASE DFD BORDER OF OPEN DFD

Type: PROCESS
Number: 1.3.8
Description:

```
if PROJECT DFD BORDER VALUE = NO BORDER
  then
    erase existing border for OPEN DFD
  else
    draw a new border around the perimeter of DFD
       indicated by OPEN DFD, ensuring that the
       border is not shown at those points where
       DFD elements lie (as described by DFD ELEMENT
       POSITIONS)
end if
```

Notes:

Name: DUPLICATE SELECTED DATA FLOW

Type: PROCESS
Number: 1.6.4.4
Description:

```
determine objects at end points of DATA FLOW PATH DESCRIPTION
    (END POINT OBJECT IDS) using DFD ELEMENT POSITIONS
if selected data flow (SELECTED OBJECT) has an arg label
  then
    calculate ARG POSITION for new data flow occurrence
end if
similarly for data flow name label
create a NEW DF OCCURRENCE setting DATA FLOW DESC to DATA FLOW
    PATH DESCRIPTION, LINE WIDTH and NUM ARROWS to values of
    selected data flow (SELECTED OBJECT), ARG POSITION and NAME
    POSITION to previously calculated values
if SELECTED OBJECT has a dd entry (D9)
  then
    line NEW DF OCCURRENCE to dd entry
end if
link NEW DF OCCURRENCE to OPEN DFD
create END POINTS OF DUPLICATED DATA FLOW using
    END POINT OBJECT IDS
set SELECTED OBJECT to DFD ELEMENT ID of NEW DATA FLOW
    OCCURRENCE
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: DUPLICATE SELECTED DATA STORE

Type: PROCESS
Number: 1.6.2.6
Description:

```
if there is room for an DATA STORE at
    DUPLICATE SCREEN POSITION
  then
    create NEW DATA STORE OCCURRENCE with POSITION set to
        DUPLICATE SCREEN POSITION and LINE WIDTH set
        to value for SELECTED OBJECT
    link NEW DATA STORE OCCURRENCE to OPEN DFD
    if SELECTED OBJECT has a DD ENTRY
      then
        link NEW DATA STORE OCCURRENCE to same DD ENTRY
    end if
    update DATE PROJECT LAST AMMENDED and
        DATE DFD LAST AMMENDED
  else
    sound audible alarm
end if
```

Notes:

Name: DUPLICATE SELECTED EE

Type: PROCESS
Number: 1.6.2.3
Description:

```
if there is room for an EE at DUPLICATE SCREEN POSITION
  then
    create NEW EE OCCURRENCE with POSITION set to
        DUPLICATE SCREEN POSITION and LINE WIDTH set
        to value for SELECTED OBJECT
    link NEW EE OCCURRENCE to OPEN DFD
    if SELECTED OBJECT has a DD ENTRY
      then
        link NEW EE OCCURRENCE to same DD ENTRY
    end if
    update DATE PROJECT LAST AMMENDED and
        DATE DFD LAST AMMENDED
  else
    sound audible alarm
end if
```

Notes:

Name: EDIT SELECTED DD ENTRY

Type: PROCESS
Number: 1.7.6
Description:

Notes:

Undefined.

Name: ERASE OPEN DFD ELEMENTS

Type: PROCESS
Number: 1.6.1.5
Description:

```
for each DFD element in OPEN DFD do:
  add DD ENTRY ID of element to ERASED ELEMENTS
  delete relevant DF END POINTS row
      (DELETED DATA FLOW END POINTS)
  delete element occurrence (DELETED DFD ELEMENT OCCURRENCE)
end for
if one or more DFD elements were erased
  then
    update DATA PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
end if
```

Notes:

Name: LINK SPECIFIED DFD HIERARCHY TO SELECTED PROCESS

Type: PROCESS
Number: 1.6.3.4
Description:

```
search for DFD with DFD NAME = SPECIFIED DFD NAME or
    parent process PROCESS NUMBER = SPECIFIED DFD NUMBER
if not found
  then
    display error dialogue box
  else
    link selected process (SELECTED OBJECT) to found DFD
    update found DFD NAME by setting NEW DFD NAME to
      name of selected process (SELECTED PROCESS NAME)
    delete process entry from DD (DELETED ORDERED DD ENTRY)
    update DATE PROJECT LAST AMMENDED
end if
```

Notes:

Name: MAKE BACKUP COPY OF SELECTED DD ENTRY

Type: PROCESS
Number: 1.7.2
Description:

copy DD entry indicated by SELECTED OBJECT ID from
database (SELECTED DD ENTRY) to COPY OF SELECTED DD ENTRY

Notes:

Name: MAKE WORKING COPY OF OPEN PROJECT

Type: PROCESS
Number: 1.1.1
Description:

copy PROJECT DATABASE from ORIGINAL MUSSAT DB
(EXISTING PROJECT) for project named as
OPEN PROJECT to OPEN PROJECT DB
Notes:

Name: MODIFY SELECTED OBJECT NAME

Type: PROCESS
Number: 1.6.1.2
Description:

```
if MODIFIED ELEMENT NAME exists in EXISTING DD ENTRY NAMES
  then
    display error dialogue box
  else
    update DD entry name of SELECTED OBJECT in DD
        (UPDATED ORDERED DD ENTRY)
   update DD entry name in DD entry for SELECTED OBJECT
       (UPDATED OBJECT NAME)
   if OBJECT TYPE for SELECTED OBJECT is = process
     then
       if SELECTED OBJECT has a child DFD
        then
           update child DFD NAME by setting UPDATED DFD NAME to
             MODIFIED ELEMENT NAME
       end if
   end if
end if
```

Notes:

Name: MOVE SELECTED DATA FLOW NAME LABEL

Type: PROCESS
Number: 1.6.4.2
Description:

```
update NAME POSITION of SELECTED OBJECT name label using
    SELECTED SCREEN POSITION (UPDATE NAME POSITION)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: MOVE SELECTED DATA STORE

Type: PROCESS
Number: 1.6.2.5
Description:

update position of SELECTED OBJECT using SELECTED SCREEN
    POSITION (UPDATED DATA STORE POSITION)
update DATA FLOW DESC of each connected data flow
    (UPDATED DATA FLOW DESC)
recalculate and update position of NAME POSITION and ARG POSITION
    (UPDATED DATA FLOW LABEL POSITIONS)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: MOVE SELECTED EE

Type: PROCESS
Number: 1.6.2.2
Description:

update position of SELECTED OBJECT using SELECTED SCREEN
    POSITION (UPDATED EE POSITION)
update DATA FLOW DESC of each connected data flow
    (UPDATED DATA FLOW DESC)
recalculate and update position of NAME POSITION and ARG POSITION
    (UPDATED DATA FLOW LABEL POSITIONS)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: MOVE SELECTED PROCESS

Type: PROCESS
Number: 1.6.3.3
Description:

update position of SELECTED OBJECT using SELECTED POSITION
    (UPDATED PROCESS POSITION)
for each connected data flow (CONNECTED DATA FLOW IDS) do:
  update DATA FLOW DESC (UPDATED DATA FLOW DESC)
  recalculate new positions for name and arg labels
  update ARG and NAME POSITION (UPDATED DATA FLOW LABEL POSITIONS)
      with calculated positions
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED

Notes:

Name: MOVE SELECTED WINDOW

Type: PROCESS
Number: 1.5.5
Description:

redraw SELECTED DD WINDOW at position indicated by
    SELECTED SCREEN POSITION
update WINDOW POSITION by setting OPEN WINDOW POSITION
    to position indicated by SELECTED SCREEN POSITION

Notes:

Name: NAME OBJECT WITH EXISTING NAME

Type: SUB-PROCESS
Description:

```
link DFD ELEMENT OCCURRENCE indicated by
    SELECTED OBJECT to DD entry with same name
    as ELEMENT NAME
if OBJECT TYPE of SELECTED OBJECT is data flow
  then
    calculate and update NAME POSITION for SELECTED OBJECT
    if DD ENTRY with same name as ELEMENT NAME has a non-blank
        ARG NAME
      then
        calculate and update ARG POSITION for SELECTED OBJECT
    end if
end if
```

Notes:

Name: NAME OBJECT WITH UNIQUE NAME

Type: SUB-PROCESS
Description:

```
create a NEW DFD ELEMENT DD ENTRY for SELECTED OBJECT
    with name = ELEMENT NAME and all other fields set
    to default values
create NEW DD ENTRY using ELEMENT NAME and DD ENTRY ID
    of NEW DFD ELEMENT DD ENTRY
if OBJECT TYPE of SELECTED OBJECT = data flow
  then
    calculate and update NAME POSITION
end if
```

Notes:

```
Name: NAME SELECTED OBJECT

Type: PROCESS
Number: 1.6.1.3
Description:

if ELEMENT NAME exists in EXISTING DD ENTRY NAMES & IDS
  then
    if OBJECT TYPE of SELECTED OBJECT = process
      then
        display error dialogue box
      else
        if SELECTED OBJECT is not the same type as the DD entry in
            EXISTING DD ENTRY NAMES & IDS
          then
            display error dialogue box
          else
            use prompt dialogue box to ask whether to create dupli
            if answer = yes
              then
                *NAME OBJECT WITH EXISTING NAME* using ELEMENT NAM
                update DATE PROJECT LAST AMMENDED and
                    DATE DFD LAST AMMENDED
            end if
        end if
    end if
  else
    if ELEMENT NAME exists in ERASED ELEMENTS
      then
        *NAME OBJECT WITH EXISTING NAME* using ELEMENT NAME
        update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMEND
        if OBJECT TYPE of SELECTED OBJECT = process
          then
            delete ELEMENT NAME from ERASED ELEMENTS
        end if
    else
      *NAME OBJECT WITH UNIQUE NAME* using ELEMENT NAME
      update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
    end if
end if

Notes:
```

Name: NUMBER DATA STORE WITH EXISTING NUMBER

Type: SUB-PROCESS
Description:

link SELECTED ELEMENT to DATA STORE DD ENTRY with
    DATA STORE NUMBER = SPECIFIED DATA STORE NUMBER

Notes:

Name: NUMBER DATA STORE WITH UNIQUE NUMBER

Type: SUB-PROCESS
Description:

create NEW DATA STORE DD ENTRY with DATA STORE NUMBER
    set to SPECIFIED DATA STORE NUMBER, all other
    fields set to default values
link SELECTED OBJECT to NEW DATA STORE DD ENTRY

Notes:

Name: OPEN CHILD DFD

Type: PROCESS
Number:1.4.3
Description:

set OPEN DFD to DFD ID of DFD where DFD NUMBER
    = PROCESS NUMBER for process indicated by
    SELECTED PROCESS ID
DISPLAY DFD indicated by OPEN DFD

Notes:

Name: OPEN PARENT DFD

Type: PROCESS
Number: 1.4.4
Description:

set OPEN DFD to DFD ID of parent DFD (PARENT DFD ID)
    of SELECTED OBJECT
DISPLAY DFD indicated by OPEN DFD

Notes:

Name: OPEN SELECTED PROJECT

Type: PROCESS
Number: 1.8.2
Description:

erase PROJECT LIST from screen
set OPEN PROJECT to SELECTED PROJECT value

Notes:

Name: OPEN SPECIFIED DFD

Type: PROCESS
Number: 1.4.1
Description:

```
if DFD specified by either SPECIFIED DFD NAME or
    SPECIFIED DFD NUMBER does not exist
  then
    display error dialogue box
  else
    set OPEN DFD to DFD ID of found DFD
    *DISPLAY DFD* indicated by OPEN DFD
end if
```

Notes:

Name: PREPARE DD ENTRY OF SELECTED OBJECT

Type: PROCESS
Number: 1.5.1.4
Description:

CREATE NEW DD WINDOW for DFD element indicated
by SELECTED OBJECT ID

Notes:

Name: PREPARE DD ENTRY SPECIFIED BY NAME

Type: PROCESS
Number: 1.5.1.1
Description:

```
if SPECIFIED ENTRY NAME contains the wildcard character
  then
    if a DD entry exists that matches the specified string
        in EXISTING DD ENTRY NAMES & IDs
      then
        *CREATE NEW DD WINDOW* for found entry
      else
        display error dialogue box
  else
    if an entry exists that exactly matches SPECIFIED ENTRY NAME
        in EXISTING DD ENTRY NAMES & IDS
      then
        *CREATE NEW DD WINDOW* for found entry
      else
        display error dialogue box
    end if
end if
```

Notes:

Name: PRINT CURRENT DFD

Type: PROCESS
Number: 1.2.1
Description:

PRINT DFD for DFD indicated by OPEN DFD producing
    (PRINTED CURRENT DFD)

Notes:

Name: PRINT DFD

Type: SUB-PROCESS
Description:

```
if PROJECT DFD BORDER VALUE = full
  then
    format DFD for printing with a border
  else
    format DFD without a border
end if
print each page of DFD
```

Notes:

Name: PRINT DFD BALANCING RESULTS

Type: SUB-PROCESS
Description:

```
print DFD heading as per report mock-up
if DFD has no net flows in
  then
    print "None" in net flows in column
  else
    for each net data flow into DFD do:
      *PRINT NET FLOWS INTO DFD BALANCING RESULTS*
    end for
end if
if DFD has no net flows out
  then
    print "None" in net flows in column
  else
    for each net data flow out of DFD do:
      *PRINT NET FLOWS OUT OF DFD BALANCING RESULTS*
    end for
end if
```

Notes:

Name: PRINT DFD HIERARCHY

Type: PROCESS
Number: 1.2.4
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    get PROJECT NAME & DATES
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON
    sort DFD IDS in depth first order
    for each DFD ID do:
      *PRINT DFD*
    end for
end if
```

Notes:

Name: PRINT NET FLOWS INTO DFD BALANCING RESULTS

Type: SUB-PROCESS
Description:

```
order names of net data flows into DFD by data flow name
  (in the case that not matched with a parent process data flow
   in), or on the name of the matched parent process data flow in
for each of these data flows do:
  if no match was found
    then
      print data flow name enclosed in "*"
    else
      print data flow name
      if match was made at sub-data flow level
        then
          print names of all matched and unmatched
            sub-components as per report mock-up
      end if
  end if
end for
```

Notes:

Name: PRINT NET FLOWS OUT OF DFD BALANCING RESULTS

Type: SUB-PROCESS
Description:

```
order names of net data flows out of DFD by data flow name
  (in the case that not matched with a parent process data flow
    out), or on the name of the matched parent process data flow ou
for each of these data flows do:
  if no match was found
    then
      print data flow name enclosed in "*"
    else
      print data flow name
      if match was made at sub-data flow level
        then
          print names of all matched and unmatched
            sub-components as per report mock-up
      end if
  end if
end for
```

Notes:

Name: PRINT PARENT PROCESS BALANCING RESULTS

Type: SUB-PROCESS
Description:

```
print parent process heading as per report mock-up
if no parent process exists for DFD
  then
    print "None" in net flows columns as per report mock-up
  else
    if parent process has no net flows in
      then
        print "None" in net flows in column
      else
        sort net flow in names into alphabetic order
        for each net flow in do:
          if a match was found in child DFD
            then
              print data flow name
            else
              print data flow name enclosed in
                   "*" to indicate no match
          end if
        end for
    end if
    if parent process has no net flows out
      then
        print "None" in net flows out column
      else
        sort net flow out names into alphabetic order
        for each net flow out do:
          if a match was found in child DFD
            then
              print data flow name
            else
              print data flow name enclosed in
                   "*" to indicate no match
          end if
        end for
    end if
end if
```

Notes:

Name: PRODUCE DFD BALANCING REPORT

Type: PROCESS
Number: 1.2.6
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    retrieve PROJECT NAME & DATES
    print heaer page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON
    sort DFD IDS in depth first order
    for each DFD ID do:
     *BALANCE FLOWS INTO PARENT PROCESS WITH NET FLOWS
      INTO DFD*
     *BALANCE FLOWS OUT OF PARENT PROCESS WITH NET FLOWS
      OUT OF DFD*
     *PRINT PARENT PROCESS BALANCING RESULTS*
     *PRINT DFD BALANCING RESULTS*
end if
```

Notes:

Name: PRODUCE DFD STATISTICS REPORT

Type: PROCESS
Number: 1.2.5
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    get PROJECT NAME & DATES
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON
    sort DFD IDS in depth first order
    for each DFD indicated in sorted DFDS TO REPORT ON do:
      retrieve DFD DETAILS
      print DFD header as per report mock-up
      retrieve IDS OF ALL ELEMENTS IN DFD
      reset all totals
      for each element in DFD do:
        if the DFD element is named (i.e. an associated DD entry
            exists)
          then
            if the associated name has not already been counted
              then
                add 1 to appropriate DFD element count
            end if
          else
            an unnamed element has been found
        end if
      end for
      print totals as per report mock-up
      if all elements were named
        then
          print "All DFD elements included."
        else
          print "Unnamed DFD elements found and not included."
    end for
end if
```

Notes:

Name: PRODUCE FULL DFD DD ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.1.1
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    retrieve PROJECT NAME & DATES
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON (DFDS TO REPORT ON)
    retrieve the names of all named DFD elements within these DFDS
       except the names of non-leaf level processes
       (DFD ELEMENT NAMES)
    sort names into alphabetic order, removing duplicates
    for each DFD ELEMENT NAME do:
      *RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    end for
    if any unnamed elements were found in any of the DFDS TO REPORT
      then
        print "Unnamed DFD element(s) found and not included."
    end if
end if
```

Notes:

Name: PRODUCE FULL PRINTED DD ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.3.1
Description:

```
if a wildcard character is not used in START NAME and
    a matching DD ENTRY NAME is not found in
    EXISTING DD ENTRY NAMES & IDS
  then
    display error dialogue box
  else
    if a wildcard character is used
      then
        find first matching DD ENTRY NAME in
            EXISTING DD ENTRY NAMES & IDS
    end if
    retrieve PROJECT NAME & DATES
    print header page as per report mock-up
    for each DD ENTRY NAME in EXISTING DD ENTRY NAMES & IDS repeat
      *RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    until FINISH NAME entry is printed or end of DD ENTRY NAMES &
       is reached
end if
```

Notes:

Name: PRODUCE SUMMARY UNUSED ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.2.1
Description:

```
if a wildcard character is not used in START NAME and
    a matching DD ENTRY NAME is not found in
    EXISTING DD ENTRY NAMES & IDS
  then
    display error dialogue box
  else
    if a wildcard character is used
      then
        find first matching DD ENTRY NAME in
            EXISTING DD ENTRY NAMES & IDS
    end if
    for each DD ENTRY NAME in EXISTING DD ENTRY NAMES & IDS do:
      if entry is not used as the main entry of another dd entry,
           and has no asociated DFD elements and is not used in the
           definition field of any other DD entry
        then
          *RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING*
          print formatted entry
      end if
end if
```

Notes:

Name: PRODUCE FULL DFD DD ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.2.2
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    retrieve PROJECT NAME & DATES
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON (DFDS TO REPORT ON)
    retrieve the names of all named DFD elements within these DFDS
       except the names of non-leaf level processes
       (DFD ELEMENT NAMES)
    sort names into alphabetic order, removing duplicates
    for each DFD ELEMENT NAME do:
      *RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    end for
    if any unnamed elements were found in any of the DFDS TO REPOR
      then
        print "Unnamed DFD element(s) found and not included."
    end if
end if
```

Notes:

Name: PRODUCE FULL MAIN & SUB ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.4.1
Description:

```
if MAIN ENTRY NAME does not exist in EXISTING DD ENTRY NAMES
  then
    display error dialogue box
  else
    retrieve the names of all DD entries that are components
        of, or are components of the MAIN ENTRY NAME (i.e. the
        specified MAIN ENTRY NAME has an alias)
        do this recursively for each entry used by each of these
        components, and so forth (EXISTING DD ENTRY NAMES)
    sort names into alphabetic sequence
    print "MAIN ENTRY" heading as per report mock-up
    *RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING* for MAIN ENTRY N/
    print formatted entry
    print "SUBORDINATE ENTRIES" heading as per report mock-up
    for each  DD ENTRY NAME in EXISTING DD ENTRY NAMES do:
      *RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    end for
end if
```

Notes:

Name: PRODUCE SUMMARY MAIN & SUB ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.4.2
Description:

```
if MAIN ENTRY NAME does not exist in EXISTING DD ENTRY NAMES
  then
    display error dialogue box
  else
    retrieve the names of all DD entries that are components
        of, or are components of the MAIN ENTRY NAME (i.e. the
        specified MAIN ENTRY NAME has an alias)
        do this recursively for each entry used by each of these
        components, and so forth (EXISTING DD ENTRY NAMES)
    sort names into alphabetic sequence
    print "MAIN ENTRY" heading as per report mock-up
    *RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING* for MAIN ENTR
    print formatted entry
    print "SUBORDINATE ENTRIES" heading as per report mock-up
    for each  DD ENTRY NAME in EXISTING DD ENTRY NAMES do:
      *RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    end for
end if
```

Notes:

Name: PRODUCE SUMMARY PRINTED DD ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.3.2
Description:

```
if a wildcard character is not used in START NAME and
    a matching DD ENTRY NAME is not found in
    EXISTING DD ENTRY NAMES & IDS
  then
    display error dialogue box
  else
    if a wildcard character is used
      then
        find first matching DD ENTRY NAME in
            EXISTING DD ENTRY NAMES & IDS
    end if
    retrieve PROJECT NAME & DATES
    print header page as per report mock-up
    for each DD ENTRY NAME in EXISTING DD ENTRY NAMES & IDS repeat:
      *RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING*
      print formatted entry
    until FINISH NAME entry is printed or end of DD ENTRY NAMES &
       is reached
end if
```

Notes:

Name: PRODUCE SUMMARY UNUSED ENTRIES REPORT

Type: PROCESS
Number: 1.2.2.2.1
Description:

```
if a wildcard character is not used in START NAME and
    a matching DD ENTRY NAME is not found in
    EXISTING DD ENTRY NAMES & IDS
  then
    display error dialogue box
  else
    if a wildcard character is used
      then
        find first matching DD ENTRY NAME in
            EXISTING DD ENTRY NAMES & IDS
    end if
    for each DD ENTRY NAME in EXISTING DD ENTRY NAMES & IDS do:
      if entry is not used as the main entry of another dd entry,
           and has no asociated DFD elements and is not used in the
           definition field of any other DD entry
        then
          *RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING*
          print formatted entry
      end if
end if
```

Notes:

Name: PRODUCE UNDEFINED ELEMENTS REPORT

Type: PROCESS
Number: 1.2.7
Description:

```
if DFD specified as TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    retrieve PROJECT NAME & DATES
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON
    sort DFD IDs in depth first order
    for each DFD ID do:
      retrieve DFD NAME (NAME OF EXISTING DFD)
      retrieve PARENT PROCESS NUMBER (if one exists)
      print DFD heading as per report mock-up
      retrieve the names of all DFD elements within the DFD that
        are not named and have a blank DEFN field and are not
        the parent process of a child DFD
      sort these names into alphabetic order
      print names as per report mock-up
    end for
end if
```

Notes:

```
Name: PRODUCE UNNAMED ELEMENTS REPORT

Type: PROCESS
Number: 1.2.3
Description:

if DFD specified in TOP DFD SPECIFIER does not exist in
    EXISTING DFD NAMES
  then
    display error dialogue box
  else
    print header page as per report mock-up
    determine DFDS TO REPORT ON using TOP DFD SPECIFIER and
        NUMBER OF LEVELS TO REPORT ON
    sort DFD IDS in depth first order
    for each DFD ID in sorted DFDS TO REPORT ON do:
      retrieve IDS OF ALL ELEMENTS IN DFD
      reset all totals and sub-totals
      for each DFD ELEMENT ID do:
        depending on the type of DFD element:

          process, EE or data store:
                       if no associated DD ENTRY exists
                         then
                           add 1 to number of unnamed processes, EI
                              or data stores
                       end if
          data flows:  if no associated DATA FLOW DD ENTRY exists
                         then
                           get DATA FLOW END POINT IDS for
                               DATA FLOW OCCURRENCE
                           if either SOURCE or DESTN OBJECT ID
                               indicates a connected data sotre
                             then
                               add 1 to unnamed data flows to/from
                                   data stores
                             else
                               add 1 to other unnamed data stores
                           end if
                       end if
        end depending
        add together all unnamed DFD element totals to give grand t
        print results as per report mock-up
end if
                  then
                    add 1 to number of unnamed EEs
                end if
          data store: if no assocaited DATA STORE DD ENTRY exists
                        then
                          add
Notes:
```

Name: REDRAW SELECTED DATA FLOW

Type: PROCESS
Number: 1.6.4.3
Description:

```
determine objects at end points of DF PATH DESCRIPTION
    (END POINT OBJECT IDS) using DFD ELEMENT POSITIONS
calculate new ARG POSITION and/or NAME POSITION for
    selected data flow on DATA FLOW PATH DESC
update calculated label positions and
    DATA FLOW PATH DESCRIPTION for DATA FLOW OCCURRENCE
update DATA FLOW END POINTS for selected data flow using
    NEW END POINT OBJECT IDS
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: REFORMAT ALL DD WINDOWS

Type: PROCESS
Number: 1.5.8
Description:

```
if PROJECT DD DISPLAY VALUE = FULL
  then
    for each DD WINDOW in OPEN DD WINDOWS do:
      *REFORMAT TO FULL DD ENTRY FOR DISPLAY*
      update FORMATTED DD ENTRY TEXT in DD WINDOW
          with reformatted DD ENTRY TEXT
      redisplay DD WINDOW
  else
    for each DD WINDOW in OPEN DD WINDOWS do:
       *REFORMAT TO SUMMARY DD ENTRY FOR DISPLAY*
       update FORMATTED DD ENTRY TEXT in DD WINDOW
          with reformatted DD ENTRY TEXT
       redisplay DD WINDOW
```

Notes:

Will be used when the user changes PROJECT DD DISPLAY VALUE and there are one or more DD entry windows open.

Name: REFORMAT TO FULL DD ENTRY FOR DISPLAY

Type: SUB-PROCESS
Description:

```
if the DD entry is a DFD element
  then
    for each occurrence of the DFD element (D4-D7) do:
       retrieve and format OCCURRENCE DETAILS
end if
if the DD entry is a data flow or any other non-DFD element
  then
    retrieve and format the names of all DD entries that use
        this entry
end if
```

Notes:

Name: REFORMAT TO SUMMARY DD ENTRY DD DISPLAY

Type: SUB-PROCESS
Description:

```
delete all occurrence and usage details from the
    DD WINDOW
```

Notes:

```
Name: RENAME DFD

Type: PROCESS
Number: 1.3.2
Description:

if NEW DFD NAME already exists in EXISTING DFD NAMES
  then
    display error dialogue box
  else
    update DFD NAME by setting UPDATED DFD NAME to
        NEW DFD NAME for DFD indicated by OPEN DFD
    if DFD indicated by OPEN DFD has a parent process
      then
        update PROCESS NAME of parent process by setting
            the value of UPDATED PARENT PROCESS NAME to
            NEW DFD NAME
    end if
    update DATA PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
end if

Notes:
```

```
Name: RENAME PROJECT

Type: PROCESS
Number: 1.8.4
Description:

get names of all existing projects (EXISTING
    PROJECT NAMES)
if NEW PROJECT NAME exists in EXISTING PROJECT NAMES
  then
    display error dialogue box
  else
    update PROJECT name to NEW PROJECT NAME
        (UPDATED PROJECT NAME) in ORIGINAL MUSSAT DB
        and PROJECT LIST
    redisplay PROJECT LIST
end if
Notes:
```

Name: RENUMBER DATA STORE

Type: SUB-PROCESS
Description:

update DATA STORE NUMBER of DATA STORE DD ENTRY
    associated with SELECTED OBJECT to
    SPECIFIEC DATA STORE NUMBER (UPDATED DATA
        STORE NUMBER)

Notes:

Name: (RE) NUMBER SELECTED DATA STORE

Type: PROCESS
Number: 1.6.2.7
Description:

```
if no DATA STORE DD ENTRY exists for
    SELECTED OBJECT
  then
    if SPECIFIED DATA STORE NUMBER does not
        exist in EXISTING DATA STORE NUMBERS
      then
        *NUMBER DATA STORE WITH UNIQUE NUMBER*
        update DATE PROJECT LAST AMMENDED and
            DATE DFD LAST AMMENDED
      else
        if ID of DATA STORE DD ENTRY with
            matching number exists in ERASED ELEMENTS
          then
            *NUMBER DATA STORE WITH EXISTING NUMBER*
            update DATE PROJECT LAST AMMENDED and
               DATE DFD LAST AMMENDED
          else
            use a prompt dialogue box to ask whether to
                create a duplicate
            if answer = yes
              then
                *NUMBER DATA STORE WITH EXISTING NUMBER*
            end if
        end if
    end if
  else  (* already named and/or numbered *)
    if SPECIFIED DATA STORE NUMBER exists in
        EXISTING DATA STORE NUMBERS
      then
        dislay error dialogue box
      else
        *RENUMBER DATA STORE*
end if
```

Notes:

Name: (RE)NUMBER SELECTED PROCESS

Type: PROCESS
Number: 1.6.3.2
Description:

get the PROCESS NUMBER of all processes in OPEN DFD
    other than SELECTED OBJECT (OPEN DFD PROCESS
    NUMBERS)
if no existing PROCESS NUMBER ends in the same digit
    as specified in NEW PROCESS NUMBER
  then
    update PROCESS NUMBER of SELECTED OBJECT by
        replacing last digit of existing
        PROCESS NUMBER with NEW PROCESS NUMBER
        (UPDATED PROCESS NUMBER)
  else
    display error dialogue box
end if

Notes:

Name: REPLACE OPEN PROJECT WITH SAVED VERSION

Type: PROCESS
Number: 1.1.6
Description:

copy project indicated by OPEN PROJECT (EXISTING PROJECT)
    from ORIGINAL MUSSAT DB to OPEN PROJECT DB
Notes:

Name: RESET EDIT ENTRY

Type: PROCESS
Number: 1.7.8
Description:

reset EDIT ENTRY
remove "E" from SELECTED DD WINDOW close box

Notes:

Name: RESTORE SAVED VERSION OF DD ENTRY

Type: PROCESS
Number: 1.7.3
Description:

copy COPY OF SELECTED DD ENTRY back to DD entry indicated
    by SELECTED DD ENTRY ID (RESTORED DD ENTRY)
reformat window contents (UPDATED WINDOW CONTENTS)

Notes:

Name: RETRIEVE FULL DD ENTRY DETAILS FOR DISPLAY

Type: SUB-PROCESS
Description:

```
retrieve and format DD ENTRY DETAILS for specified entry
if DD entry is a DFD element entry
  then
    retrieve and format OCCURRENCE DETAILS  and
        APPEARS IN DFD NAME for each occurrence
end if
if DD entry is a data flow, or non-DFD element entry
  then
    retrieve and format names of all DD entries that
        use this entry in defn fields and as MAIN ENTRY
end if
```

Notes:

Name: RETRIEVE FULL DD ENTRY DETAILS FOR PRINTING

Type: SUB-PROCESS
Description:

```
retrieve and format DD ENTRY DETAILS for specified entry
if DD entry is a DFD element entry
  then
    retrieve and format OCCURRENCE DETAILS  and
        APPEARS IN DFD NAME for each occurrence
end if
if DD entry is a data flow, or non-DFD element entry
  then
    retrieve and format names of all DD entries that
        use this entry in defn fields and as MAIN ENTRY
end if
```

Notes:

Name: RETRIEVE SUMMARY DD ENTRY DETAILS FOR DISPLAY

Type: SUB-PROCESS
Description:

retrieve and format DD ENTRY DETAILS for
specified DD entry

Notes:

Name: RETRIEVE SUMMARY DD ENTRY DETAILS FOR PRINTING

Type: SUB-PROCESS
Description:

retrieve and format DD ENTRY DETAILS for
specified DD entry

Notes:

Name: SAVE WORKING COPY OF OPEN PROJECT

Type: PROCESS
Number: 1.1.10
Description:

replace PROJECT DATABASE of OPEN PROJECT in
ORIGINAL MUSSAT DB with OPEN PROJECT DB (EXISTING PROJECT)

Notes:

Name: SCROLL DFD

Type: PROCESS
Number: 1.3.3
Description:

update DFD WINDOW VIEW by recalculating visible postion
of DFD using NEW HORIZ SCROLL BOX POSITION or
NEW VERT SCROLL BOX POSITION and DFD ZOOM LEVEL values
(UPDATED DFD WINDOW VIEW)

Notes:

Name: SCROLL PROJECT LIST

Type: PROCESS
Number: 1.8.5
Description:

redisplay PROJECT LIST so that the portion of
PROJECTLIST indicated by NEW VERT SCROLL BOX
POSITION is displayed
update PORTION VISIBLE to values indicated by
NEW VERT SCROLL BOX POSITION (UPDATED PORTION
VISIBLE)

Notes:

Name: SCROLL SELECTED DD ENTRY WINDOW

Type: PROCESS
Number: 1.5.7
Description:

redisplay SELECTED DD WINDOW so that the portion
    of FORMATTED DD ENTRY TEXT or ERRORS LISTING
    indicated by NEW VERT SCROLL BOX POSITION is
    displayed
update EDIT ERRORS WINDOW VIEW or DD WINDOW VIEW
    by setting UPDATED WINDOW VIEW to value
    indicated by NEW VERT SCROLL BOX POSITION

Notes:

Name: SELECT DD WINDOW

Type: PROCESS
Number: 1.5.4
Description:

set SELECTED DD WINDOW to WINDOW ID of
    window indicated by SELECTED SCREEN POSITION
        (CORRESPONDING WINDOW ID)
bring window to front and display with hightlighted
    window frame

Notes:

Name: SELECT DFD OBJECT

Type: PROCESS
Number: 1.6.5
Description:

```
determine ID of object located at SELECTED
    SCREEN POSITION
if no SELECTED SCREEN POSITION is unoccupied
  then
    reset SELECTED OBJECT
  else
    set OBJECT ID to ID of selected object
    if object is a data flow
      then
        if SELECTED SCREEN POSITION indicates the
          data flow name label
            then
              set OBJECT TYPE to name label
            else
              if SELECTED SCREEN POSITION indicates
                  the arg label
                then
                  set OBJECT TYPE to arg label
              end if
        end if
    end if
    highlight selected object
end if
```

Notes:

Name: SELECT PROJECT FROM PROJECT LIST

Type: PROCESS
Number: 1.8.1
Description:

```
find PROJECT NAME in PROJECT LIST that
    corresponds to SELECTED SCREEN POSITION
set SELECTED PROJECT to PROJECT NAME selected
```

Notes:

Name: SET EDIT ENTRY

Type: PROCESS
Number: 1.7.7
Description:

set EDIT ENTRY to WINDOW ID of SELECTED DD WINDOW
display "E" in SELECTED DD WINDOW close box

Notes:

Name: SPECIFY DATA FLOW ARGUMENT

Type: PROCESS
Number: 1.6.4.6
Description:

```
if DD entry associated with SELECTED OBJECT does not
    have a ARG NAME
  then
    for each occurrence of selected data flow (DATA FLOW
        OCCURRENCE IDS) do:
       calculate position for the new arg label
           using SELECTED DATA FLOW DESC
       set UPDATED ARG POSITION to calculated position
    end for
end if
update ARG NAME by setting UPDATED DATA FLOW ARG to
    NEW DATA FLOW ARGUMENT (UPDATED DATA FLOW ARG)
update DATE PROJECT LAST AMMENDED and DATE DFD LAST AMMENDED
```

Notes:

Name: UNLINK DFD HIERARCHY FROM SELECTED PROCESS

Type: PROCESS
Number: 1.6.3.5
Description:

```
if UNLINKED DFD NAME exists in EXISTING DFD NAMES
  then
    display error dialogue box
  else
    update DFD NAME of child DFD of selected process
       (SELECTED OBJECT) by setting UPDATED CHILD DFD
       NAME to UNLINKED DFD NAME
    unlink selected process (SELECTED OBJECT) and
       child DFD
    create a DD entry for process (NEW ORDERED DD ENTRY)
    update DATE PROJECT LAST AMMENDED
end if
```

Notes:

Name: UNSELECT DD WINDOW

Type: PROCESS
Number: 1.5.6
Description:

```
display SELECTED DD WINDOW with unhighlighted
    window frame
reset SELECTED DD WINDOW
```

Notes:

Name: UNSELECT OBJECT

Type: PROCESS
Number: 1.6.6
Description:

unhighlight SELECTED OBJECT
reset SELECTED OBJECT

Notes:

Name: UPDATE DB TO REFLECT EDITS

Type: SUB-PROCESS
Description:

```
if ALIAS field has been changed
  then
    *ACTION EDITED ALIAS FIELD*
end if
if definition field has been changed
  then
    if entry is not a DATA ELEMENT DD ENTRY
      then
        *ACTION EDITED DEFN FIELD*
    end if
end if
update DD entry indicated by EDIT ENTRY
    with new version (UPDATED SELECTED DD ENTRY)
update DATE PROJECT LAST AMMENDED to current DATE/TIMESTAMP
    (UPDATED DATE PROJECT LAST AMMENDED)
```

Notes:

Example MUSSAT Reports

The following report mock-ups are based on the order processing example introduced in Chapter 2.

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting entry: "A"
Ending entry: "CUST*"
Entry format: full

Data structure name: ADDRESS-LINE
Currency: used
See alias entry:
Aliases:
Volume:
Definition: {ALPHA-CHAR} 30
Notes: One line of an address.
Component of: CUSTOMER-POSTAL-ADDRESS

Process name: CHECK CUSTOMER DETAILS
number: 1
Process type:
Definition: read CUSTOMER-DETAILS from CUSTOMERS using
CUSTOMER-IDENTIFIER
if not found
then invalid CUSTOMER-IDENTIFIER
else
check that CUSTOMER-CREDIT-RATING is ok
if ok
then set CUSTOMER-NUM to CUSTOMER-IDENTIFIER.
Notes: Checks that a valid CUSTOMER-IDENTIFIER is given and that the customer doesn't have a blacklisted credit rating.
Appears in: PROCESS ORDERS
Flows in: CUSTOMER-DETAILS, CUSTOMER-IDENTIFIER
Flows out: CUSTOMER-NUM

Process name: CHECK ORDER
Process number: 2
Process type:
Definition: child dfd defined
Notes:
Appears in: PROCESS ORDERS
Flows in: ORDER-DETAILS, PART-DESC
Flows out: OUT-OF-STOCK-REPORT, VALID-ORDER-DETAILS

Process name: CHECK PART NUM
number: 2.1
Currency: used
Process type:
Definition:
Notes: Checks that a part number as specified by a customer in an order is an existing part number.
Appears in: CHECK ORDER
Flows in: CUST-PART-NUM, PART-NUM
Flows out: VALID-PART-NUM

Process name: CHECK QTY AVAILABLE
number: 2.3
Currency: used
Process type:
Definition: read QTY-ON-HAND from PARTS using VALID-PART-NUM
if QTY-ORDERED <= QTY-ON-HAND
then set QTY-SUPPLIED to VALID-QTY-ORDERED
else set QTY-NOT-SUPPLIED to VALID-QTY-ORDERED.
Notes:
Appears in: CHECK ORDER
Flows in: QTY-ON-HAND, VALID-PART-NUM, VALID-QTY-ORDERED
Flows out: INSUFFICIENT-STOCK-DETAILS, VALID-ORDER-DETAILS

Process name: CHECK QTY ORDERED
Process number: 2.2
Process type:
Definition:
Notes:
Appears in: CHECK ORDER
Flows in: QTY-ORDERED
Flows out: VALID-QTY-ORDERED

Data flow name: CUST-PART-NUM
Currency: used
See alias entry: PART-NUM
Aliases:
Data flow type:
Volume:
Definition:
Notes: A part number as specified by the customer, on a received order.
Appears in: **[1]** CHECK ORDER
Source:
Destination: CHECK PART NUM
Component of: ORDER-LINE

External entity name: CUSTOMER
Notes:
Appears in: **[1]** CONTEXT DFD
Flows in:
Flows out: ORDER
[2] CONTEXT DFD
Flows in: INVOICE, OUT-OF-STOCK-REPORT
Flows out:

[3] PROCESS ORDERS
Flows in: INVOICE, OUT-OF-STOCK-REPORT
Flows out: CUSTOMER-IDENTIFIER, ORDER-DETAILS

Data structure name: CUSTOMER-ADDRESS
Currency: used
See alias entry:
Aliases:
Volume:
Definition: 1 {ADDRESS-LINE} 5
Notes:
Component of: CUSTOMER-POSTAL-ADDRESS, CUSTOMER-RECORD

Data flow name: CUSTOMER-DETAILS
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: CUSTOMER-NAME + CUSTOMER-IDENTIFIER +
CUSTOMER-CREDIT-RATING
Notes:
Appears in: [1] PROCESS ORDERS
Source: CUSTOMERS
Destination: CHECK CUSTOMER DETAILS
Component of: CUSTOMERS

Data flow name: CUSTOMER-IDENTIFIER
Currency: used
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: 1 {digit} 4
Notes:

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: 0 "PROCESS ORDERS"
Number of additional levels: all

Appears in: [1] PROCESS ORDERS
Source: CUSTOMER
Destination: CHECK CUSTOMER DETAILS
Component of: CUSTOMER-DETAILS

Data flow name: CUSTOMER-NAME
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: 1 {ALPHA-CHAR} 30
Notes:
Appears in:
Component of: CUSTOMER-DETAILS

Data flow name: CUSTOMER-POSTAL-ADDRESS
Currency: used
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: CUSTOMER-ADDRESS
Notes:
Appears in: [1] PROCESS ORDERS
Source: CUSTOMER
Destination: PRODUCE INVOICE
Component of:

Data structure name: CUSTOMER-RECORD
See alias entry:
Aliases:
Volume:

Definition: CUSTOMER-DETAILS + CUSTOMER-ADDRESS
Notes:
Component of: CUSTOMERS

Data store name: CUSTOMERS
number:
search arguments:
See alias entry:
Aliases:
Organisation:
Definition: {CUSTOMER-RECORD}
Notes:
Appears in: [1] PROCESS ORDERS
Flows in:
Flows out: CUSTOMER-DETAILS,
CUSTOMER-POSTAL-ADDRESS

DFD 0: "PROCESS ORDERS"

Undefined DFD element totals by element type:

Data flows	
to/from data stores:	2
other:	1
Data stores:	1
External entities:	0
Processes:	1
Total:	5

DFD 2: "CHECK ORDER"

Undefined DFD element totals by element type:

Data flows	
to/from data stores:	0
other:	0
Data stores:	
External entities:	0
Processes:	0
Total:	0

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Main entry: "PARTS"
Entry format: summary

Main entry:

Data store name: PARTS
number:
See alias entry:
Aliases:
Organisation:
Definiton: {PART}
Notes:

Subordinate entries:

Data element name: CENTS-AMOUNT
See alias entry:
Values & meaning: 0 <= DOLLAR-AMOUNT <= 99 = valid range
Notes:

Data element name: DOLLAR-AMOUNT
See alias entry:
Values & meaning: 0 <= DOLLAR-AMOUNT <= 9 999 999 = valid range
Notes:

Data structure name: DOLLAR-VALUE
See alias entry:
Aliases:
Volume:
Definition: DOLLAR-AMOUNT + CENTS-AMOUNT
Notes:

Data structure name: PART
Currency: used
See alias entry:
Aliases:
Volume:
Definition: {PART-NUM + QTY-ON-HAND + UNIT-PRICE + PART-NAME}

Notes:

Unknown type name: PART-NAME
Currency:
See alias entry:
Aliases:
Definition:
Notes:

Data structure name: PART-NUM
Currency: used
See alias entry:
Aliases: CUST-PART-NUM, VALID-PART-NUM
Volume:
Definition: 1 {DIGIT} 6
Notes:

Data element name: QTY-ON-HAND
Currency: used
See alias entry:
Aliases:
Values & meanings: > 0 = in stock
< 10 = reorder level
<= 5 = danger level
Notes:

Data structure name: UNIT-PRICE
See alias entry:
Aliases:
Volume:
Definition: DOLLAR-VALUE
Notes: Per-unit price of an item in stock - excluding GST.

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: "Context DFD"
Number of additional levels: all

DFD: "CONTEXT DFD"

PARENT PROCESS: none

DFD:

NET FLOWS IN	NET FLOWS OUT
None	None

DFD 0: "PROCESS-ORDERS"

PARENT PROCESS:

FLOWS IN	FLOWS OUT
ORDER	INVOICE OUT-OF-STOCK-REPORT

DFD:

NET FLOWS IN	NET FLOWS OUT
ORDER *ORDER-ADDRESS* *ORDER-DATE* ORDER-DETAILS	INVOICE OUT-OF-STOCK-REPORT

Incompletely specified elements found and not included.

DFD NUMBER 2: "CHECK-ORDER"

PARENT PROCESS:

FLOWS IN	FLOWS OUT
ORDER-DETAILS	OUT-OF-STOCK-REPORT
PART-DESC	VALID-ORDER-DETAILS

DFD:

NET FLOWS IN	NET FLOWS OUT
ORDER-DETAILS	OUT-OF-STOCK-REPORT
CUST-ORDER-DATE	VALID-ORDER-DETAILS
CUST-PART-NUM	
CUST-POSTING-ADDRESS	
QTY-ORDERED	
PART-DESC	
PART-NAME	
PART-NUM	
QTY-ON-HAND	
UNIT-PRICE	

All DFD elements included.

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: 2 "CHECK ORDER"
Number of additional levels: 0
Entry format: summary

Unnamed DFD elements found and not included.

Process name: CHECK PART NUM
Process number: 2.1
Process type:
Definition:
Notes: Checks that a part number as specified by a customer
in an order is an existing part number.

Process name: CHECK QTY AVAILABLE
Process number: 2.3
Process type:
Definition: read QTY-ON-HAND from PARTS using VALID-PART-NUM
if QTY-ORDERED <= QTY-ON-HAND
then set QTY-SUPPLIED to VALID-QTY-ORDERED
else set QTY-NOT-SUPPLIED to VALID-QTY-ORDERED.
Notes:

Process name: CHECK QTY ORDERED
Process number: 2.2
Process type:
Definition:
Notes:

Data flow name: CUST-PART-NUM
See alias entry: PART-NUM
Aliases:
Data flow type:
Volume:
Definiton:
Notes: A part number as specified by the customer,
on a received order.

Data flow name: INSUFFICIENT-STOCK-DETAILS
See alias entry:
Aliases:
Data flow type:
Volume:
Definiton: VALID-PART-NUM + QTY-NOT-SUPPLIED
Notes:

Data flow name: OUT-OF-STOCK-REPORT
See alias entry:
Aliases:
Data flow type:
Volume:
Definiton: OUT-OF-STOCK-HEADER + 1 {INSUFFICIENT-STOCK-DETAILS}
Notes:

Data store name: PARTS
number:
See alias entry:
Aliases:
Organisation:
Definition: {PART}
Notes:

Data flow name: QTY-ORDERED
See alias entry:
Aliases: VALID-QTY-ORDERED
Data flow type:
Volume:
Definiton: 1 {DIGIT} 6
Notes:

Data flow name: VALID-ORDER-DETAILS
Currency: used
See alias entry:
Aliases:
Data flow type:
Volume:
Definiton: 1 {VALID-ORDER-LINE}
Notes:

Data flow name: VALID-PART-NUM
See alias entry: PART-NUM
Aliases:
Data flow type:
Volume:
Definiton:
Notes: A part number found to exist in the PARTS file.

Data flow name: VALID-QTY-ORDERED
See alias entry: QTY-ORDERED
Aliases:
Data flow type:
Volume:
Definiton:
Notes:

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: "Context DFD"
Number of additional levels: all

DFD: "CONTEXT DFD"

Date created: 1/6/86 8:20 am
Date last ammended: 10/6/86 5:01 pm
Version:

Number of	**unique**	**data flows**	**: 3**
		data stores	**: 0**
		external entities	**: 1**
		processes	**: 1**

All DFD elements included.

DFD 0: "PROCESS ORDERS"

Date created: 1/6/86 1:10 pm
Date last ammended: 23/12/86 10:00 am
Version:

Number of	**unique**	**data flows**	**:12**
		data stores	**: 3**
		external entities	**: 1**
		processes	**: 4**

Unnamed DFD elements found and not included.

DFD 2: "CHECK ORDER"

Date created: 10/6/86 2:22 pm
Date last ammended: 20/6/86 4:15 pm
Version:

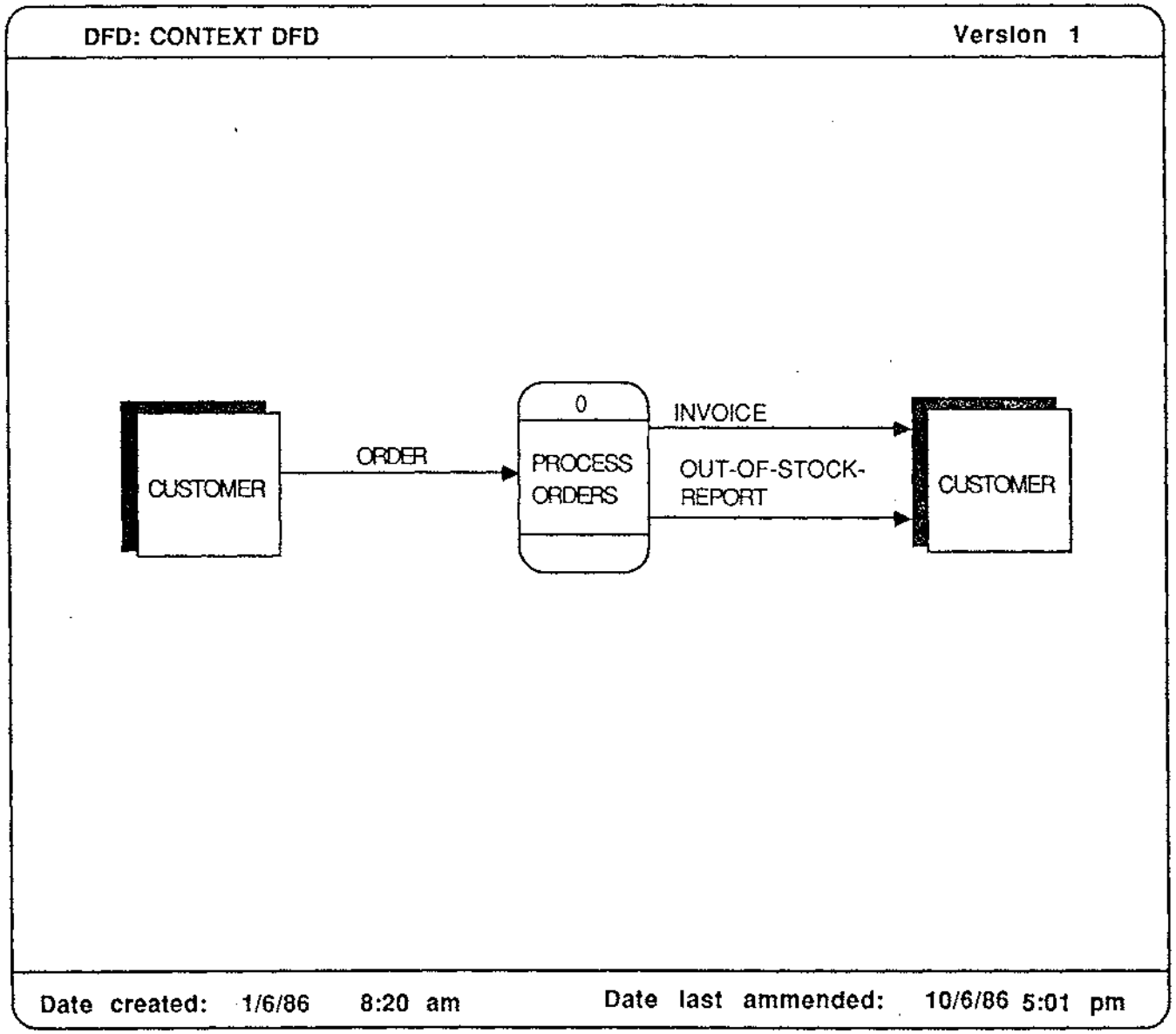
Number of	**unique**	**data flows**	**: 9**
		data stores	**: 1**
		external entities	**: 0**
		processes	**: 4**

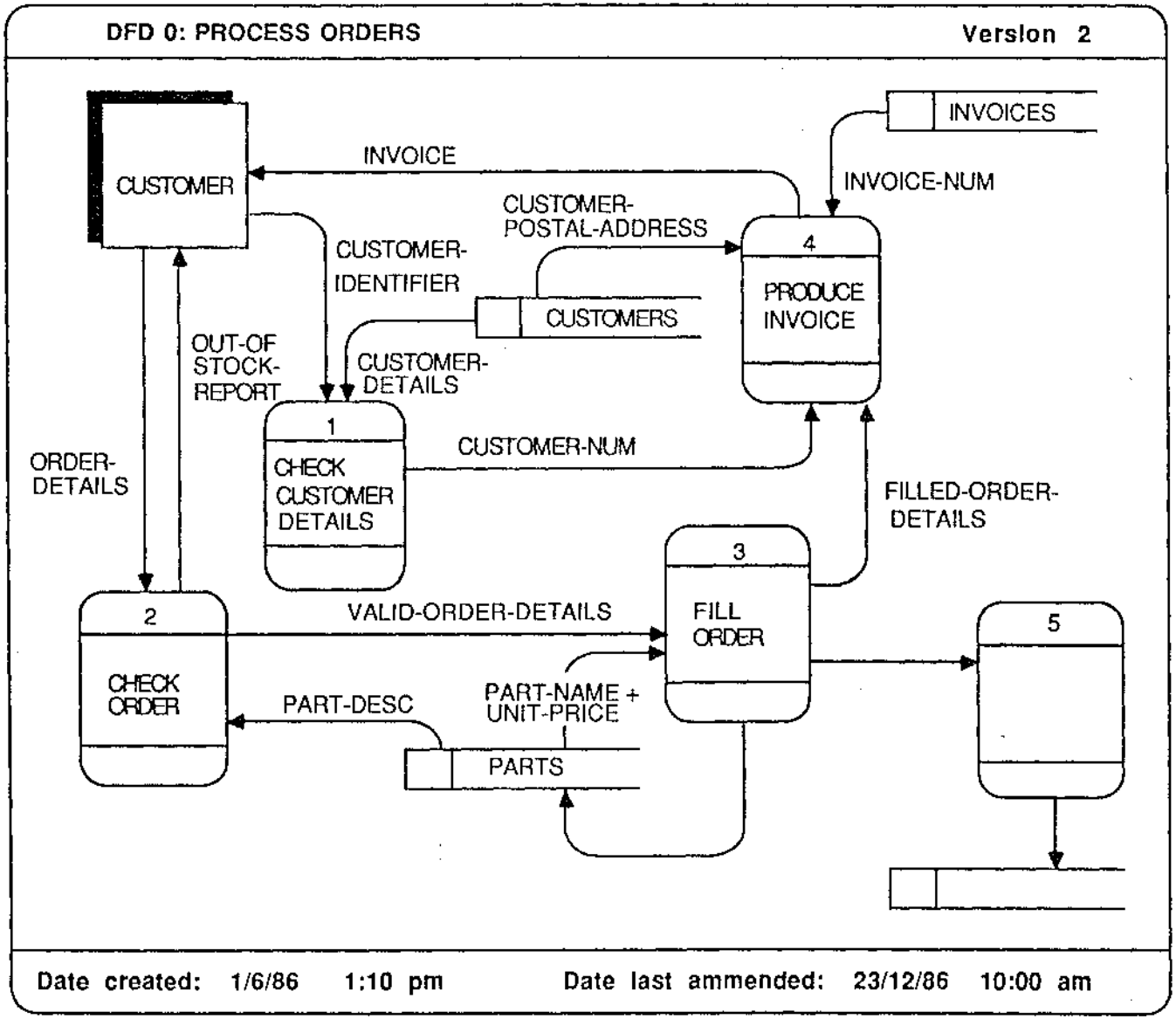
All DFD elements included.

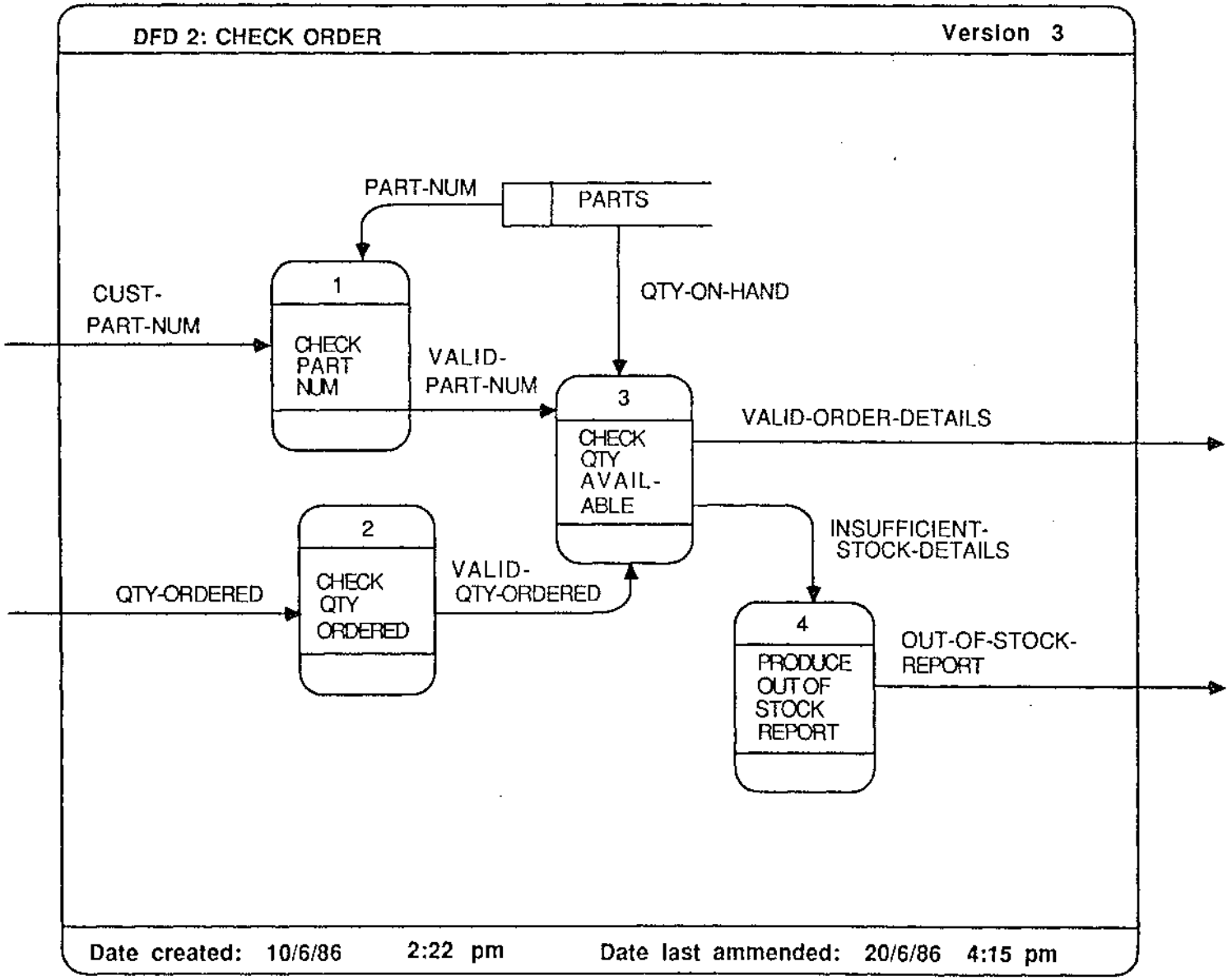
Printed DFDs Report as at 6/1/87 2:32 pm

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: "CONTEXT DFD"
Number of additional levels: all


DFD: CONTEXT DFD
Version 1
CUSTOMER
ORDER
0
PROCESS ORDERS
INVOICE
OUT-OF-STOCK-REPORT
CUSTOMER
Date created: 1/6/86 8:20 am
Date last ammended: 10/6/86 5:01 pm


DFD 0: PROCESS ORDERS
Version 2
CUSTOMER
INVOICE
INVOICES
INVOICE-NUM
CUSTOMER-
POSTAL-ADDRESS
CUSTOMER-
IDENTIFIER
4
PRODUCE
INVOICE
CUSTOMERS
OUT-OF
STOCK-
REPORT
CUSTOMER-
DETAILS
1
CHECK
CUSTOMER
DETAILS
CUSTOMER-NUM
ORDER-
DETAILS
FILLED-ORDER-
DETAILS
3
FILL
ORDER
2
CHECK
ORDER
VALID-ORDER-DETAILS
5
PART-DESC
PART-NAME +
UNIT-PRICE
PARTS
Date created: 1/6/86 1:10 pm
Date last ammended: 23/12/86 10:00 am


DFD 2: CHECK ORDER
Version 3
PART-NUM
PARTS
QTY-ON-HAND
CUST-
PART-NUM
1
CHECK
PART
NUM
VALID-
PART-NUM
3
CHECK
QTY
AVAIL-
ABLE
VALID-ORDER-DETAILS
INSUFFICIENT-
STOCK-DETAILS
2
CHECK
QTY
ORDERED
QTY-ORDERED
VALID-
QTY-ORDERED
4
PRODUCE
OUT OF
STOCK
REPORT
OUT-OF-STOCK-
REPORT
Date created: 10/6/86 2:22 pm
Date last ammended: 20/6/86 4:15 pm

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting DFD: 2 "CHECK ORDER"
Number of additional levels: 0

DFD 2: "CHECK ORDER"

Element Name	Element Type
CHECK PART NUM	PROCESS
CHECK QTY ORDERED	PROCESS
CUST-PART-NUM	DATA FLOW
VALID-PART-NUM	DATA FLOW
VALID-QTY-ORDERED	DATA FLOW

Project: Order Processing Example
Date created: 31/5/86 12:07 pm
Date last ammended: 5/1/87 10:35 am

Starting entry: "UN*"
Ending entry: "VALID*"
Entry format: full

Data flow name: UNFILLED-DETAILS
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: {UNFILLED-ORDER-DETAILS}
Notes:
Appears in:
Component of:

Data flow name: VALID-CUSTOMER-NAME
See alias entry:
Aliases:
Data flow type:
Volume:
Definition: 1{ ALPHANUMERIC CHAR} 30
Notes: Customer name containing only valid alphanumeric characters.
Appears in:
Component of:

Data structure name: VALID-DATE
See alias entry:
Aliases:
Volume:
Definition: VALID-DAY + VALID-MONTH + VAID-YEAR
Notes:
Component of:

Appendix VI

Cross Reference of MUSSAT Commands and SSA Processes

Menu	Command	Process	Condition
SYSTEM	EXIT		
PROJECT	ADD	1.9	
	MODIFY	1.8.2	
		1.1.1	
	RENAME	1.8.4	
	COPY	1.8.3	
	RESTORE	1.1.6	
	SAVE	1.1.10	
	CLOSE	1.1.4	User specifies CLOSE WITH SAVE
		1.1.5	User specifies CLOSE - NO SAVE
	PURGE DD	1.3.9	
	DELETE	1.8.7	
OPTIONS	DD DISPLAY	1.1.3	Value has been changed
		1.5.8	Value has been changed and one or more DD entry windows are visible
	DD DELETE	1.1.7	Value has been changed
	TOP DFD	1.1.2	Value has been changed
	DFD BORDER	1.1.8	Value has been changed
		1.3.8	Value has been changed and a DFD is displayed
DFD	OPEN	1.4.1	
		1.4.2	Another DFD is displayed
	OPEN PARENT	1.4.4	
		1.4.2	Another DFD is displayed
	ADD	1.4.5	
		1.4.2	Another DFD is displayed
	CLOSE	1.4.2	
	DELETE	1.3.4	
		1.4.2	
	SHRINK	1.3.6	Current zoom level is 1
		1.3.7	Current zoom level is 2
	MAX SHRINK	1.3.7	
	ENLARGE	1.3.6	Current zoom level is 3
		1.3.5	Current zoom level is 2
	MAX ENLARGE	1.3.5	
	PRINT	1.2.1	
	AUTO NUMBER	1.6.1.4	
	ERASE	1.6.1.5	
	VERSION	1.3.1	

Menu	Command	Process	Condition
DFD	RENAME	1.3.2	
	DFD SIZE	1.1.9	
	SCROLL	1.3.3	
OBJECT	NAME	1.6.1.3	Selected object is unnamed
		1.6.1.2	Selected object is named
	NUMBER	1.6.2.7	Selected object is a data store
		1.6.3.2	Selected object is a process
	COPY	1.6.2.6	Selected object is a data store
		1.6.2.3	Selected object is an EE
		1.6.4.4	Selected object is a data flow
	(UN)BOLD	1.6.1.1	User specifies SELECTED OCCURRENCE ONLY
		1.6.1.2	User specifies ALL IN DFD HIERARCHY
		1.6.1.3	User specifies ALL IN OPEN DFD
		1.6.1.4	User specifies ALL OCCURRENCES
	ARGUMENT	1.6.4.6	
	REDRAW	1.6.4.3	
	COMMENT	1.6.3.1	
	EXPLODE	1.4.2	A DFD is open
		1.4.3	A child DFD exists for selected process
		1.4.6	Process does not have a child DFD and user confirms execution
	VIEW DD	1.5.1.4	
	DELETE	1.6.1.6.1	User specifies SELECTED OCCURRENCE ONLY
		1.6.1.6.2	User specifies ALL IN DFD HIERARCHY
		1.6.1.6.3	User specifies ALL IN OPEN DFD
		1.6.1.6.4	User specifies ALL OCCURRENCES
	LINK	1.6.3.4	
	UNLINK	1.6.3.5	

Menu	Command	Process	Condition
	MOVE	1.6.4.2	Selected object is a data flow name label
		1.6.4.5	Selected object is a data flow argument label
		1.6.3.3	Selected object is a process
		1.6.2.2	Selected object is an EE
		1.6.2.5	Selected object is a data store
DD ENTRY	VIEW	1.5.1.1	
	NEXT	1.5.1.2	
	PREVIOUS	1.5.1.3	
	CLOSE	1.5.2	
	CLOSE ALL	1.5.3	
	DELETE	1.7.1	
	EDIT	1.7.2	
		1.7.6	
		1.7.7	No entry is currently being edited
	RESTORE	1.7.3	
	SAVE EDITS	1.7.4	
		1.7.5	Edit errors were found
	END EDIT	1.7.4	
		1.7.5	Edit errors were found
		1.7.8	No errors were found
	SCROLL		
DFD REPORTS	PRINTED DFDS	1.2.4	
	DFD STATISTICS	1.2.5	
	UNNAMED ELEMENTS	1.2.3	
	BALANCE DFDS	1.2.6	
	UNDEFINED ELEMENTS	1.2.7	
	DFD DD ENTRIES	1.2.2.1.1	User specifies FULL format
		1.2.2.1.2	User specifies SUMMARY format

Menu	Command	Process	Condition
DD REPORTS	PRINTED DD	1.2.2.3.1	User specifies FULL format
		1.2.2.3.2	User specifies SUMMARY format
	UNUSED ENTRIES	1.2.2.2.1	User specifies FULL format
		1.2.2.2.2	User specifies SUMMARY format
	MAIN & SUB ENTRIES	1.2.2.4.1	User specifies FULL format
		1.2.2.4.2	User specifies SUMMARY format

Appendix VII

Notes on Several MUSSAT Features

Production of the DFD Balancing Report

To determine whether or not a data flow into or out of a parent process is balanced in the child DFD, MUSSAT checks for a data flow with exactly the same name as the data flow into or out of the parent process. If a match by name is not found, then flows into or out of the child DFD are checked to see if a parallel decomposition of data flows has occurred between the parent process and child DFD.

A parallel decomposition is found by attempting to match sub-components of the data flow in question with data flows into or out of the child DFD. For example, if the following data flow definitions existed:

DF1 = [DF1A | DF1B]

DF1A = {DF1A1}

DF1B = [DF1B1 | DF1B2]

and DF1 appeared as a flow into a process and DF1B1 appeared as a net flow into the associated child DFD, as shown in Figure VIII.1.

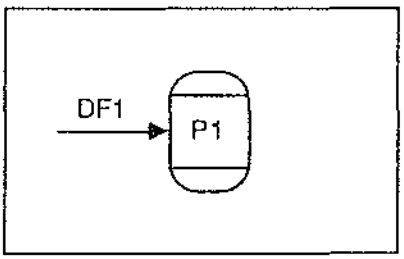


Figure VII.1a - Parent process P1 with net data flow in.

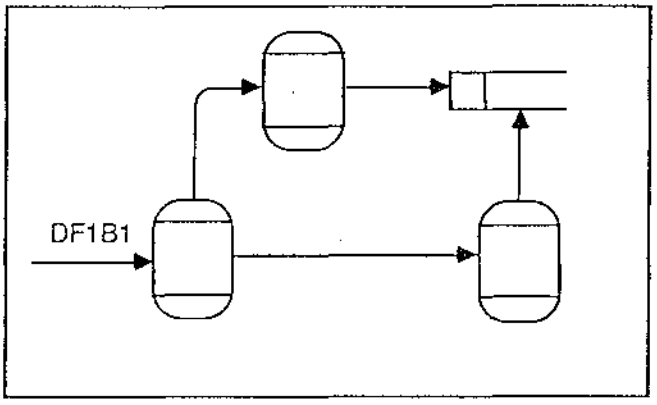


Figure VII.1b - Child DFD of process P1 with decomposed flow in.

MUSSAT would correctly identify that DF1 had been decomposed in the child DFD and that DF1B2 and DF1A should also appear in the DFD. If data flow DF1B2 and DF1A did not appear as net flows into the child DFD, then DF1 would be shown in the MUSSAT Balancing Report as follows:

```
DF1
  *DF1A*
   DF1B
     DF1B1
     *DF1B2*.
```

If DB1B2 also appeared in the child DFD, then the above output would appear as:

```
DF1
  *DF1A*
   DF1B
     DF1B1
     DF1B2.
```

If, instead of DF1B1 and DF1B2, DF1B was shown in the child DFD, then the output would be:

```
DF1
  *DF1A*
   DF1B.
```

That is, net data flows into the parent process are only decomposed if a match with a data flow into a child DFD is found. If DF1B1 consisted of DF1B1x and DF1B1y, and these data flows appeared in the child DFD, and no higher level components of DF1 appeared, then MUSSAT would not be able to identify the parallel decomposition since only two levels of decomposition are checked for.

The choice of two levels of decomposition checking was based on the author's experiences of SSA in which no more that two levels of decomposition were ever required between a parent process and child DFD. It would be ppossible to define a

data flow that required more than two levels of decomposition to be used between parent process and child DFD, however, this author suggests that such data flows are likely to be too complex and should be re-defined so that no more that two levels of decomposition are required.

The output format of the DFD BALANCING REPORT was designed to show the user where exceptions to the MUSSAT balancing rules were detected. The philosophy followed when designing this report was that the user knows best when an unbalanced data flow is an error or when it is an acceptable divergence from the normal DFD constructiion rules. Hence, the DFD BALANCING REPORT does not attempt to show 'errors' of blancing between parent process and child DFD. Rather, exceptions to the norm are noted and the user may then decide what, if any, action to take.

Unused DD Entries

A DD entry is considered to be unused if all of the following conditions are true:

> the DD entry name is not used in the DEFN field of any other DD entry;
>
> the DD entry is not aliased from another DD entry, that is, the DD ENTRY ID does not appear as MAIN ENTRY ID in an instance in the ALIAS ENTRY entity (D16); and
>
> the DD entry does not have any associated data flow, data store, EE or process DFD elements.

Note that a DD entry is not unused if it only apears in the DEFN field of a DD entry that is itself unused. Hence, the PURGE DD command, that removes unused DD entries from the DD, may cause other entries to become unused. DD entries that become unused as the result of the PURGE DD command are not automatically deleted, regardless of the value of DD DELETE. The reasons for this are as follows:

(1) To maintain consistency between commands: the DELETE command in the DD menu only deletes the selected entry as does PURGE DD.

(2) The user could find the results of PURGE DD unpredictable if the entire hierarchy of unused entries was deleted since it is not immediately apparant how many other entries may be affected by the deletion of an unused DD entry.

(3) The user may wish to retain other entries, or inspect any additional entries that become unused as the result of the PURGE DD command.

The UNUSED DD ENTRIES report will print out all DD entries that conform to the rules identified above. Hence, the UNUSED DD ENTRIES report will list all those DD entries that would be deleted if the PURGE DD command were activated.

DATE DFD LAST AMMENDED and DATE PROJECT LAST AMMENDED

The two data fields, DATA DFD LAST AMMENDED and DATE PROJECT LAST AMMENDED, found in the data structures DFD and PROJECT (respectively) are always updated as soon as an operation is performed that affects the model stored in the associated PROJECT DB.

These data values must be updated when the relevant operation is carried out so that the correct date/timestamp is always available for inclusion in the printed reports. All of the MUSSAT reports contain one, or both, of these timestamps in the report title page. Since reports may be requested at almost any stage while using MUSSAT, the date/timestamps recording when the project of DFD was last ammended must be accurate.

Neither of these dates are changed in response to the user changing any of the system parameter values.

The following commands as listed in the menu formats in Appendix IV will cause DATE DFD LAST AMMENDED to be updated:

Menu	Command
DFD	AUTONUMBER
	ERASE
	VERSION
	RENAME
	DFD SIZE
OBJECT	NAME
	NUMBER
	COPY
	(UN)BOLD
	ARGUMENT
	REDRAW
	COMMENT
	DELETE
	MOVE

The commands that may cause DATE PROJECT LAST AMMENDED to be updated include, in addition to the commands listed above, the following:

Menu	Command
PROJECT	RENAME
	PURGE DD
DD ENTRY	DELETE
	SAVE EDITS
	END EDIT
DFD	DELETE
OBJECT	LINK
	UNLINK

The PURGE DD command will cause DATE PROJECT LAST AMMENDED to be updated of one or more DD entries are deleted. Similarly, SAVE EDITS and END EDIT will cause DATE PROJECT LAST AMMENDED to be updated if no edit errors are found and the DD entry is successfull updated.

Each of the associated process entries in the DD include the appropriate statements to update one or both of the data fields. In most cases, these data flows have only been shown in intermediate level DFDs and not in the lowest leaf level DFDs where the processes for the above named commands are shown. Although leaving the updated date flows out of the leaf level DFDs leaves the DFDs unbalanced, it was decided to ommit these data flows so as not the clutter the already complex DFDs.

Process 1.7.6 - EDIT SELECTED DD ENTRY

The detailed function of the editor that allows the user to hange fields in a DD entry has not been defined, other than to note which fields the user may edit and valid values for each of these fields.

The system design presented has concentrated on the more specialised functions required in MUSSAT. For example, the logic associated with producing the DFD BALANCING REPORT has been described in detail, while the text editor has been left unspecified. The functional requirements of a text editer are generally well understood. Suffice to say that

the MUSSAT DD entry editor should use the same interface characteristics as the rest of the system.

Data Model Diagram entity D3 - DD

The entity 'DD' in the Data Model Diagram provides the mechanism that allows the user to view DD entries as they would appear if the DD entries were stored alphabetically by DD entry name regardless of the DD entry type. The data requirements for each of the DD entry types (entities D8-D14) were different enough to worrant a separate entity for each type of DD entry. Since most Database Management Systems do not allow relationships to contain different member types, 'DD' was introduced to allow an alphabetic ordering of all DD entries that the user may view and edit.

The 'DD' entity is not strictly necesary since the alphabetic ordering of all DD entries could be achieved by retrieving the names of all DD entries in entities D8-D14, sorting these names and removing the names of any process DD entries that have child DFDs. Nevertheless, 'DD' was included in the Data Model Diagram because it is an important conceptual entity, even though the entity may be removed when the physical database is designed.

Appendix VIII

Proposed Improvements to MUSSAT

(1) Addition of a command to allow the user to add free-form text blocks to any unnocupied space in a DFD. This would allow the user to show any relevant notes on the DFD.

The user would be able to add, delete and move these text blocks with the DFD.

(2) Allow the user to specify DFD names (and numbers) by selecting the relevant name from a displayed list of the names of all existing DFDs.

For commands for which the user must specify a DFD, the current MUSSAT design requires the user to type the DFD name (or number) into a dialogue box. Use of selection lists could be incorporated with such dialogue boxes, so that if the user does not wish to type in a DFD name, then a list of all existing DFD names could be displayed. The user may then select one of the displayed names.

The use of selection lists is included in Hansen's user engineering principles as outlined at the beginning of this chapter.

(3) Allow the user to change the size of DD windows so that a DD window may be shrunk so that only the window move bar is visible, or expanded so that it fills the entire viewing area.

(4) Allow the user to specify where hyphenation and breaks may occur in a DFD element name if the name must be reformatted to fit inside the DFD element shape. Since only a fixed amount of space will be available within any process, data store or external entity in which to display a name label, it is likely that the user would prefer some form of contro over how the name is displayed.

(5) When the user creates a child DFD for an existing process, a representation of the flows into and out of the parent process could be carried down to the child DFD to assist the user in balancing the child DFD with the parent process. The user could then define net data flows into and out of the child DFD using the flows carried down from the parent process.

As new data flows were added to the child DFD, those data flows that correspond to data flows in the list of carried down data flows would be marked, in the list so that at any point in time the user would have an indication of which flows were balanced between parent proces and child DFD.

This feature is potentially very useful, however, further work is required to define how the carried down list would work when a data flow is subject to parallel decomposition between parent process and child DFD.

(6) An option could be added to the production of all reports, allowing the user to specify whether the report should be printed or displayed on screen. Currently, all reports are printed.

(7) The addition of DD reports based on entry type may prove to be useful. For example, a printed DD report of all process entries.

(8) The Gane & Sarson technique of marking duplicate DFD elements with a unique number of parallel lines has not been included in the MUSSAT specification. Once a version of MUSSAT has been implemented, and the users have had the chance to experiment with the system, they may decide that duplicate DFD elements should be marked.

(9) The users may also decide that password protection should be incorporated into MUSSAT.

Appendix IX

Comments on Using SSA

The first section of this appendix describes the difficulties that were encountered in using SSA to specify MUSSAT. The second section presents some general observations about using SSA to create a system model.

1. Difficulties in Using SSA

(1) The DD (Data Dictionary) is difficult to maintain manually.

There are 22 DFDs in the MUSSAT model and over 150 pages of corresponding DD entries. A DD of this size is hard to maintain manually because no cross-referencing facility is maintained. A cross-referencing facility would allow the analyst to determine the relationships between individual DD entries. Without some sort of cross-reference of DD entry usage, it is difficult to determine the repercussions of changes made to a particular DD entry.

For example, if a data element DD entry named 'A' is renamed 'B', then all DD entries that reference 'A' must be similarly updated. In a purely manual DD it is usually difficult and tedious to find out which entries are affected by a change to another DD entry.

The DD is also hard to maintain because that it quickly becomes large, especially when complex data structures are defined.

(2) It is difficult to reference sub-structures of a complex data store.

DeMarco suggests that databases should be shown as data stores ([DEM79] p. 57), hence the MUSSAT database was shown using one data store. Since a data store may only consist of data structures and data elements, each entity in the MUSSAT database design is represented as a data structure in the data store definition of the database.

Difficulties arose in showing the relationship between data in the database and the contents of data flows connected to the data store representing the database. For example, the entities DATA STORE DD ENTRY and PROCESS DD ENTRY both have the attributes DD ENTRY ID and NOTES. The entities are defined in the database as data structures and the attributes as data element components of each of the data structures. If a data flow connected to the data store representing the MUSSAT database was defined as containing two data elements: DD ENTRY ID and NOTES then it would be unclear as to which logical database entity these data elements were associated with.

The problem was overcome by showing the logical entity numbers (from the MUSSAT Data Model Diagrams) that the contents of a data flow are related to in the NOTES section of the associated data flow DD entry.

(3) The methodology does not differentiate between types of data store accesses.

Neither DeMarco nor Gane & Sarson give any guidelines as to whether, or how, different types of data store access should be shown. A data flow out of a data store will contain data retrieved from the data store ([DEM] pp. 52-54 and [GAN80] pp.39-40), however, it is this writer's experience that a data flow flowing into a data store may represent one of three types of access:

adding new information to the data store;
updating existing information in the data store; or
deleting existing information from the data store.

It is this author's experience that a DFD is easier to understand if the purpose of a data flow into a data store is apparent from the DFD. Hence, a data flow naming convention was adopted for all data flows into a data store, to indicate the role of data flow. Details of the naming convention are included in Appendix V.

It is also unclear whether or not deletion of existing information from a data store should be shown as a data flow.

(4) There are no guidelines supplied for using Structured English to show how the mapping between data flows into a process and the data flows out of a process should be represented.

When a process transforms data that exists as physical objects in the real world then the transformation is usually straightforward. For example, a process called PRODUCE INVOICE which accepts details of a filled order and produces an invoice. In this case it is clear from the function of the process that the order details are transformed into the invoice.

The transformation is not so straight-forward when a process has more than one input and output data flows, especially when the data flows do not represent physical objects. The MUSSAT system model abounds with such processes. The convention developed to show the relationship between input and output data flows in the MUSSAT system model is discussed in Appendix VI.

2. General comments on using SSA

(1) DeMarco's seven plus or minus two guideline for the number of processes on a DFD is a useful guideline for drawing

DFDs, however, the number of data flows connected to a process seems to be just as important. A DFD with between five and nine processes can still be difficult to understand if the processes have too many connected data flows.

DeMarco recognises the need to limit the of the number of data flows connected to process, but does not offer any advice other than "Less is better." ([DEM79] p. 113).

This author suggests that the maximum number of data flows connected to any one process should be seven (plus or minus two). There are several reasons why this is a useful rule of thumb:

> As DeMarco suggests, the human mind is best at dealing with sets of seven or less items ([DEM79] p. 82).
>
> The more data flows that are connected to a process, the more likely that one or more will intersect with another data flow in order to reach the process; intersecting data flows make a DFD more difficlt to read.
>
> The more data flows that are connected to a process, the more likely that additional duplicates of data stores and external entities will be required to help minismise the number of intersecting data flows and hence cause the DFD to become larger.

A large number of data flows connected to a process may indicate that summary data flows should be used with a parallel decomposition shown in lower level DFDs.

The minimum number of data flows that should be connected to any one process is one.

(2) It is useful to show, within a DFD, which processes have been exploded into child DFDs, rather than searching through the DD for a process definition; if a process definition does not exist in the DD for a process, then the assumption is that the process has been exploded into a child DFD.

In the MUSSAT system model, a "*" was placed in the process bubble of those processes that had child DFDs.

(3) It is difficult to produce easy to read DFDs with good partitioning. The MUSSAT system model presented in the Appendices is the result of many months of work.

DeMarco suggests that the author of a DFD should be willing to rework DFDs, possibly several times over, in order to produce the final version ([DEM78] p. 69). Such reworking can include repartitioning ([DEM78] p. 114). DeMarco gives some guidelines on how to repartition a set of levelled DFDs, and when repartitioning may be necessary ([DEM78] p. 113-114). Nevertheless, good partitioning is hard to achieve.

(4) The most useful error detection technique during the design of the MUSSAT model was to simulate each of the MUSSAT commands, following the transaction through the DFDs, and ensuring that all the functions necessary to execute the command were defined.