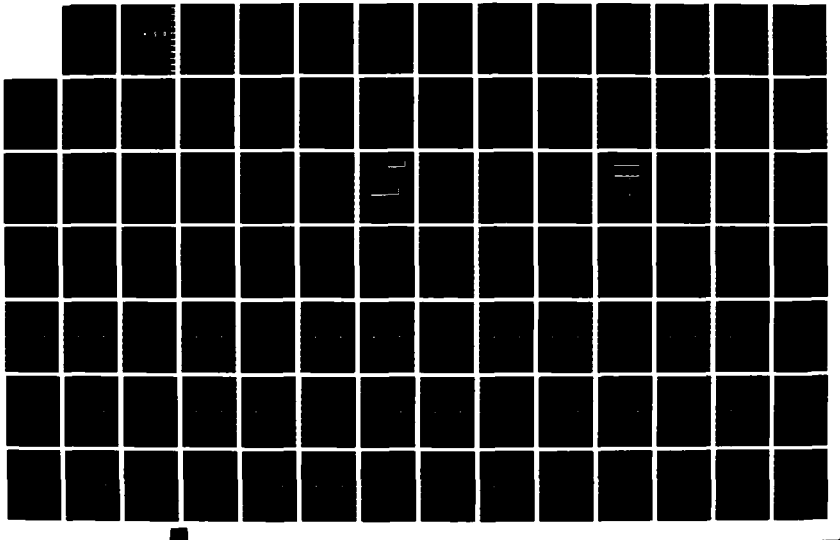
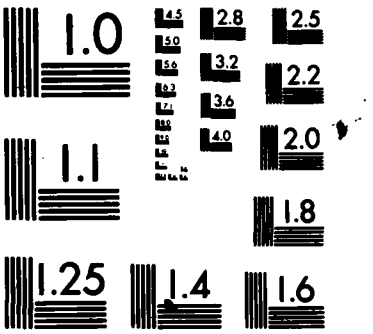
AD-A171 393 DESIGN OF A GRAPHICS USER INTERFACE FOR A DATABASE MANAGEMENT SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J K ADCOCK JUN 86 1/2

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A171 393

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
SEP 05 1986
S D
D

THESIS

DESIGN OF A GRAPHICS USER INTERFACE
FOR A DATABASE MANAGEMENT SYSTEM

by

Jerry K. Adcock

June 1986

Thesis Advisor : C. T. Wu

Approved for public release; distribution is unlimited.

DTIC FILE COPY

86 9 5 008

AD-A171 393

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) UNCLASSIFIED
Design of a Graphics User Interface for a Database Management System

12. PERSONAL AUTHOR(S)
Jerry K. Adcock

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 June 20	168

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	GLAD, Graphics User Interface, Graphics Language, for Accessing a Database, Hierarchical Input Process Output, HIPO

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis presents a solution to the problems associated with database management systems. User needs are discussed, with a methodology to meet those needs. It is shown that no current system exists which can satisfy all requirements, so a new system or interface must emerge.
The remainder of the thesis presents the design of such a system, called Graphics Language for Accessing a Database (GLAD). The Hierarchical Input Process Output (HIPO) System is used for design representation.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. C. T. Wu	22b. TELEPHONE (Include Area Code) 408 646-3391
	22c. OFFICE SYMBOL 52Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Design of a Graphics User Interface for a Database Management System

by

Jerry K. Adcock
Lieutenant, United States Navy
B.S.E., Purdue University, 1978

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1986

Author: _____
Jerry K. Adcock

Approved by: _____
C. T. Wu, Thesis Advisor

_____
Michael Zyda, Second Reader

_____
Vincent Y. Lum, Chairman,
Department of Computer Science

_____
Kneale T. Marshall,
Dean of Information and Policy Sciences

ABSTRACT

This thesis presents a solution to the problems associated with database management systems. User needs are discussed, with a methodology to meet those needs. It is shown that no current system exists wich can satisfy all requirements, so a new system or interface must emerge.

The remaineder of the thesis presents the design of such a system, called Graphics Language for Accessing a Database (GLAD). The Hierarchical Input Process Output (HIPO) System is used for design representation.

Accesion For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION 6

A. BACKGROUND 6

B. MOTIVATION 7

C. POSSIBILITIES 10

D. EARLY WORKS 13

E. NEW SYSTEM PROPOSAL 15

F. THESIS ORGANIZATION 17

II. SPECIFICATIONS 19

A. INTRODUCTION 19

B. ADVANTAGES OF GLAD 19

C. BASIC PROGRAM COMPONENT DESCRIPTIONS 24

D. EXCEPTION HANDLING 38

E. PRIORITIES IN DESIGN 39

F. DESIGN HINTS AND GUIDELINES 40

III. DESIGN 42

A. METHODOLOGY 42

B. DESIGN CONSIDERATIONS 43

C. DESIGN DECISIONS 44

D. DESIGN LAYOUT 45

IV. CONCLUSIONS AND RECOMMENDATIONS 46

A. CONCLUSIONS 46

B. RECOMMENDATIONS 47

APPENDIX 52

LIST OF REFERENCES..... 164

BIBLIOGRAPHY..... 166

INITIAL DISTRIBUTION LIST..... 167

I. INTRODUCTION

A. BACKGROUND

Electronic assistance to office workers and the resulting productivity increases have risen dramatically in the past two decades. Even the early 1970's products, such as the IBM magnetic card typewriters, enabled workers to vastly improve both the quantity and quality of their efforts. The largest gains, however, resulted from the introduction of affordable microcomputers. Microcomputers in the office environment provided seemingly endless possibilities, such as word processing, data processing, information storage, message/letter routing via networks, facility for automated computations, integration of multiple office functions, graphical displays, and numerous others. In fact, office workers discovered that one of the marvels of microcomputers is that every use of a micro prompts a "wouldn't-it-be-nice-if" for other users.

The challenge to meet these "enhanced capability" desires was met with gusto by computer programmers--there are now thousands of applications programs in the marketplace to meet almost any need. However, since they were written in response to specific needs and desires, almost all of these programs share a common deficiency: they are too specific to be generally useful. Sophisticated word processors

cannot handle spreadsheet applications. Simple, menu-driven programs are annoyingly tedious to experienced users. Integrated packages lack sophistication in all areas.

B. MOTIVATION

One area that provides good example of the wide range of capabilities of applications-based programs (and the motivation for this thesis) is that of database management systems. They cover the spectrum from very easy to use (but very limited in complex querying capability) to very powerful (but too difficult for the novice user). While any one of these programs might be ideal in a particular situation, any change in the environment necessitates a change of systems or an extensive training program. These changes might be prohibitively expensive in terms of time, money, or both, and should not be necessary. "Wouldn't-it-be-nice-if" there was a system that was capable of satisfying both needs? There will be.

Before attempting to design such a system, we must address two major areas of consideration: 1) the needs of the user, and 2) the requirements of the program to meet those needs. These areas will be addressed directly here, and indirectly throughout the remainder of this thesis.

1. The Needs of the User.

General requirements for office workers are covered expressly in (LAR 84), and less formal approaches in (WON 82), (WU 85), and (ZLO 77) also address user needs. Following is a consolidation of the ideas expressed in those articles.

a. Information must be presented in the user's view. It must be presented in a form that is natural and familiar to the user, or he will never be comfortable with the system--it will always remain somewhat magical (and not to be trusted completely).

b. Memorization requirements must be minimized. Most keywords and interaction procedures are arbitarily assigned, and requiring the user to memorize them introduces a new abstraction that is not natural for him to accept. This concept should be applied to several areas:

(1) Database. (LAR 84) states "the office worker should not need to remember the logical structure of the database...the system should display this information";

(2) Query language. Whether the language is composed of words, pictures, or some combination of those, the user should be able to quickly gain an intuition about the meaning of the symbols;

(3) Query formulation. The formulation process should coincide with the user's thought process, so the program must include the capability for the user to formulate a query in a piecemeal fashion.

c. Training time must be minimized. While there must be some training for any non-trivial program, an excessive training time requirement will exclude some users (who simply cannot invest that amount of time) and will discourage those who do attempt the training program.

d. The possibility of erroneous input must be minimized. There are two types of errors which can be easily detected (and therefore avoided): 1) an inappropriate command (i.e. a normally valid command at an inappropriate

time in program execution), and 2) a mistyped command (i.e. improper spelling or invalid format). Taking action to avoid these two errors will not assure the user a mistake-free session with the program, but it will go far to increase the user's confidence by eliminating major problems caused by trivial errors.

e. Feedback must be provided. A good example of this (without examining a specific program) is the capability to display intermediate results during the query process. While it is not desirable to overburdent the user with informatin (don't make this display mandatory), he should be able to access it if desired to verify the correctness of his query. Another benefit of feedback is that it encourages experimentation--it answers the "what-would-happen-if-I-asked-this" question.

f. Help must be provided. User help can take on many forms, such as menus, subject directories, help messages, error messages, structure displays, screen layout, intermediate result displays, input prompts, and many others. It is the responsibility of the designer to provide sufficient help to the novice user without forcing unnecessary help onto the sophisticated user (WU 85).

g. It must be capable. Though all other goals may make the user comfortable and confident, they are all for naught if the user cannot extract the required information. He must be able to perform a wide range of activities,

from simple data/structure viewing to the formulation of complex queries.

2. The Requirements of a Program to Meet User Needs.

The characteristics of such a program (or, more specifically, a database interface package) are addressed in (WU 85). Those characteristics are reiterated here, along with explanatory comments and the user requirements (described above) that they satisfy.

a. It must be descriptive. This includes both the kinds of data stored and their relationships. This characteristic satisfies requirements B.1)(a), B.1)(b), B.1)(c), and B.1)(f).

b. It must be powerful. If the information is contained in the database, the user must be able to extract it, regardless of the complexity of the query. This characteristic satisfies requirement B.1)(g).

c. It must be easy to learn. Many users of the program will be novices unfamiliar with database terminology, and the designer is challenged to produce a program which can be quickly learned (interactive tutorials are often helpful in this process). Designing a program with this characteristic will necessarily satisfy requirements B.1(a) B.1(b), B.1(d), B.1(e), B.1(f) and B.1(g).

C. POSSIBILITIES

We now have some solid requirements to begin a program design. However, there still remain several questions that

must be answered. These questions present themselves in a sequential nature, so let us now address them in that manner and explore the possibilities.

1. Has the Problem Already Been Solved?

The answer is "no". While there are many database programs available (some of which will be discussed in Section D), none of them satisfy all the requirements for the user and the program. The biggest trade-off in existing programs is the ease of use as opposed to power.

2. Is an Entire New Program Required?

Probably. While it might be possible to modify existing programs to eliminate some of the disadvantages and meet more of the requirements, there would inevitably remain some inappropriate characteristics which are either impossible or not cost effective to remove. It would be much better to incorporate the positive characteristics of many such programs into a new one while avoiding the negatives.

3. Will an Interface Program Meet Our Needs?

Yes. The only caution here is to ensure that the underlying program/query language is capable. A powerful program can be made easy to use: an inherently weak program cannot be made powerful without extensive, fundamental changes.

4. What Type Of Interface Do We Want To Use?

As discussed in (WU 86), there are three ways the user interacts with a database management system: the

creation of the database, the manipulation of data, and the development of applications programs. He goes on to discuss several existing interface methods, such as natural language interface (BOG 84, COD 74, HEN 77, WAL 78), modified query language interface (KOR 84, MAC 85), graphics interface (HER 80, LAR 84, MCD 74, STO 82, WON 84, ZLO 77), fourth generation languages, and fill-in-the-form programming (ROW 85). None of these existing interfaces, however, address all three types of user interaction. What is needed is a single, unified interface which will enable the user to accomplish all his activities within one environment. The interface method which presents the greatest potential for this is the graphics interface.

5. Why a Graphics Interface?

(RAE 85) presents an excellent discussion of the advantages of using graphics in programming, and those points can be directly related to program users. Some of those advantages are: the random-access nature of text, the increased dimension of expressions one can achieve with pictures, the higher rate of knowledge transfer through pictures, and the increased ability to represent the real world through pictures. While it would not be universally useful to devise an interface which presents only pictures, some combination of pictures and text layed out in a graphical representation would achieve the same advantages.

D. EARLY WORKS

The potential for a userful graphics interface has long been recognized, as evidenced by the number of graphics interfaces developed over the past ten years. Following is a brief review of four of these interfaces, using the criteria in (WU 85) (described in Section B.2) to judge their effectiveness.

1. Query-by-Example (QBE).

QBE (ZLO 77) was one of the first DBMS graphics interfaces. Its philosophy was to minimize the requirements (of initial knowledge and memorization) imposed on the user. QBE is relationally complete, so users can formulate any query that can be expressed in relational algebra or predicate calculus; however, "skeletons" (templates) are provided for query formulation to alleviate the need for the user to know first order predicate calculus. The major problem with QBE and similar approaches, such as those reported in (LAR 84) and (SUG 84), is that they lack descriptiveness. All input and output in QBE is in tabular form, so it is difficult for the user to get a good overview of the system and relationships of the data, and it is not possible for him to easily browse through the data or database schema.

2. Spatial Data Management System (SDMS).

SDMS (HER 80) is a good example of a program that is easy to use, but has limited capability. Data are represented in graphical form, and their relationships are

determined by their spatial positions in a graphical data space. The system was written for novice users, and seems to encourage browsing with its "zoom", "unzoom", and "position cursor" commands. Simple data retrieval is relatively easy with SDMS, but it lacks a simple method to formulate a complex query. Therefore, the poser of SDMS is not accessible to many users.

3. Text, Icon, and Map Browser for Extended Relations (TIMBER).

TIMBER (STO 82) is described by its author as "a user friendly, graphics-oriented browser for a relational data base". It provides the same type of browse capabilities as SDMS, enhanced by incorporating some of the concepts of QBE. Its ability to support icons, maps, text, and normal fixed format relations is an improvement over SDMS, but it still lacks power and descriptiveness.

4. Graphical User Interface for Database Exploration (GUIDE).

GUIDE (WON 82) is the first graphics interface package that attempts to address all the requirements described ablve. It is descriptive in that it displays the database schema as a network of entity and relationship types. It also provides hierarchical subject directories to further describe database contents and assist in data location. It allows for piecemeal query formulation and gives feedback by displaying intermediate results, making

complex queries possible. However, some aspects of GUIDE still hinder its effective use. These include the lack of relation browsing capability, the lack of aggregate functions, and the use of two different types of diagrams (Entity/ Relationship and hierarchical subject directories) during program execution. These disadvantages can be major hinderances to the novice user.

E. NEW SYSTEM PROPOSAL

As we have seen, each of the previous works in graphics interfaces addresses one or more of the requirements for an effective system. However, none of them satisfactorily provides solutions to all requirements. It is the purpose of this thesis, as the initial step in the production of a useful graphics interface, to introduce and design a system to address all requirements of both the user and the program.

Basic descriptions of such a system are presented in (WU 85) and (WU 86). The system is called Graphics Language for Accessing Database (GLAD). Its intent is to provide a complete, effective interface devoid of the previously discussed disadvantages by incorporating the positive characteristics of several earlier works into a single, unified interface package. A complete description of GLAD characteristics is given in Chapter 2, but let me now briefly address how GLAD will satisfy the four requirements for an interface presented in (WU 85).

1. Descriptiveness.

GLAD will employ a diagrammatic display of the database schema. This diagram will be rich in meaning because it gives a graphical representation of both the entities involved and their relationships, and is applicable to all data models. In addition, help facilities will be included to describe object names and formats to the user.

2. Power.

GLAD's querying capability will be based on the concepts of QBE. As previously discussed, QBE is relationally complete, so GLAD will inherit from QBE the ability to formulate any query which can be expressed in relational algebra or predicate calculus.

3. Ease of Learning.

GLAD will be easy to learn because it employs the use of common, easily recognizable symbols to identify objects. These symbols are circles, dotted and solid lines, and regular, nested and repeated rectables. They will be arranged in a natural, sensible manner with the complexity of the arrangement corresponding to the complexity of the objects/relationships. In addition, many help facilities will be available to (but not imposed upon) the user.

4. Ease of Use.

GLAD will have many features making it easy to use. One of these is a convenient browsing facility with its own submenu to allow the user easy access to any area

of the database and its information. Another is its mechanism for query formulation. The user can formulate queries in piecemeal fashion, access intermediate results, and combine results in any manner convenient to him. Finally, help facilities should quickly alleviate any stumbling block the user might encounter.

F. THESIS ORGANIZATION

The remainder of this thesis will be organized as described below.

1. Chapter 2: Specifications.

This chapter will include informal specifications for GLAD, including characteristics, program components, menu lay-outs, use of the mouse, design priorities, and design hints and guidelines.

2. Chapter 3: Design.

This chapter will be divided into two major sections--architectural design and detailed design. The architectural design section will decompose the program functions into "black boxes", with emphasis on relationships and interfaces. The detailed design section will be a repeated stepwise refinement of those functions until they are reduced to a sufficiently low level to be readily implemented in a programming language (in this case the C Programming Language).

3. Chapter 4: Conclusions and Recommendations.

This chapter will provide the conclusions of the author as a result of the design of this program, along with

some recommendations to aid/guide further work. These recommendations will address the required steps to complete the production of this system, and some avenues of further research which present themselves during this current work.

Finally, sections will be included for the list of references and bibliography.

II. SPECIFICATIONS

A. INTRODUCTION

This chapter describes GLAD as it is to be designed, including its screen presentation, possible user actions, data representations, and other characteristics of the program. It does not provide requirements in the strict sense (FAI 85), because the specifications are not (nor are they intended to be) complete. Formal specifications for a software project generally provide a means to ensure the designers, testers, and customers are in agreement about the use and action of the program, provide a basis for product validation, and provide a document to assist maintenance personnel. In this project, however, there is only the design team and the motivation for writing "specifications" is to solidify the concepts and general operational characteristics of the program in our own minds. As such, this chapter might be more appropriately titled "GOALS for GLAD". Areas to be addressed include advantages of GLAD, basic program component descriptions, exception handling, priorities in design, and design hints and guidelines.

B. ADVANTAGES OF GLAD

The motivation behind designing this new database management program interface stems directly from the disadvantages found in earlier works in this field. Several such works

were discussed in Chapter 1, so we will confine our discussions here to the advantages we will achieve with GLAD.

1. Ease of Use.

In today's environment, many database management programs are used by those who are either unfamiliar with computers in general or are not experienced in database terminology. This necessitates a program which can show the user what he needs to know and what he needs to do without requiring him to invest excessive time to learn database terminology or a query language. At the same time, ease of use implies that a user should be able to formulate complex queries in a simple, straightforward manner. GLAD will accomplish ease of use by:

(a) Presenting the database schema in a graphical form, as shown in Figure 2.1. This will provide the user an immediate overview of the database and its relationships, which will benefit the novice by giving him an intuitive feel for the relationships in the database, and the experienced user by showing all top-level aggregates and their associations. It also serves as a data dictionary, since the user can easily examine the format or contents of an object.

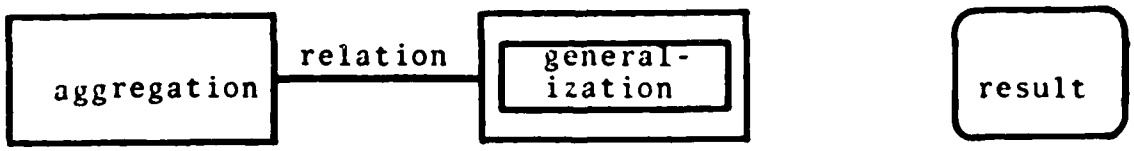


Figure 2.1 Graphical Representation of Data

(b) Encouraging relation browsing. The graphical presentation of the data and the simple use of the three-button mouse make it very easy for any user to browse any object or relation in the database.

(c) Using similar techniques for all user actions. The user will be able to press the same buttons on the mouse for similar functions, whether he is viewing the database or formulating queries. For example, the center button will be used for displaying a pop-up menu and selecting an action both to query the database and to print the results of the query.

(d) Making it simple to change levels of abstraction. By a single press on a mouse button, the user can change from the display of all aggregates to showing the attributes (sub-objects) or members of a single object.

2. Power.

The most essential element in any database program is the ability to extract the required information from it. GLAD will use graphics to assist the user in query formulation and will pattern its querying capability after QBE (ZLO 77), which will ensure it is capable of any query which can be expressed in relational algebra or predicate calculus.

3. Descriptiveness.

This entails both displaying all data and their relationships and assisting the user by describing his options for actions and their consequences. GLAD will achieve these by providing:

(a) Aggregation. This is a grouping of objects or sub-objects. The user will see aggregates as object names enclosed in rectangles, and can view the sub-objects by EXPANDing the object (see Figure 2.2).

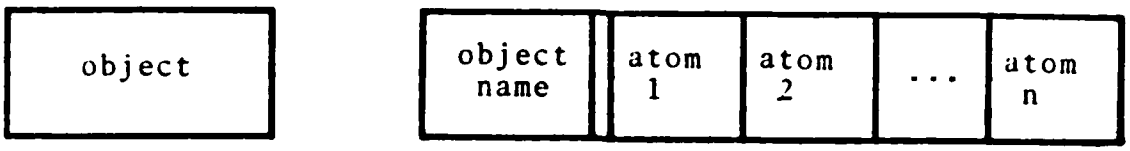


Figure 2.2 Object Representation

(b) Generalization. This is a grouping of objects by their general category. The user will see generalization names enclosed in double (nested) rectangles, and can view the individual objects by EXPANDing the generalization.

(c) Classification. Each item in the database will be described (classified) as a piece of information about an object. These data items will be defined as members of an object and can be viewed by selecting the LIST MEMBER command (from the pop-up menu on the mouse or the menu along the top border).

(d) Relationships. Relationships between objects in GLAD are broken down into four categories (see Figure 2.3):

(1) Relation. This is a basic association between two objects as defined by the user (i.e. identical object or sub-object names that provide a link between the objects). A relation is indicated by a solid line connecting the two objects (see Figure 2.3a).

(2) Partial relation. This is a relation where, given two objects A and B, 1) a sub-object of A is related to B, 2) a sub-object of B is related to A, or 3) a sub-object of A is related to a sub-object of B. A partial relation is indicated by a dotted line connecting the objects (see Figure 2.3b).

Figure 2.3a Relation

Figure 2.3b Partial Relation

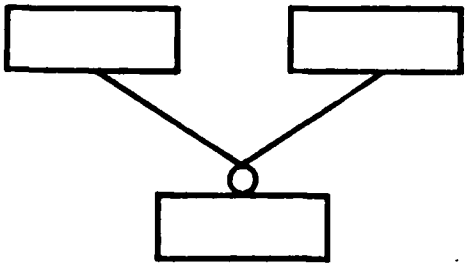


Figure 2.3c Disjunctive Relation

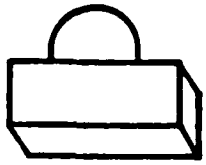


Figure 2.3d Recursive Relation

Figure 2.3 Relation Representation

(3) Disjunctive Relation. This is a relationship between one object and one of multiple other objects. A disjunctive relation is indicated by a solid line connecting the first object to each of the other objects with a circle at the end of the lines terminating at the first object (see Figure 2.3c).

(4) Recursive Relation. This is a situation in which two specialized objects within the same generalized object are related to each other. Recursive relations are represented by a repeated rectangle (see Figure 2.3d) and each association in the recursive relation is indicated by a semi-circular solid line beginning and ending at the object.

(e) Help. On each menu level, there will be provided HELP selection which will describe the actions the user may take at that point and the consequences resulting from each.

4. Ease of Learning.

This generally follows from (1), with the addition that the program must provide sufficient assistance so that a novice can quickly acquire the ability to command those actions necessary for him to extract his required information. This is provided by the graphical display of the data, the permanent menu displayed along the top border, the use of the mouse, and easy-to-access help.

C. BASIC PROGRAM COMPONENT DESCRIPTIONS

This section will provide definitions of terms, descriptions and lay-out of menus, screen display characteristics, program flow, and use of the mouse. It is not intended to be intractible, but will provide guidelines for the design phase.

1. Definition of Terms.

a. Object--any single piece of information or collection of information which is intended to be recognized as a single entity.

b. Atomic Object--a single piece of information which represents only one system-or user-defined base object (string, number, enumeration, subrange, and boolean).

c. Agregate object--a specific collection of one or more (sub)objects.

d. Generalized Object--a collection of objects grouped together by a common category or subject.

e. Specialized object--any one of the grouping in a generalized object.

2. Descriptions and Lay-outs of Menus.

GLAD will provide menus to cover all user actions. They will be heirarchically structured, and will be accessed in two ways: by selecting one of the items listed on the menu along the top border and by pressing the center button on the mouse and selecting one of the items on the menu which pops up. There are three main functional menus in GLAD: the Administration Menu, the Browse Menu, and the Execution Menu. The Administration Menu will be displayed when the program starts, the Browse Menu is accessed by selecting EXPLORE, and the Execution Menu is accessed by selecting QUERY. In addition, several items on these menus will have their own short sub-menus, which will appear below them (and can be accessed by the mouse when appropriate. Only the current main menu will be displayed, and others will appear when an appropriate selection is made on the current level. Menu items are selected by pointing to them with the mouse pointed and pressing the right (select) button, or by displaying the pop-up menu by pressing the center mouse button and keeping it depressed while you roll the mouse down until the correct choice is highlighted and releasing it. Menu items are de-selected (i.e. results are erased and screen is rewritten by re-selecting an already active (selected) item. All prompts to the user will be

placed on a Command Line located immediately below the top border menu.

a. Administration Menu (see Figure 2.4). This menu will be displayed immediately when the program starts, and provides the user with his initial choices for manipulating the database. Menu items include:

(1) OPEN--this option opens the database to be used by prompting the user to enter the database name. When it receives the name, it causes the initial screen display showing object rectangles, names, and relationship connecting lines.

(2) CLOSE--this option closes the database by ensuring all files are closed. The user is queried to save or abandon any open files. This can be done at any time during the execution of the program, providing the user the ability to ensure all prior work is complete before continuing or quitting.

(3) EXPLORE--this option allows the user to view and query the database. While no database manipulation is performed as a direct result of choosing EXPLORE, the menu is changed to the Browse Menu (described later), and the level of abstraction is changed so that data manipulation can be done.

(4) SET-UP--this option allows the user to alter default values in the program. For example, he can instruct the program to SHOW RESULTs each time they are created, or he can have the results DESCRIBEd automatically.

(5) HELP--this option provides help with all Administration Menu items. It describes each of the menu items, actions which will be performed when selected, and associated sub-menus.

(6) QUIT--this option exits the database program and returns the system's prompt. If there are any open files, it provides a prompt to the user to CLOSE first or QUIT abandoning open files.

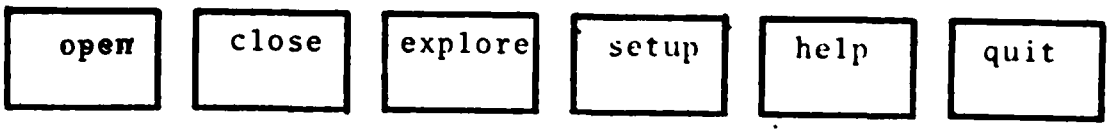


Figure 2.4 Administration Menu

b. Browse Menu (accessed by selecting EXPLORE on the Administration Menu) (see Figure 2.5). This menu provides options to the user to manipulate the database on the "aggregate object" level. Items provided in this menu are:

(1) DESCRIBE--this option provides the user with a definition-type description of an object. The object and all sub-object names are displayed with their associated data types, and all relations linking this object with others are listed. If the object is a generalized object, its associated specific objects are displayed with their data types.

(2) QUERY--this option is similar to the EXPLORE menu item in that no direct data manipulation is performed when it is selected. It does, however, change the level of abstraction and displays the Execution Menu to allow the user to query the database.

(3) UPDATE--this option allows the user to add to or change the information in an object. It provides an object skeleton (with current values if the mouse pointing to an object) and a sub-menu to prompt the user for pertinent information.

(4) PRINT--this option allows the user to print portions of the database or results of a query. In addition, there is provided a means to "screen dump" to allow the user to make a hardcopy of the graphical representation of any screen display during the execution of the program. These options are provided in PRINT's sub-menu.

(5) EXPAND--this option is similar to DESCRIBE except that it is intended to be used in a "browse mode", and just shows the atomic sub-objects of the selected object. It is important to note here that non-atomic sub-objects are not displayed: they are the items which link the object to other objects and are not important to the object definition. These are only displayed during the UPDATE operation.

(6) LIST MEMBER--this option is used to display an instantiation of an object or sub-object, listing current information contained in it.

(7) CENTER--this option will center the display around the object pointed to by the mouse pointer. If no object is being pointed to, the user is prompted for the object name around which to center the display.

(8) HELP--this option is identical to the HELP option in the Administration Menu, except that it covers items in the Browse Menu.

(9) QUIT--this option returns the user to the Administration Menu. If there are any open files or queries, it will prompt the user to complete the action before the return is executed.

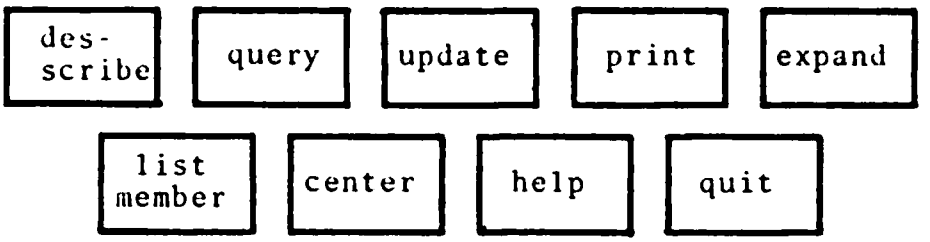


Figure 2.5 Browse Menu

c. Execution Menu (accessed by selecting QUERY on the Browse Menu) (see Figure 2.6). The functions described in this menu demonstrates the real strength of GLAD. They allow the user to specify objects and sub-objects, create results of queries, and display, print, combine, and save those results. The Execution Menu items are:

(1) SAVE RESULT--this option provides to the user a permanent record of process used to achieve his result. This is done by saving to disk a file (user is prompted for filename) containing the process. This process can be called later by the user by the use of a "program box" (which will be designed and implemented later).

(2) SHOW RESULT--this option is identical to the LIST MEMBER command, except that it acts on the result pointed to by the mouse pointer. By requiring the mouse to point to the result, the user can review any of several results he has created. It should be noted here that SHOW RESULT causes the actual result formulation: before this time, only the process for result formulation is saved.

(3) CLEAR RESULT--this option simply erases the process which creates the result pointed to by the mouse pointer. This enables the user to "take back" erroneous query formulations, and eliminate old results before proceeding with new queries.

(4) CREATE RESULT--this option tells the program that all specifications have been made and the user is ready to form the result. The result formulation is not actually performed at this time (since it is time-consuming and may not ever be required), but the process is saved to be used if needed. When the process has been saved, an icon (rounded-corner rectangle) is placed at the bottom of the screen labeled RESULT X, and all rectangles associated with the query are identically shaded (see Figure 2.7).

(5) COPY RESULT--this option makes an identical copy of the result process pointed to by the mouse pointer. It is placed adjacent to the copied result at the bottom of the screen and is labeled RESULT X COPY. By performing the COPY RESULT, the user is able to experiment with several combinations of results without destroying the original contents of the individual result processes.

(6) COMBINE RESULT--this option performs a join of two or more intermediate results. If there are only two results present, the join is done immediately. If there are more than two, the user is prompted for the names of the results to be joined. When the join is completed, the individual result processes are erased and a new result process is placed at the bottom of the screen with all associated rectangles (from the previous results) identically shaded (see Figure 2.8).

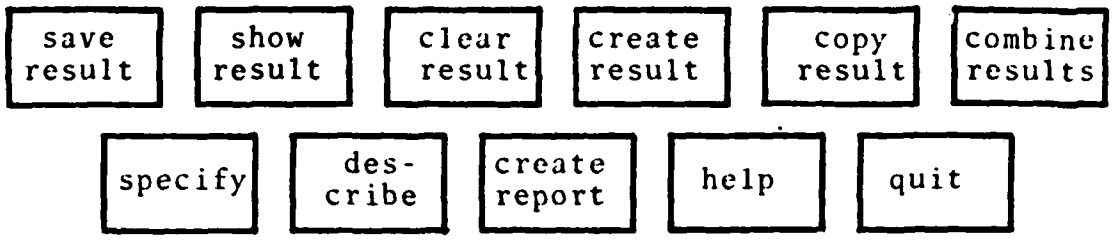


Figure 2.6 Execution Menu.

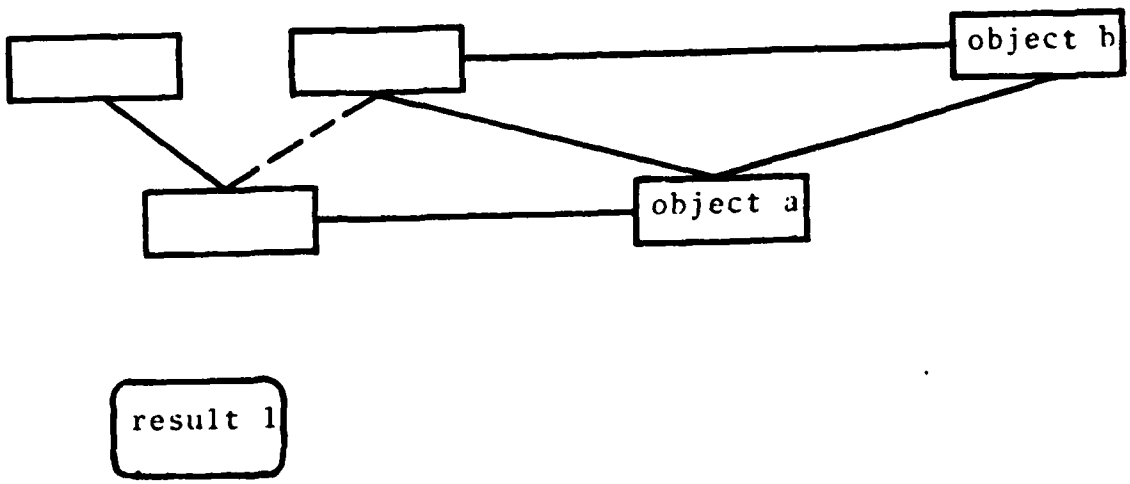


Figure 2.7 Create Result.

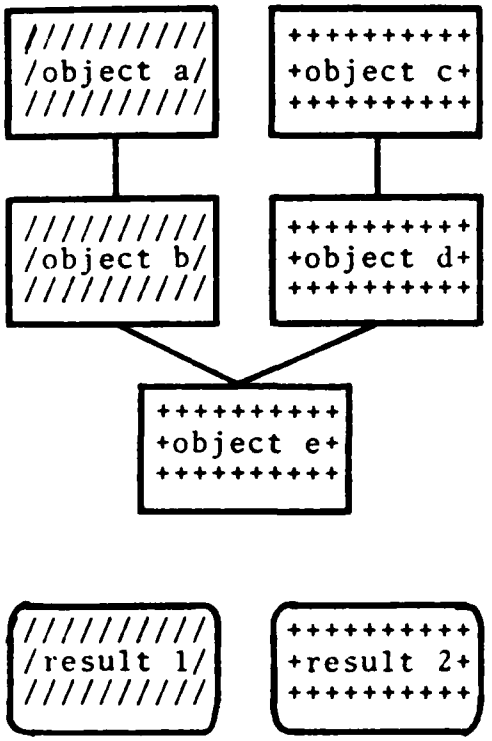


Figure 2.8a Before Combine

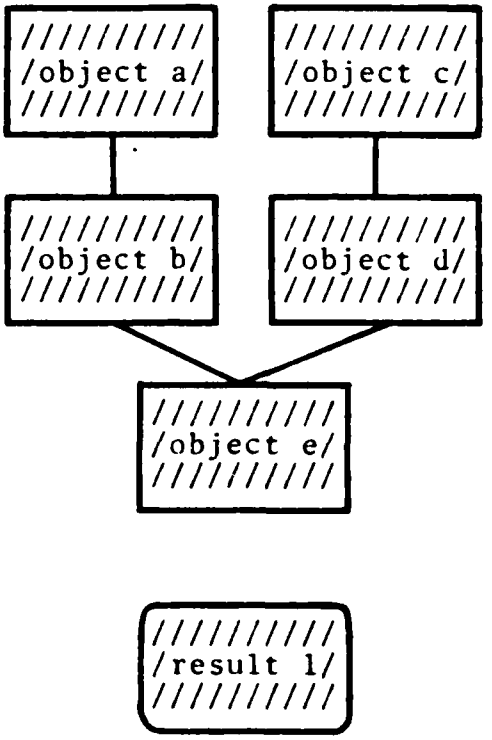


Figure 2.8b After Combine

Figure 2.8 Combine Result

(7) SPECIFY--this option creates an "object skeleton" for the object pointed to by the mouse pointer. This skeleton contains the name of the object and all associated fields in table-name format, and an area below these for query specification (see Figure 2.9).

(8) DESCRIBE--this option provides a description of the result pointed to by the mouse pointer. Included are the name of the queried object and the specifications entered by the user. This description is placed immediately below the result. It remains on the screen until the result is CLEARed or COMBINEd (see Figure 2.10).

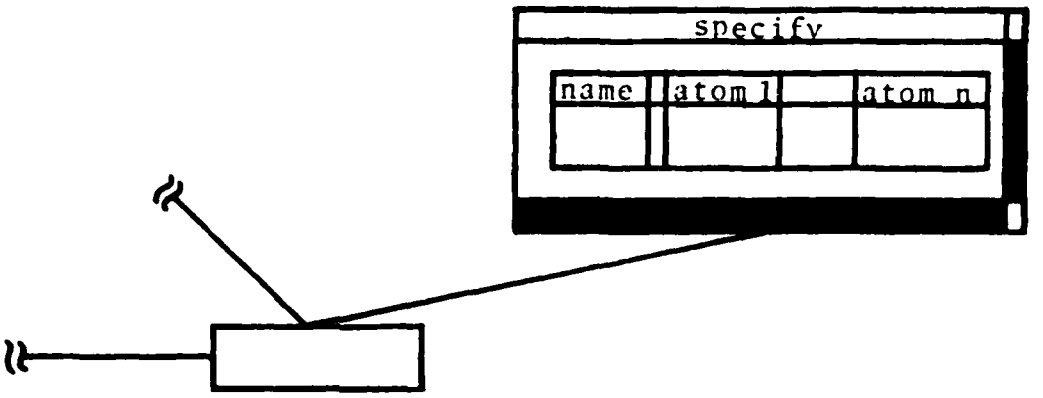


Figure 2.9 Specify Skeleton.

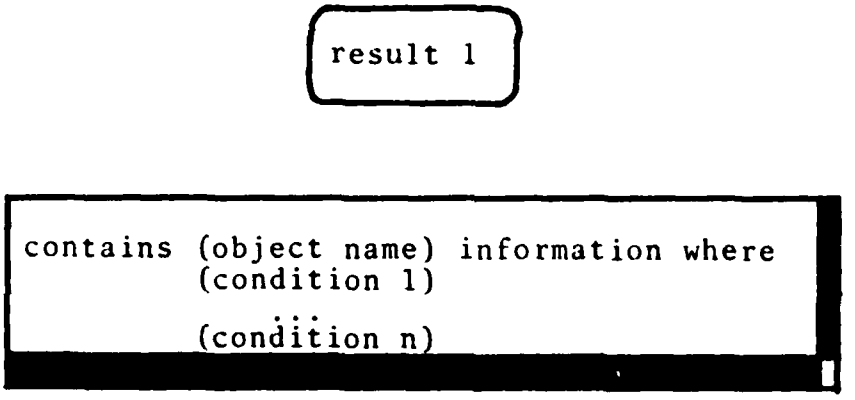


Figure 2.10 Describe Result.

(9) CREATE REPORT FORM--this option allows the user to format a report to be printed (and will apply to both objects and results). He is given a skeleton in which he must provide the form name, report title, and field names, data types, and sizes. The field names are especially important because when the report form is used, type-checking will be accomplished to ensure those names match the sub-object names in the object chosen to be printed.

(10) HELP--this option is identical to the former HELP options except that it acts on Execution Menu items.

(11) QUIT--this option returns the user to the Browse Menu. If there are any open queries, it prompts the user to either complete or abandon those actions before QUIT is executed.

d. Update Sub-Menu (Execution Level) (see Figure 2.11). This sub-menu provides a means for the user to enter or update information and relations. This ability applies to all actions normally performed in the creation of an object/relation, and the capability to change things such as object names, attributes, values, and relations. Access to this ability must be limited to specific users in order to preserve the integrity of the system, and that access will probably be determined by the database administrator. (The specifics of this process have not yet been determined. The following description assumes complete access.) If the mouse pointer is on an object, the selected object is expanded to an object skeleton, where the user can add to or change any part of the object. If not, the user is prompted to determine if he wants to create a link or an object. If he chooses an object, a blank object skeleton is generated where the user can identify names and information. If he chooses link, he is prompted for objects to be linked and the link field. Items included on this menu are:

(1) SAVE--this option allows the user to save his changes or additions. If he has created a new object, it will not yet be linked to any other, and will be displayed as an object, except that it will show no

links. If he has created or changed a link, the screen presentation will be altered to show this.

(2) ABANDON--this option allows the user to exit UPDATE without any changes or additions made.

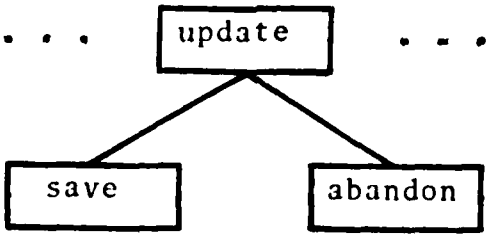


Figure 2.11 UPDATE Sub-Menu.

e. Print Sub-Menu (Execution level) (see Figure 2.12). This sub-menu allows the user to obtain a hardcopy of any object (including results) or screen display during the execution of the program. Items on its sub-menu include:

(1) SCREEN DUMP--this option prints the entire screen with the exception of menu items and user prompts, to enable the user to presentations and records of his work.

(2) PRINT OBJECT--this option prompts the user for the name of the object (which can be RESULT X) and sends it to print. It will be printed in tabular form with the name of the object displayed as the report title.

(3) PRINT REPORT--this option allows the user to print a previously formatted report. He will be required to enter two parameters: 1) the name of the report form. He can either type the name or depress the center mouse button and roll it down until the correct choice is highlighted and release it, and 2)

the object (including results) to be printed. Again, he can either type the name of the object or position the mouse over the object and press the select (right) button. These inputs can be in either order, but the second choice will be type-checked against the first to ensure the field names to be printed in the report actually exist in the object.

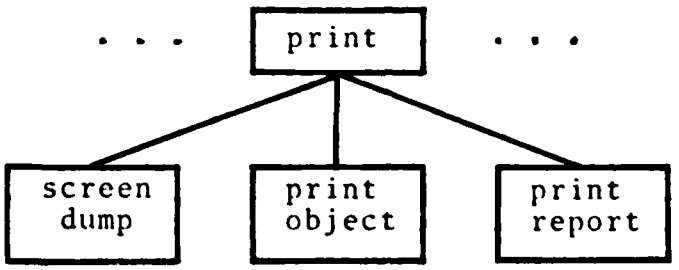


Figure 2.12 PRINT Sub-Menu.

3. Screen Display Characteristics.

Since GLAD is being written to be run on a graphics terminal, several advantages are gained in the screen display. The basic display, as shown in Figure 2.13, is as follows:

(1) All major program components are placed in windows. Included are menus, object-relation layout, and menu operations (e.g. LIST MEMBER, SELECT).

(2) All windows will include elevator bars to show the user where he is in relation to the entire window if it does not all fit into the allotted screen space.

(3) Any window size can be changed at any time by pressing the left (drag) button on the mouse and expanding or contracting the elastic rectangle. When the button is released, the amount of information displayed will be altered to accommodate that window size.

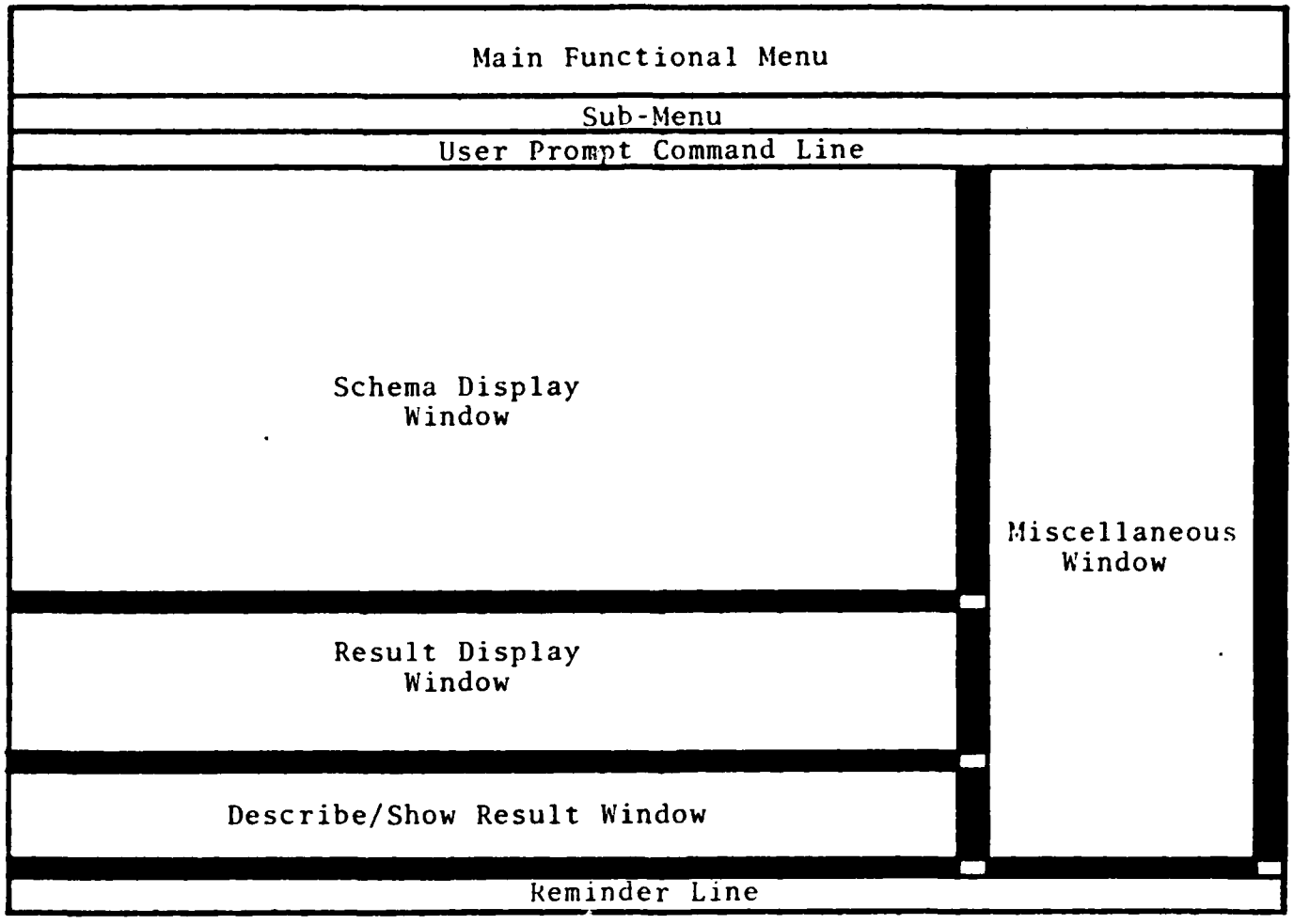


Figure 2.13 Screen Display.

4. Program Flow.

Basic program flow has been discussed at some length and the basic control flow diagram is shown in Figure 2.14. In amplification of those two sources, the following lists the steps required to operate the program.

(1) Boot or execute the program. This will present the Administration Menu and object display windows.

(2) Perform administrative tasks or select EXPLORE to enter the Browse Menu.

(3) Perform object/relation browsing and/or select QUERY to enter the Execution Menu.

(4) Do database querying at this level. Create and save results, then return to higher-level menus by selecting QUIT.

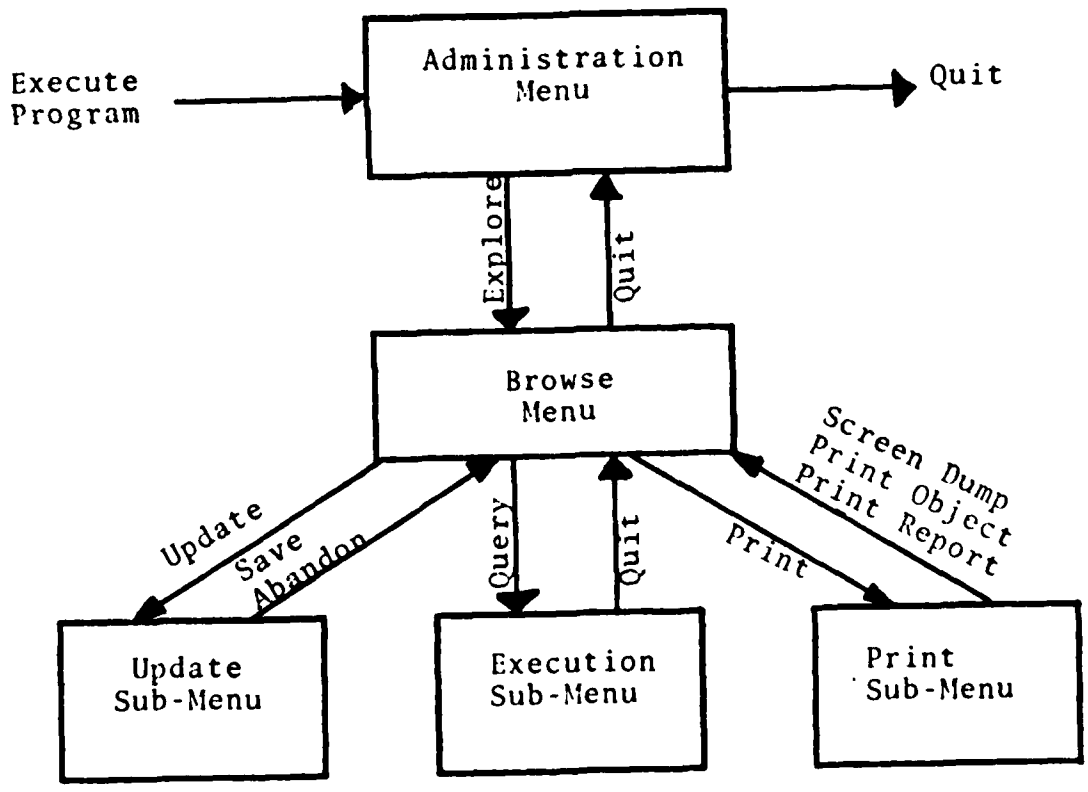


Figure 2.14 Basic Control Flow Diagram.

5. Use of the Mouse.

A three-button mouse is required for the operation of this program. Its use is as follows:

(1) Button 1. This is the "drag" button to be used for adjusting window sizes. To alter the size of a window, put the mouse pointer on the upper right corner of the window and depress button 1 (the left button). Keeping the button depressed, roll the mouse in the desired direction until the elastic window is the right size. Then release the button, and the window will expand/contract to the new size and the amount of information displayed (and the elevator bars) will be altered accordingly.

(2) Button 2. This is the "menu" button to be used for displaying and selecting menu items. When button 2 (the center button) is depressed, the same menu as appears along the top border will pop up. Keeping the button depressed, roll the mouse down until the desired option is highlighted. Then release the button, selecting the highlighted option. Repeating this procedure on a previously selected item will de-select it, clearing the operation and rewriting the screen.

(3) Button 3. This is the "select" button to be used for selecting an item. Roll the mouse until the pointer rests on the desired item, then press button 3 (the right button). Repeating this procedure on a previously selected item will de-select it, clearing the operation and rewriting the screen.

D. EXCEPTION HANDLING

This selection will not be all-inclusive in that we cannot at this time foresee all possible situations which might present "fatal errors". However, several areas can be discussed that will contribute significantly to consistency and "friendliness" in program execution. The design should incorporate the following procedures and use them as conceptual guidance in other exception handling decisions that might be required.

(1) In general, test specifically for acceptable input rather than testing for specific erroneous input. For example, let's look at the situation in which we are waiting for a REPORT FORM object to be selected. Rather than deciding how to handle a mouse click in the describe window, or a mouse click outside all windows, etc., ask "is the mouse positioned on an object?" If it is not, then it is an error, regardless of where the mouse was when it clicked.

(2) When an error has been detected, prompt the user for the correct input by printing a "reminder" along the bottom line of the screen (in inverse video). In the example above, the prompt might read "position mouse over desired object and click right button, or type object name".

(3) In situations where user confusion might be anticipated, look for specific errors that would enable the program to assist the user. In the example above, if the user selected another menu item when the object selection was expected, a reminder along the bottom of the screen might read "operation pending...must complete or de-select previous operation".

E. PRIORITIES IN DESIGN

The design for GLAD will be accomplished in two distinct stages. First will be the preliminary (or architectural) design. This entails decomposing the program into "black box" modules according to functions to be accomplished. It will concentrate on the properties of the modules and their interconnnections. The second stage of design will be a stepwise refinement of the abstractions introduced in the first stage. The completion of design will be defined as the point at which all modules are written in low-level algorithmic language appropriate for direct implementation in a programming language (in this case, the C programming language).

Accomplishing the design as described above supports a consistent, organized, hierarchical program structure. However, it does not lend itself well to assigning priorities to designing particular program components before others. In general, we do not desire to assign such priorities, but there are several areas which could be addressed that would enable early completion of a prototype of the program if it is deemed necessary. These areas include:

(1) Screen display. Program components such as generation of windows, use of elevator bars, and arrangement of windows on the screen are important to the program, but are not dependent upon the data structures or other aspects of the program. Therefore, implementation work on these components could begin as soon as the exact screen layout is designed.

(2) Use of the mouse. Again, the use of the mouse will not depend upon the details of the rest of the program, but only needs to know the general layout of pop-up menus and the use of each of the buttons.

(3) File handling. This aspect of the program is general in nature and is identical to any other in that files will be opened, appended, edited, deleted, and saved. The modules dealing with these functions could be implemented at any time.

F. DESIGN HINTS AND GUIDELINES.

Many aspects of the design have already been addressed in these specifications, so this section is intended to augment rather than supercede any previous design discussion. The following comments will assist in the design phase by providing the "policy" for program characteristics and user friendliness.

(1) Always design to allow the user to escape from any action before it is initiated. The user should always be able to press the "escape key" (actual keystroke not yet determined) to erase an operation before it begins. Let's look at CREATE RESULT for an example. In this case, the user will have selected the modules and determined the conditions, but may change his mind before selecting CREATE RESULT. Pressing the "escape key" will cancel all current SPECIFYs and return to the current menu. This ability will greatly enhance the user friendliness of the program to novice users, and will benefit sophisticated users as well.

(2) Do not make the user a slave to the program by making him memorize arbitrary formats. For example, the program may store all REPORT FORM files as a filename ending with ".form", but do not make the user remember to put ".form" at the end of all such filenames. Rather, make the program append it to the name supplied by the user. In addition, when printing these filenames in the CREATE REPORT FORM pop-up menu, take the ".form" off to avoid any user confusion.

(3) Do not force unnecessary information upon the user. It might confuse the novice user, and it would definitely annoy the sophisticated user. A good example of this point is the situation described in the "Exception Handling" section. When an error has been detected, it is appropriate to remind the user of what type of input is expected. Providing this reminder before that time could be both distracting and discomforting.

III. DESIGN

A. METHODOLOGY

The design phase of the life-cycle of a computer program is one which is of utmost importance. The reason is that its usefulness is directly related to the value placed on it by the software development team. It is sometimes poorly performed or omitted entirely. When this occurs, the implementation is much more difficult and the maintenance phase must depend entirely on the information supplied in the code by the implementors.

Conversely, a good design greatly simplifies implementation and facilitates maintenance by providing documentation and considering modern programming practices (as described in (FAI 85)). The benefits available from a good design of GLAD are even greater, since the implementor will not have the advantage of communications with the designer. For this reason, it is very important to find a representation which will be well-documented and easily understandable.

The design notation chose for this project is Hierarchy-Input-Process-Output (HIPO) diagrams. Some of the advantages HIPO provides include:

1. Improved Documentation.

Since HIPO is a multi-stage process, both the function and method of each module are well-explained. The advantage

is that the designer is not required to generate this documentation as a separate step: it is inherent to the process of designing the overview and detail diagrams.

2. Application of Modern Programming Techniques.

HIPO is a top-down design approach, which lends iteself well to the considerations of modern practices, such as choosing an efficient design technique and other design considerations (such as coupling and cohesion (YOU 79)).

3. Descriptions Vice Algorithms.

This can also be a disadvantage since the design cannot be mindlessly implemented, but it allows the implementor some freedom in implementation style and techniques. Since the implementation will be a follow-on project in this case, this HIPO property should prove advantageous.

B. DESIGN CONSIDERATIONS

Throughout this (and any other) design activity, many choices evolved. Some affected the nature of the program, some were "the best of several alternatives", and some were just a matter of style. Since our approach here is top-down, all decisions were put off as long as possible (and, indeed, many will be made during implementation). Of the decisions which were made during design, some are self-explanatory and are not discussed in this document. Others are minor (perhaps style) decisions, and are addressed in the "notes" section of the HIPO diagrams. There are a few,

however, which are important to the very nature of the program and are explained here.

C. DESIGN DECISIONS

Perhaps the most important decision is whether this program will result in a new database management program or serve as an interface to an existing program. While either decision could be accomplished, we decided to make this an interface. However, as discussed in Chapter I, we must ensure that the underlying query language is capable. The intention is that the implementation will take the actions to a certain level of abstraction, then a simple adaptive interface could be written that would "translate" GLAD's symbolic instructional words to those of any underlying query language. It is interesting to note, however, that the implementation could just as easily be written so that the instructions are in a particular query language to facilitate maintenance.

A second "early" design decision was the type of modularization that would be used. We decided to break the program down into modules that would most closely follow the static, hierarchical nature of the program, since that would provide modules which are easily recognizable by function. In addition, this provides very cohesive modules with minimum coupling (again, to make implementation straightforward and maintenance easier).

Another important decision involves the user interaction. Two aspects of that decision, which will be addressed here, are menus and windows. These are discussed at some length in (RAE 85a), and are summarized here to provide justification for the decisions we made. First, menus will be used because:

(1) They do not require the user to memorize any (arbitrary) syntax or reserved words.

(2) They encourage exploration.

(3) They are feasible with fast screen updates.

(4) They do not require much overhead.

Windows will be used because:

(1) They greatly increase the amount of information which can be displayed at one time by breaking it up into sections (this concept is explained in (IVE 82)).

(2) They provide easy access to different levels of the program, since they can be easily shifted by the use of a mouse.

D. DESIGN LAYOUT

As stated earlier, HIPO diagrams were chosen as our design representation. The figures in Appendix A are those HIPO diagrams. They can be broken down as follows:

(1) Figure 3.1 is the HIPO Table of Contents. It shows the hierarchical layout of the program and the numbering system which is used.

(2) The remainder of the figures are grouped by module. They are identified by the number which corresponds to the module number in the HIPO Table of Contents.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

There were two major conclusions reached during the research for (and the writing of) this thesis. They are both broad in nature and encompass several other findings.

1. GLAD is Necessary.

Although there are numerous products on the market which address database problems, one cannot find any product which fill the requirements of descriptiveness, power, and ease of use and learning. These properties are absolutely necessary in today's environment, since there is an ever-growing diversity in database users. GLAD will provide these properties.

2. GLAD Can Be Done.

There are many properties which must be incorporated into a database management system as was discussed in Chapter I. However, by a combination of incorporating the good properties of existing programs and designing new properties to meet the needs which have not yet been satisfied, we can develop a program which should help today's workers to become highly successful database users. The design for such a program has now been accomplished.

B. RECOMMENDATIONS

The implementation of GLAD represents the amount of work still remaining on this project. Much thought will be required during that phase, since many of the decisions (especially those of style) were left for that time. However, many ideas relating to the implementation presented themselves during the design. The following list of recommendations is provided to the implementor to assist him by presenting some further study which will be necessary and some suggestions regarding those decisions which remain.

1. Help Facilities/Error Messages.

This aspect of a program is one of the most important (and often the largest). There is a fine line between providing enough help for the novice user and being a nisance to the experienced user. (An excellent discussion of prompts and error messages is contained in (DEA 82)). It will be important to devise a scheme which is regular (provides the same error message for the same type of mistake), robust (does not allow the user to become "hung" in a process with no escape), and appropriate (allows for different levels of help/error messages depending on the user's experience level). This design allows for such a scheme by providing a error-handling module which can be called by, and returns to, any point in the program. In addition, since a parameter is passed identifying the

calling module and the type of error, it will be very simple to provide identical error messages for similar errors. (Incidentally, this feature should also aid maintenance by enabling the maintenance programmer to identify the exact point of call for any error message).

2. Use of Forms.

There are several instances in the design where forms are used, such as the expand object form. These forms are used for two reasons. First, it separates the structure of the form from the remainder of the module to facilitate ease of implementation and maintenance. Also, it allows for flexibility in the implementation, whereby the implementor can easily adjust the structure of the forms to make them all as similar as possible (which, in turn, makes the program easier to use because it is more regular).

3. Use of the Mouse.

There is no module in the design which addresses the use of the mouse. This is because it is intended that standard library routines be used in implementing mouse operation. Again, the reason for this is to provide standardization in the operation of programs for the user. He needs to feel comfortable that he will be able to operate this program in the same manner as he does others. The use of standard library routines will allow that to happen.

4. User Access Levels.

There is no mention in this design of access levels for database users. However, it is probable that any operational system will have the requirement of restricting access depending on the clearance level of the user. One example of such a need is the actual changing of data in the database. Naturally, one will not want to allow all users to change data values. This ability should be restricted to just a few people or billets. One recommendation of how to provide this feature is a "login" process at the beginning of the session. Depending upon the clearance level of the person logging in, access to certain features of the program could be restricted. In fact, menu displays could be altered so that choices are not presented to users who are not authorized to choose them.

5. Number of Active Databases.

As designed, the program recognizes only one active database. However, there is no reason that more than one database could be used, so long as the files used in queries are compatible. Such a capability would not affect the ability to keep the database themselves separate, but facilities would have to be added to determine where query sequences would be saved if requested (and others).

6. Window Sizes.

The screen percentage allocations made for the windows in this design were best-guess estimates of actual

requirements. They can be adjusted as necessary at the discretion of the implementor, based on more specific knowledge from empirical results or further research. It is also intended that the user be able to adjust the size of windows by the use of the mouse. There is no provision yet to allow him to change the permanent default value of window sizes, but that may be another feature the implementor could add. There could be circumstances, for instance, that a user would never have the occasion to use the query facility, but only browse the database. In this instance, it would be more beneficial to have a large schema display window at the expense of the result display window (without forcing the user to effect that change every time he uses the program).

6. Use of Prototypes in Implementation.

There is little discussion in this thesis regarding the order of module implementation. However, since this project is to produce a generally useful program (as opposed to an answer to a specific application), many of the features of GLAD will be evolutionary in nature. In fact, implementation of new features could continue indefinitely if desired. The consideration will be one of value versus time to implement. Because of this property, it would be beneficial to use prototyping in order to see which features are naturally important and which might prove superficial. One recommendation towards this goal is to implement the

basic screen display and windows first (with the use of dummy variables and modules where required), then build from that point, one module at a time. This procedure would also aid the implementor in debugging and incremental testing.

7. Use of Color.

This design is intended to be displayed on a monochrome monitor, but there are some advantages which could be gained on a color system. For example, different colors for different windows might help the user to focus on the most important part of the screen. There is one caution, however. It is very easy to overuse color to the point where the information is obscured. Any use of color should be carefully examined to ensure a positive contribution results.

8. Implementation of Additional Related Packages.

There are several possibilities for additional programs that could be used in conjunction with GLAD. One such program might be a chart/graph package. While the original implementor will not be able to undertake such additional challenges, he should certainly watch for such possibilities and guide his implementation so as to easily accept them.

APPENDIX

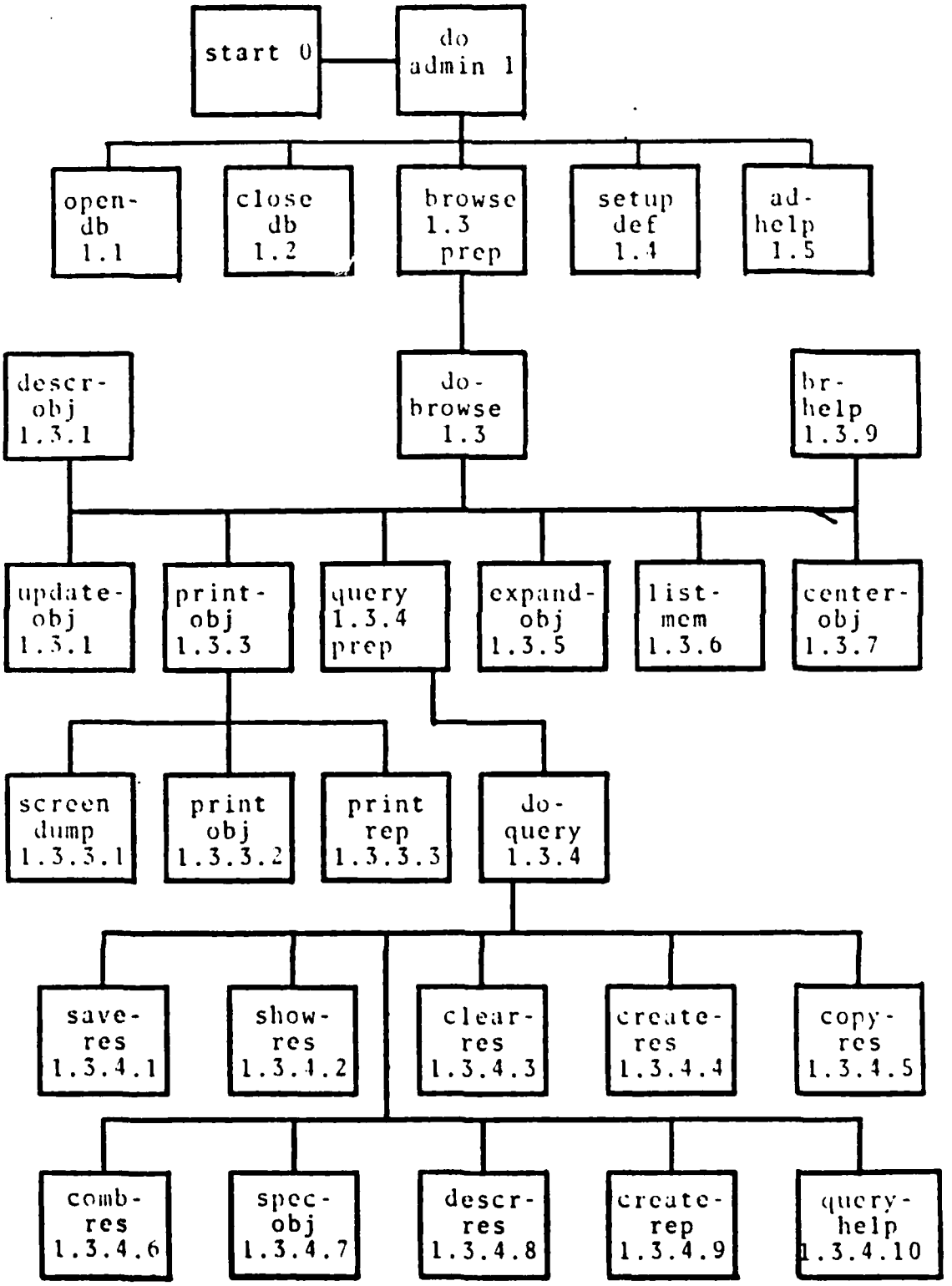


Figure 3.1 HIPO Table of Contents (Main Modules).

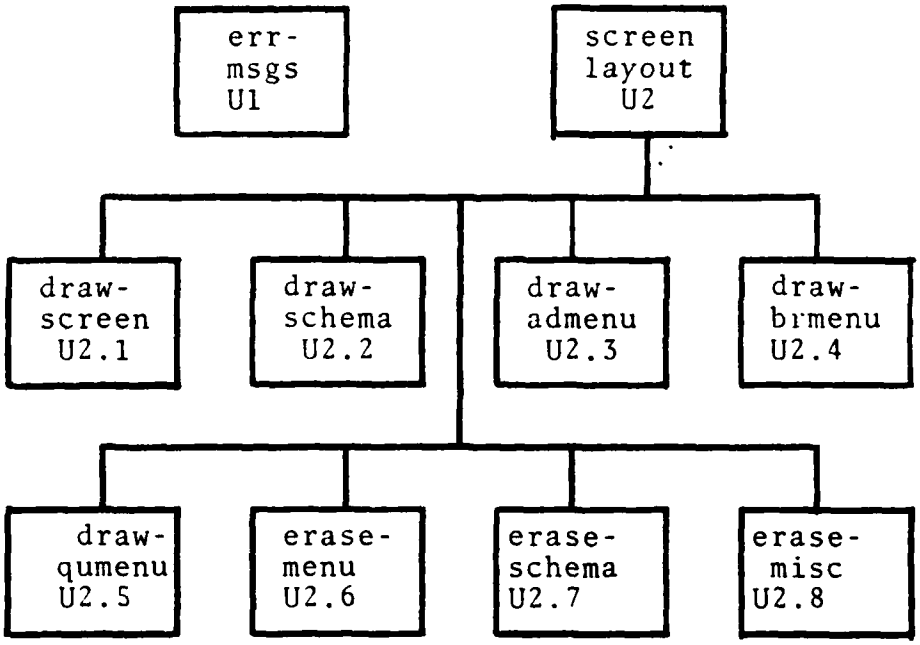
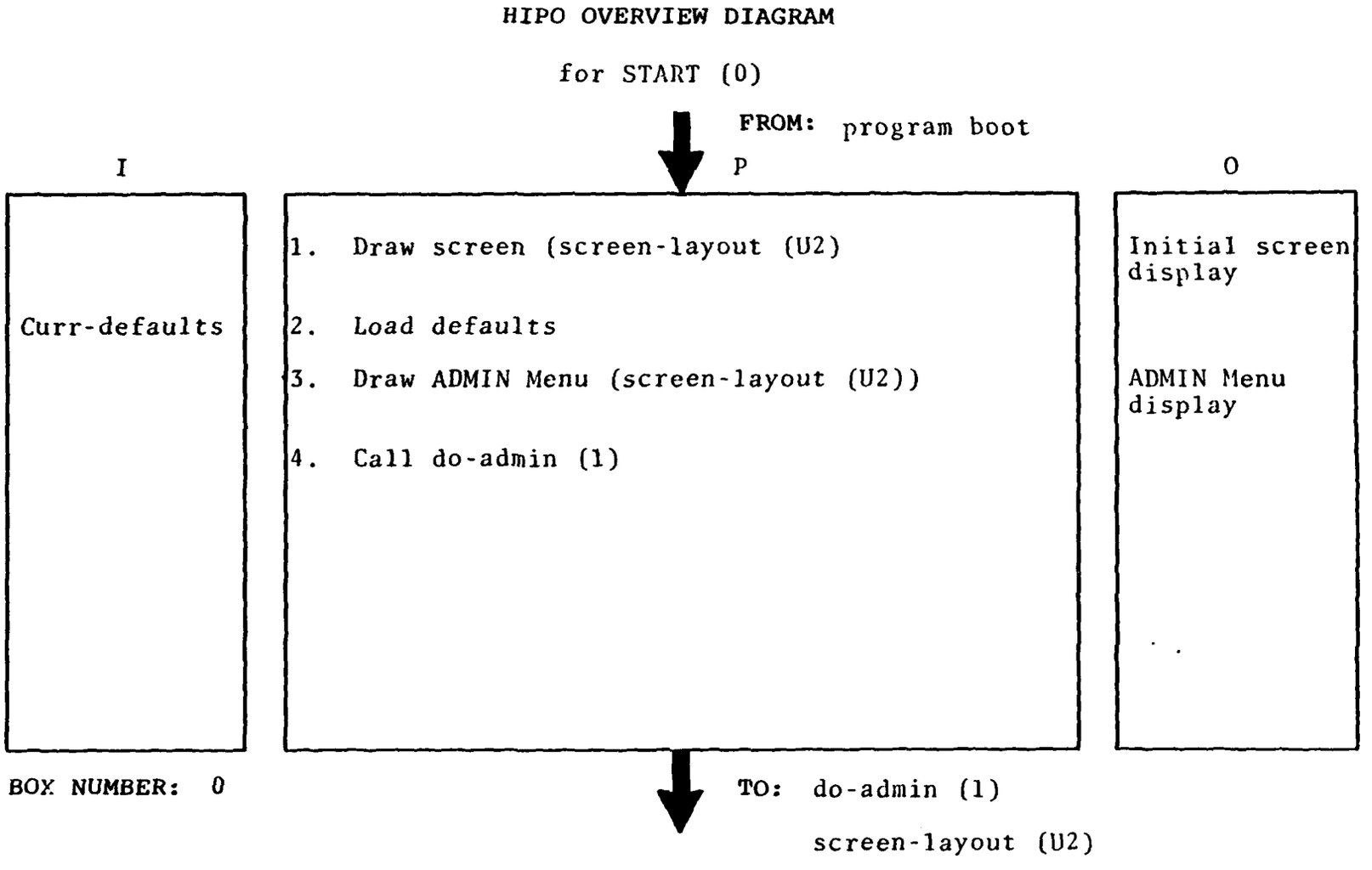


Figure 3.2 HIPO Table of Contents (Utility Modules).



HIPO OVERVIEW DIAGRAM

for START (0)

FROM: program boot

I	P	O
Curr-defaults	1. Draw screen (screen-layout (U2) 2. Load defaults 3. Draw ADMIN Menu (screen-layout (U2)) 4. Call do-admin (1)	Initial screen display ADMIN Menu display

BOX NUMBER: 0

TO: do-admin (1)

screen-layout (U2)

Figure 3.3 Start.

HIPO DETAIL DIAGRAM

for START (0)

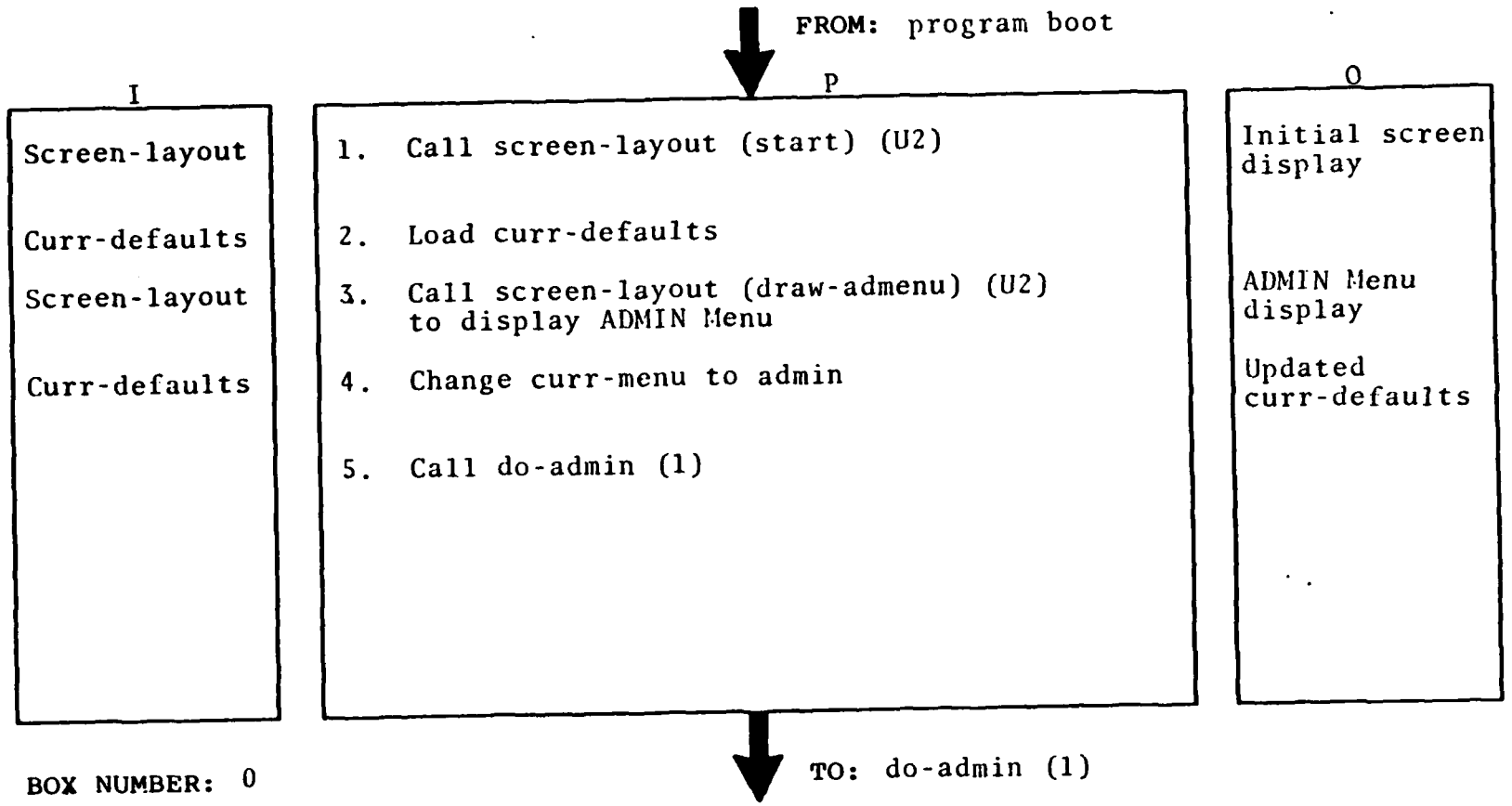


Figure 3.4 Start

HIPO NOTES
for START (0)

Step 1	Note
1.	The parameter "start" must be passed to screen-layout for the proper sub-module to be called
2.	Curr-defaults are contained in a file named curr-default-file. Items in the file include curr-menu, auto-res-show, db-sys-name, and any others which should be global to the program
3.	The initial set-up is now complete, so pass control to the administration menu module.

Figure 3.5 Start.

HIPO OVERVIEW DIAGRAM
for DO-ADMIN (1)

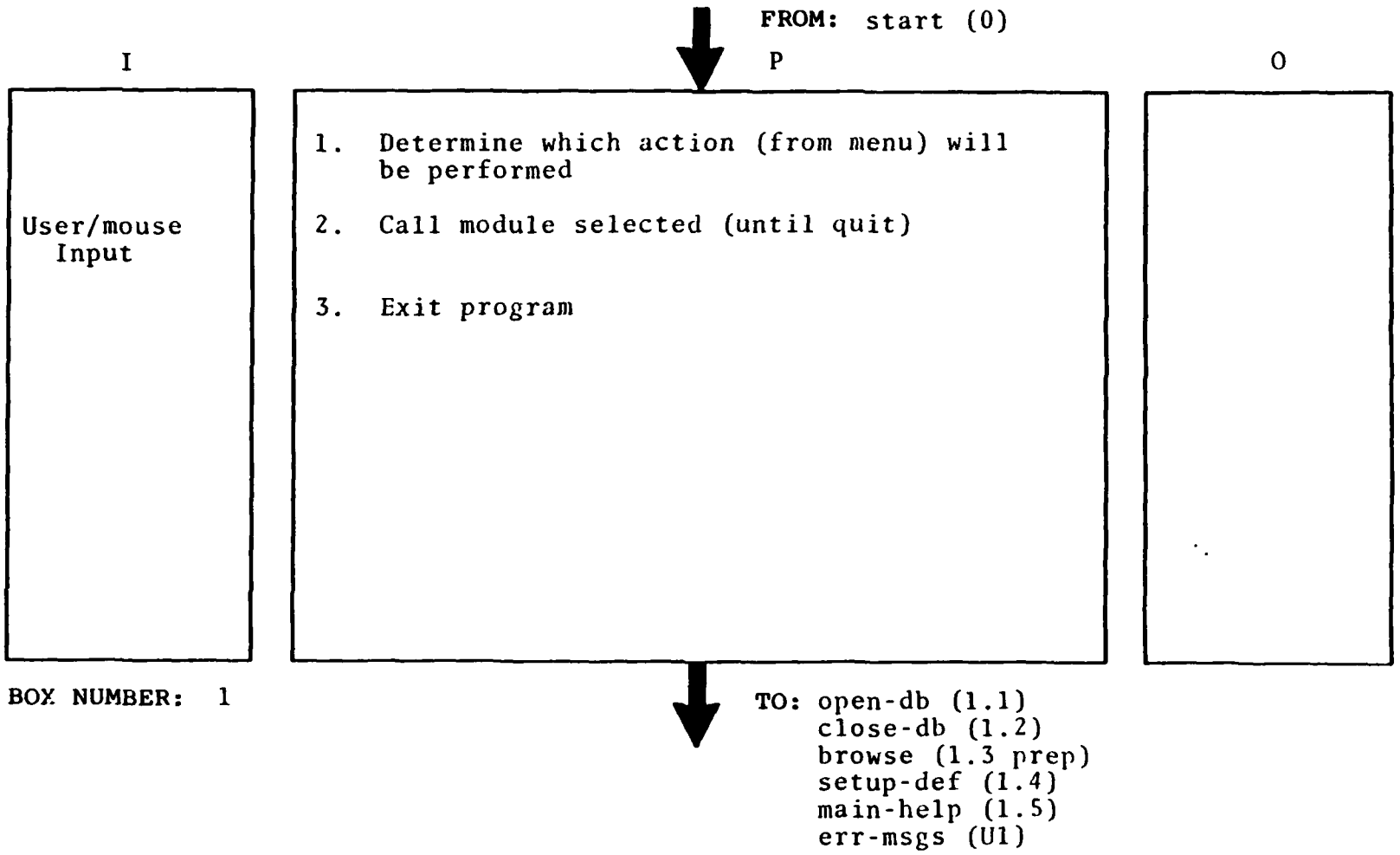


Figure 3.6 Do-Admin

HIPO NOTES

for DO ADMIN (1)

Step	Note
1.	The ADMIN Menu is displayed, but no selection has been made. Execute a check-for-input loop until a mouse button is pressed.
3.	Allow the user to assure himself that he has saved/abandoned all changes & additions before quitting

Figure 3.7 Do-Admin.

HIPO OVERVIEW DIAGRAM

for OPEN-DB (1.1)

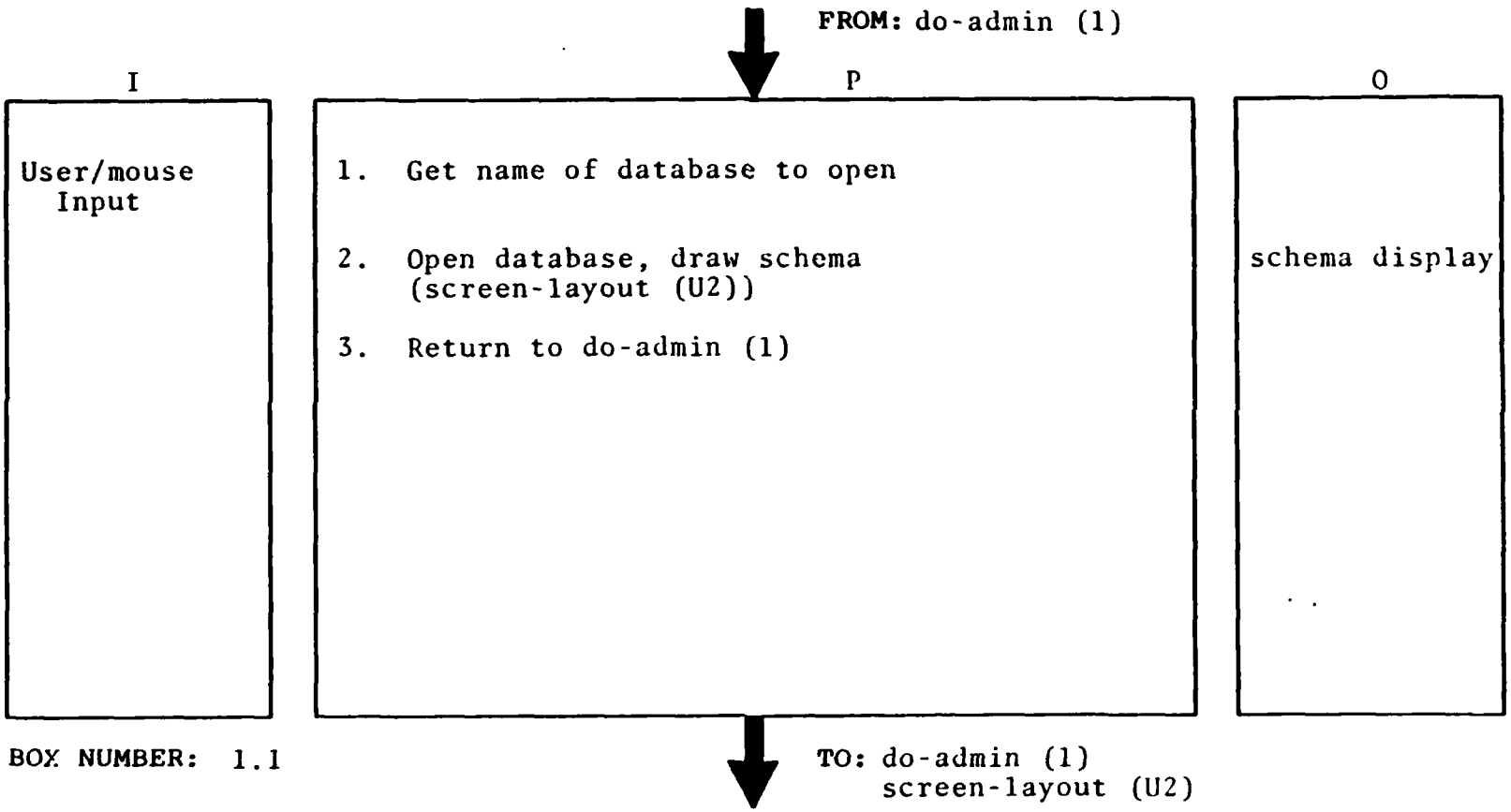


Figure 3.8 Open-db.

HIPO DETAIL DIAGRAM
for OPEN-DB (1.1)

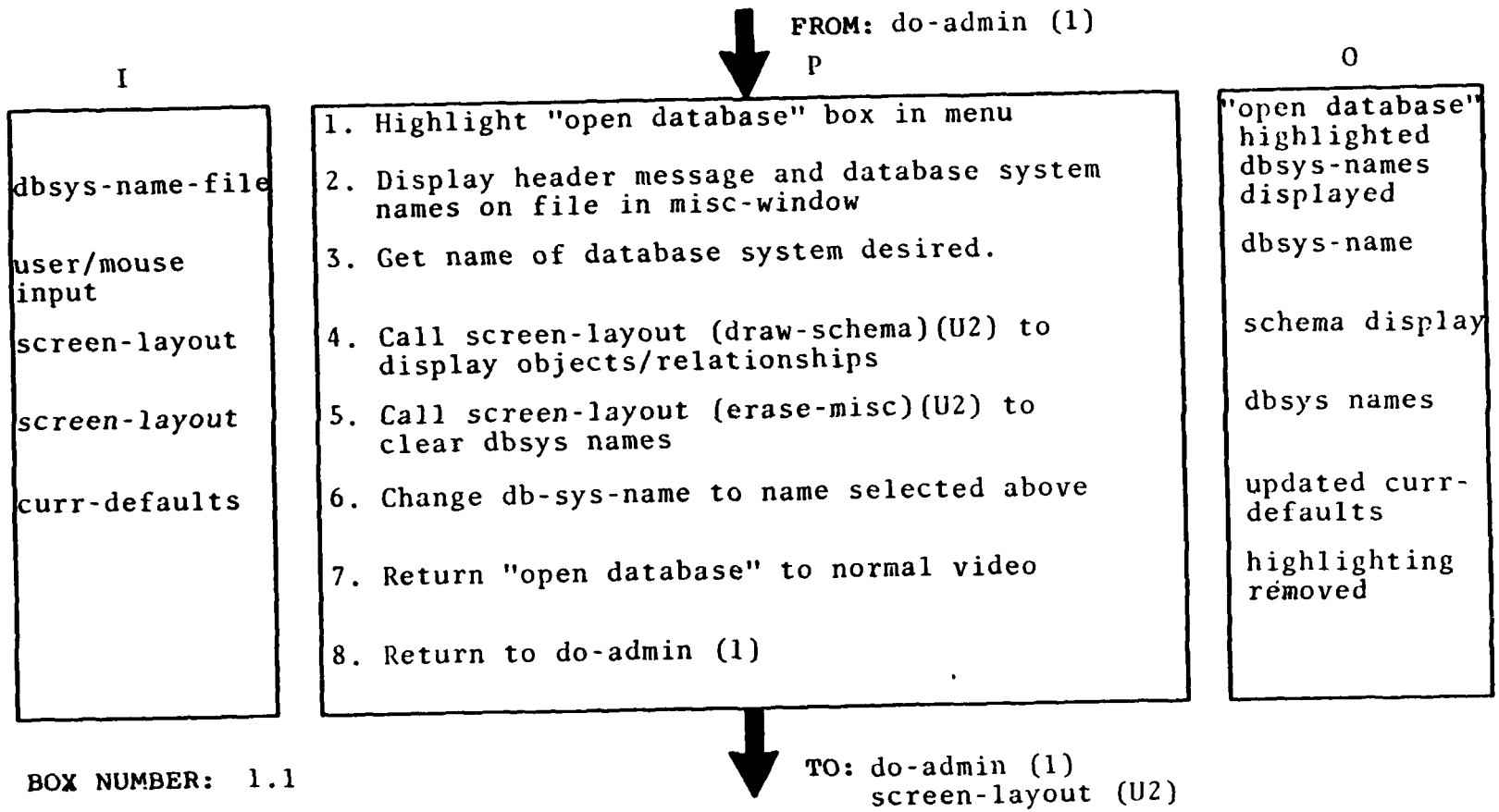


Figure 3.9 Open-db.

HIPO NOTES
for OPEN-DB (1.1)

Step	Note
1.	Throughout program, highlight action as it begins and return it to normal video when action is complete.
2.	Header message may say "Select one of the following databases on file:"
3.	db-sys-name is one of the parameters which must be accessible at any time during program execution. The distinction is made here that curr-defaults is a set of global variables used throughout the program. This is not to be confused with curr-defaults-file, which is the file where the "permanent values" of these variables are stored and is changed only by executing the setup_defaults module.

Figure 3.10 Open-db

HIPO OVERVIEW DIAGRAM
for CLOSE-DB (1.2)

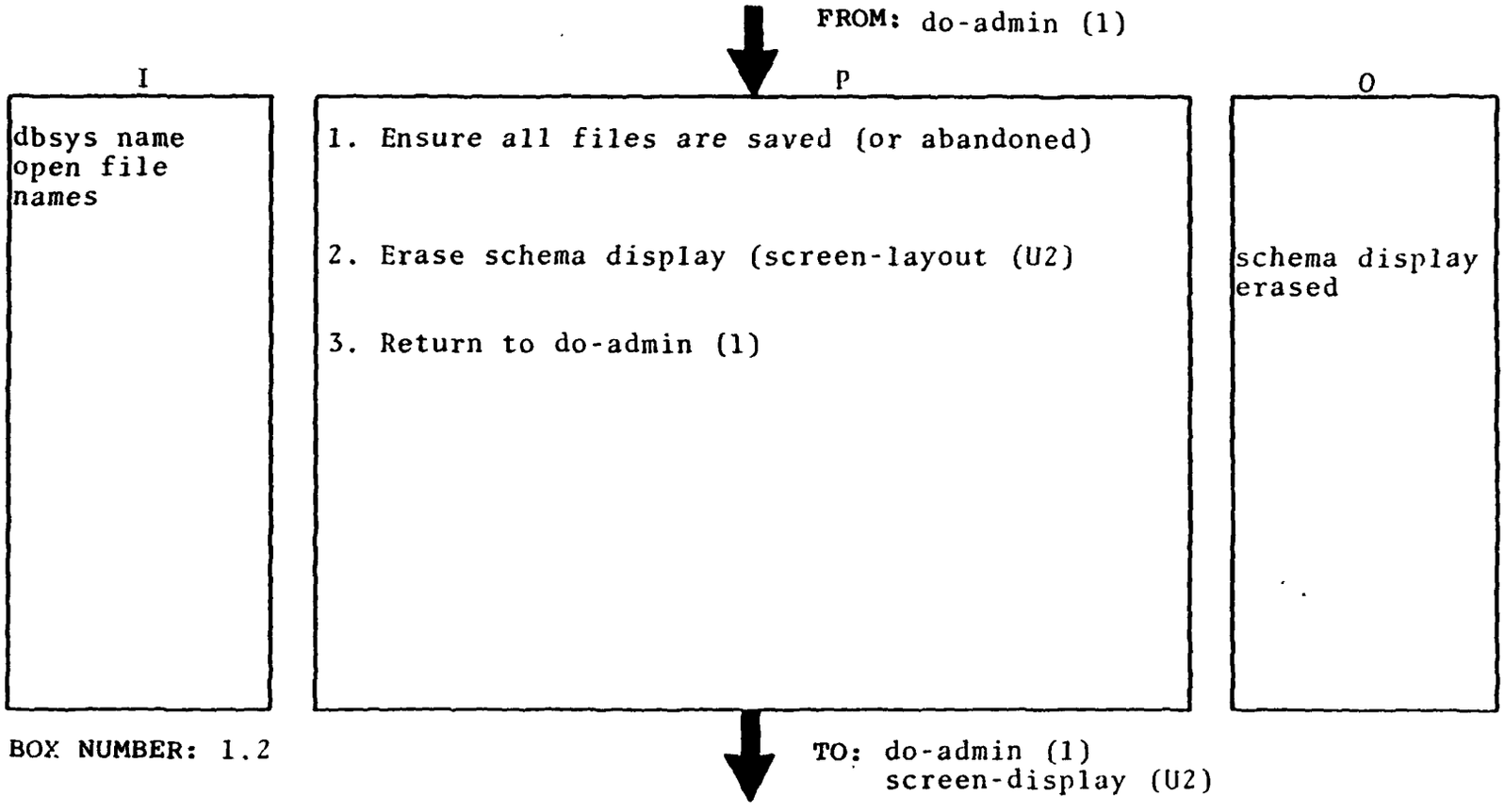


Figure 3.11 Close-db.

HIPO DETAIL DIAGRAM
for CLOSE-DB (1.2)

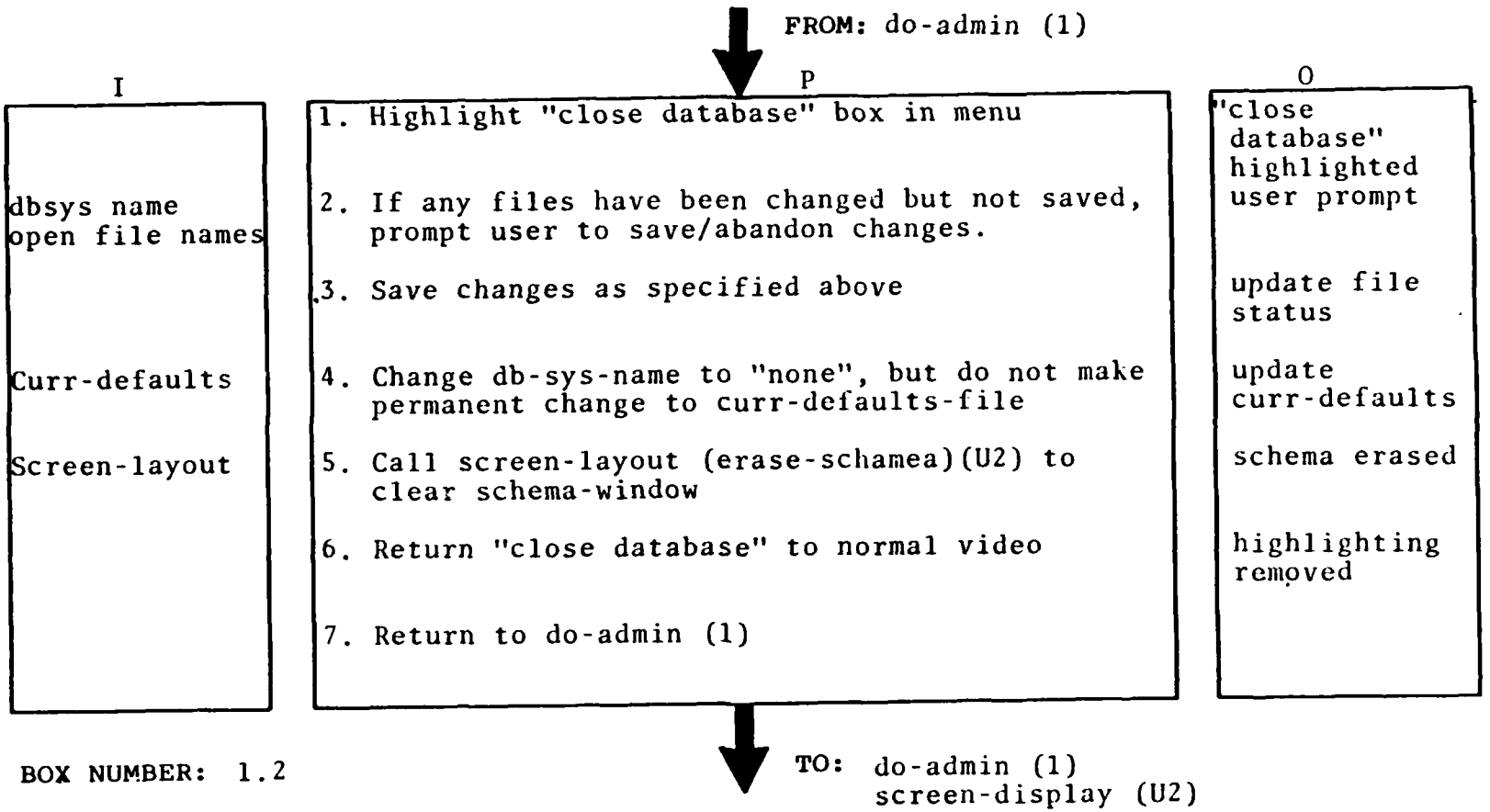


Figure 3.12 Close-db.

HIPO NOTES
for CLOSE-DB (1.2)

Step	Note
2.	It is not anticipated that any changes could remain unresolved (saved/abandoned) at this point of the program. This step is intended to be an "insurance" check.
4.	Permanent changes to the curr-defaults-file are only made in the setup-defaults module.

Figure 3.13 Close-db.

HIPO OVERVIEW DIAGRAM
fpr BROWSE (1.3 prep)

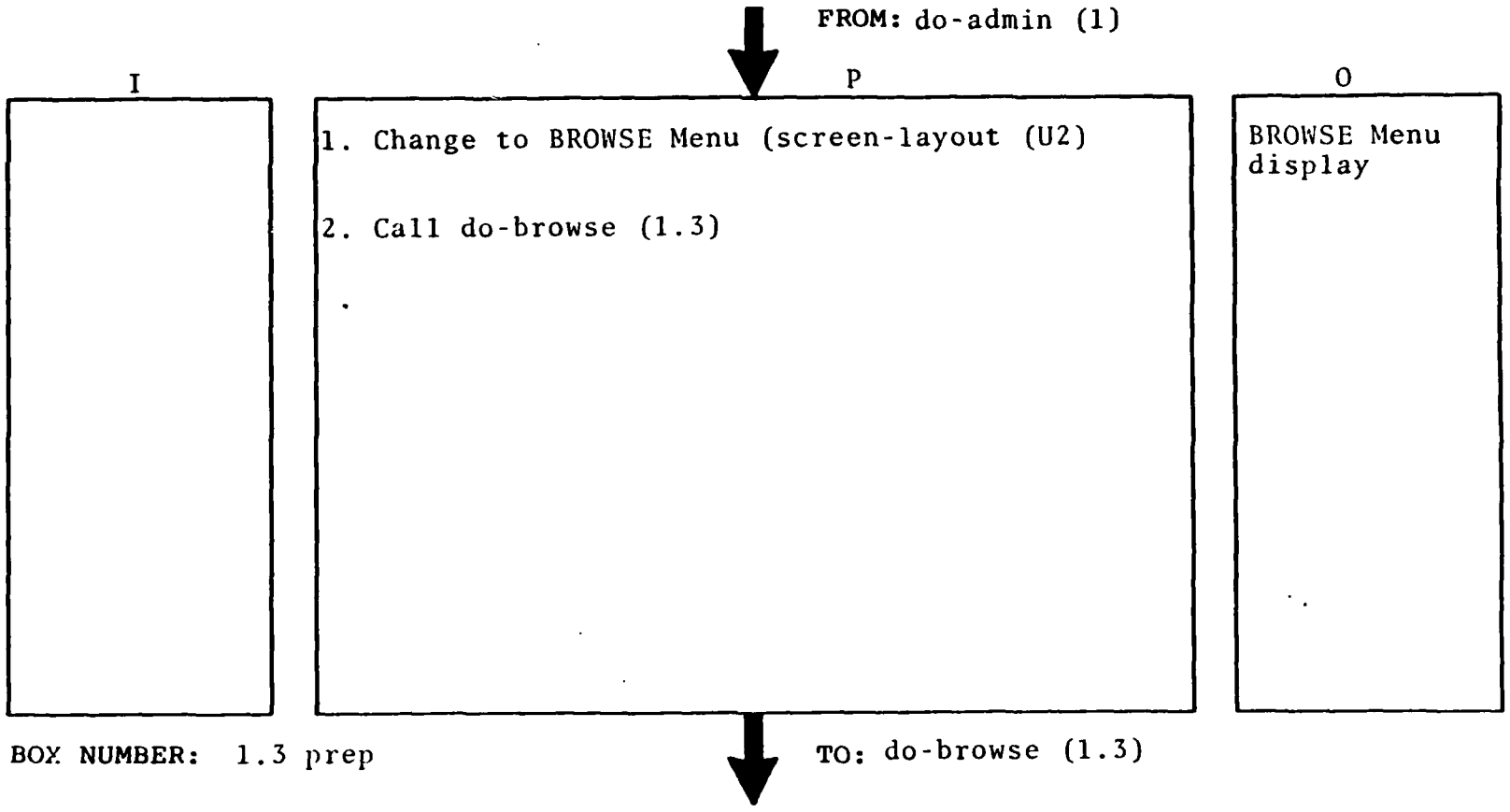


Figure 3.14 Browse.

HIPO DETAIL DIAGRAM
for BROWSE (1.3 prep)

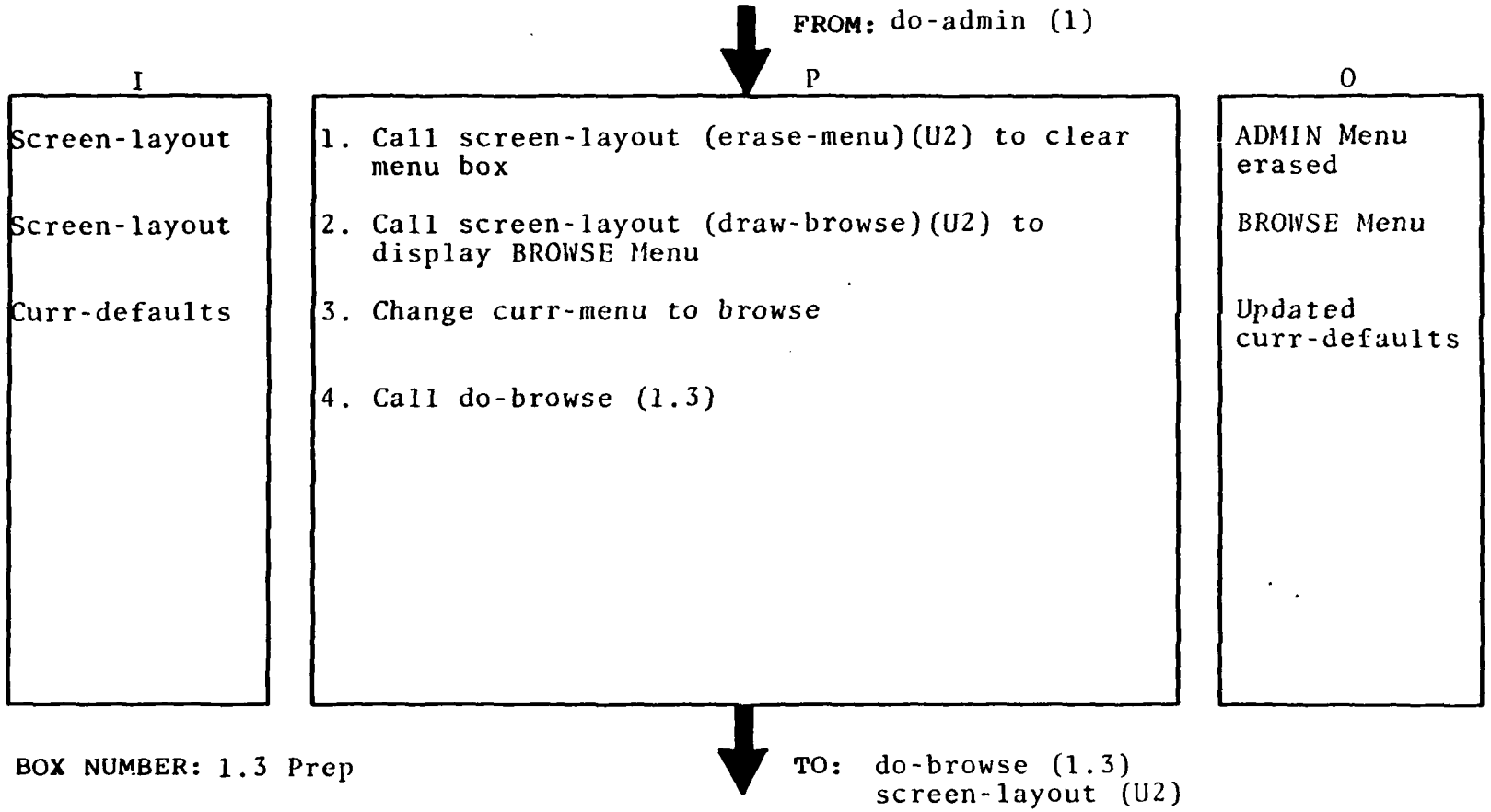


Figure 3.15 Browse.

HIPO NOTES

for BROWSE (1.3 Prep)

Step	Note
	This preparatory module, and a similar one named Query in the BROWSE Menu, are included as "lead-ins" to the main menu control modules. Since we want the program to execute as efficiently as possible, we do not want the menu re-written every time a subordinate module returns control to the main menu control modules (do-admin, do-browse, do-query). The inclusion of these "prep" modules enables us to accomplish the initial set-up activities without having to repeat them.

Figure 3.16 Browse.

HIPO OVERVIEW DIAGRAM
for DO-BROWSE (1.3)

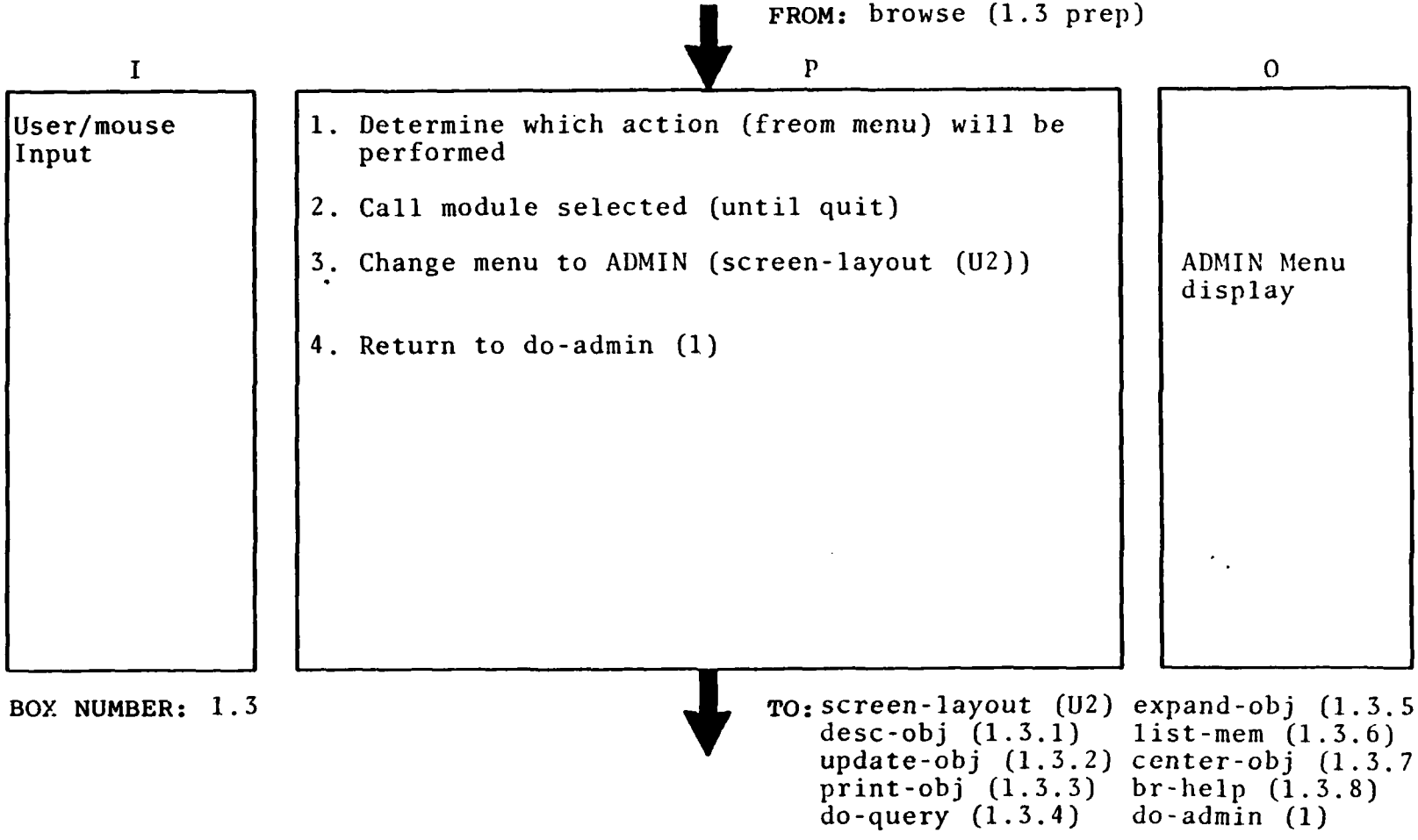


Figure 3.17 Do-browse.

HIPO DETAIL DIAGRAM
for DO-BROWSE (1.3)

FROM: browse (1.0)

I	P	O
User/mouse Input	1. Wait for mouse input	
User/mouse Input	2. If mouse is positioned in a menu box (except quit) call corresponding module	
	3. If (not quit) error msg, return to step 1 else	Error message
Screen-layout	4. Call screen-layout (erase-menu) (U2) to clear menu box	BROWSE Menu erased
Curr-defaults	5. Change curr-menu to admin	Updated curr-defaults
Screen-layout	6. Call screen-layout (draw-adminu) (U2) to dispaly ADMIN Menu	ADMIN Menu display
	7. Return to do-admin (1)	

BOX NUMBER: 1.3

TO: screen-layout (U2)
descr-obj (1.3.1)
update-obj (1.3.2)
print-obj (1.3.3)
do-query (1.3.4)
expand-obj (1.3.5)
list-mem (1.3.6)
center-obj (1.3.7)
br-help (1.3.8)
do-admin (1)
err-msgs (U1)

Figure 3.18 Do-browse.

HIPO NOTES

for DO-BROWSE (1.3)

Step	Note
3.	If we got past step 2, the only valid choice was "quit", which returns us to the ADMIN Menu control module. If "quit" is not the selection at this point, there was a mistake in the input, so output an error message and return to the top and let the user try again.
4-7.	The remainder of the steps perform a "prep" function for do-admin (1). We cannot call start (0) to do this since start (0) also does program initiation, such as loading Curr-defaults.

Figure 3.19 Do-browse.

HIPO OVERVIEW DIAGRAM
for SETUP-DEF (1.4)

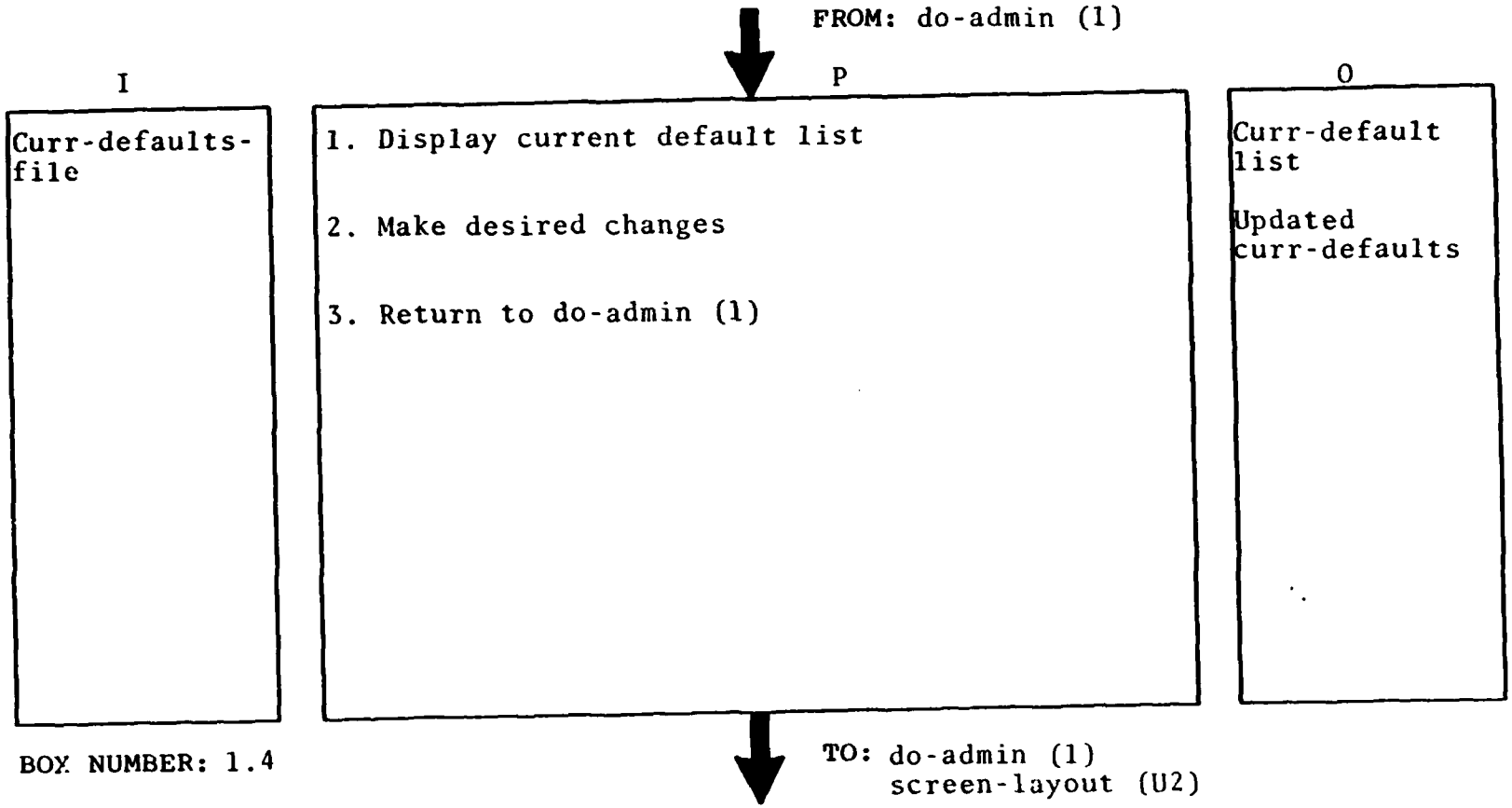


Figure 3.20 Setup-def.

HIPO DETAIL DIAGRAM
for SETUP-DEF (1.4)

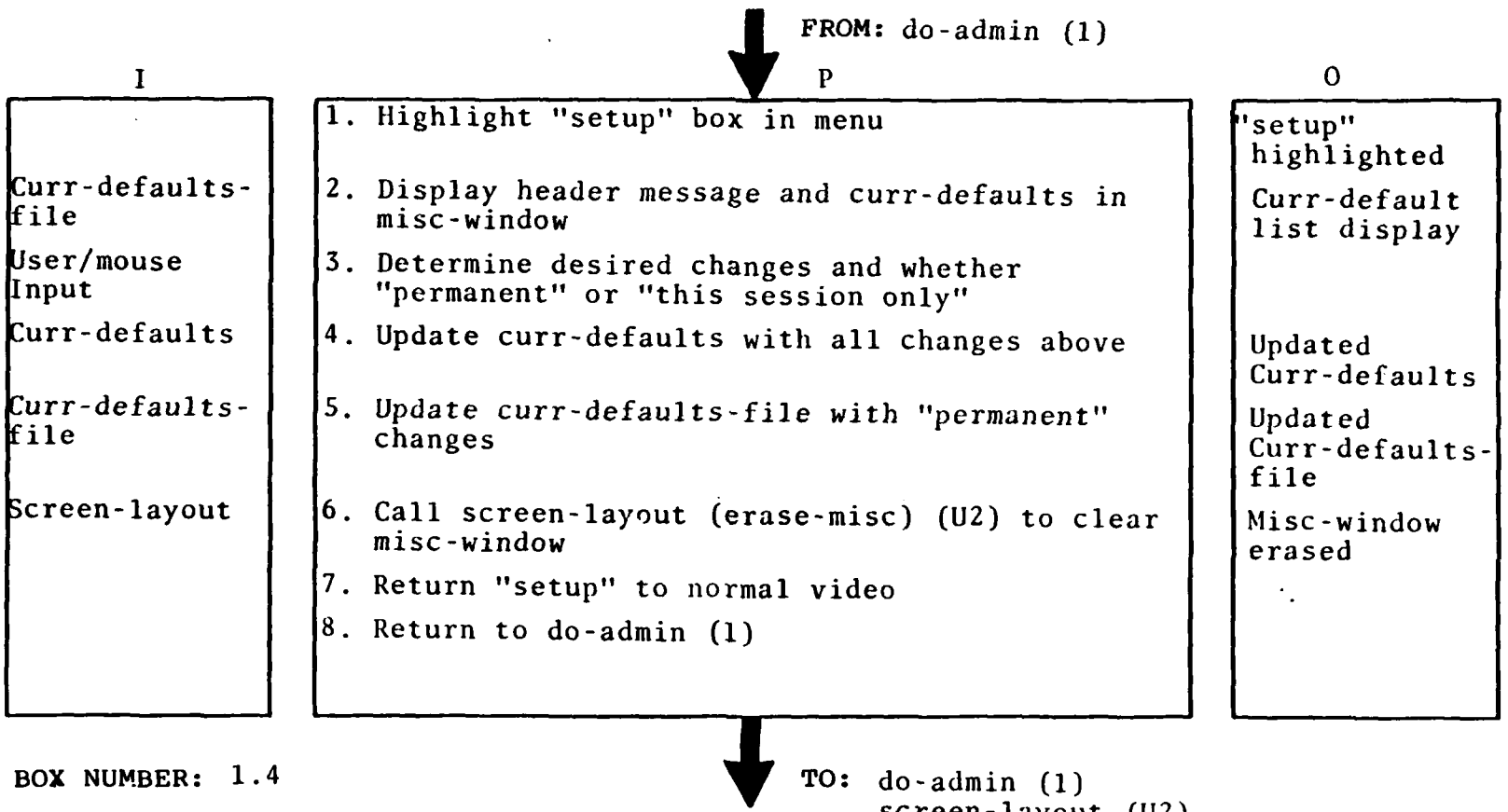


Figure 3.21 Setup-def.

HIPO NOTES
for SETUP-DEF (1.4)

Step	Note
2.	Header message might say "Following is a list of current system parameters. Move curser to appropriate box to make changes."
4.	This step only affects the variables being used during the current program run. When the program is exited, they will be lost.
5.	This step changes the file that stores the defaults. The new values will be saved and will be loaded by start (0) the next time the program is run.

Figure 3.22 Setup-Def (1.4)

HIPO OVERVIEW DIAGRAM
for AD-HELP (1.5)

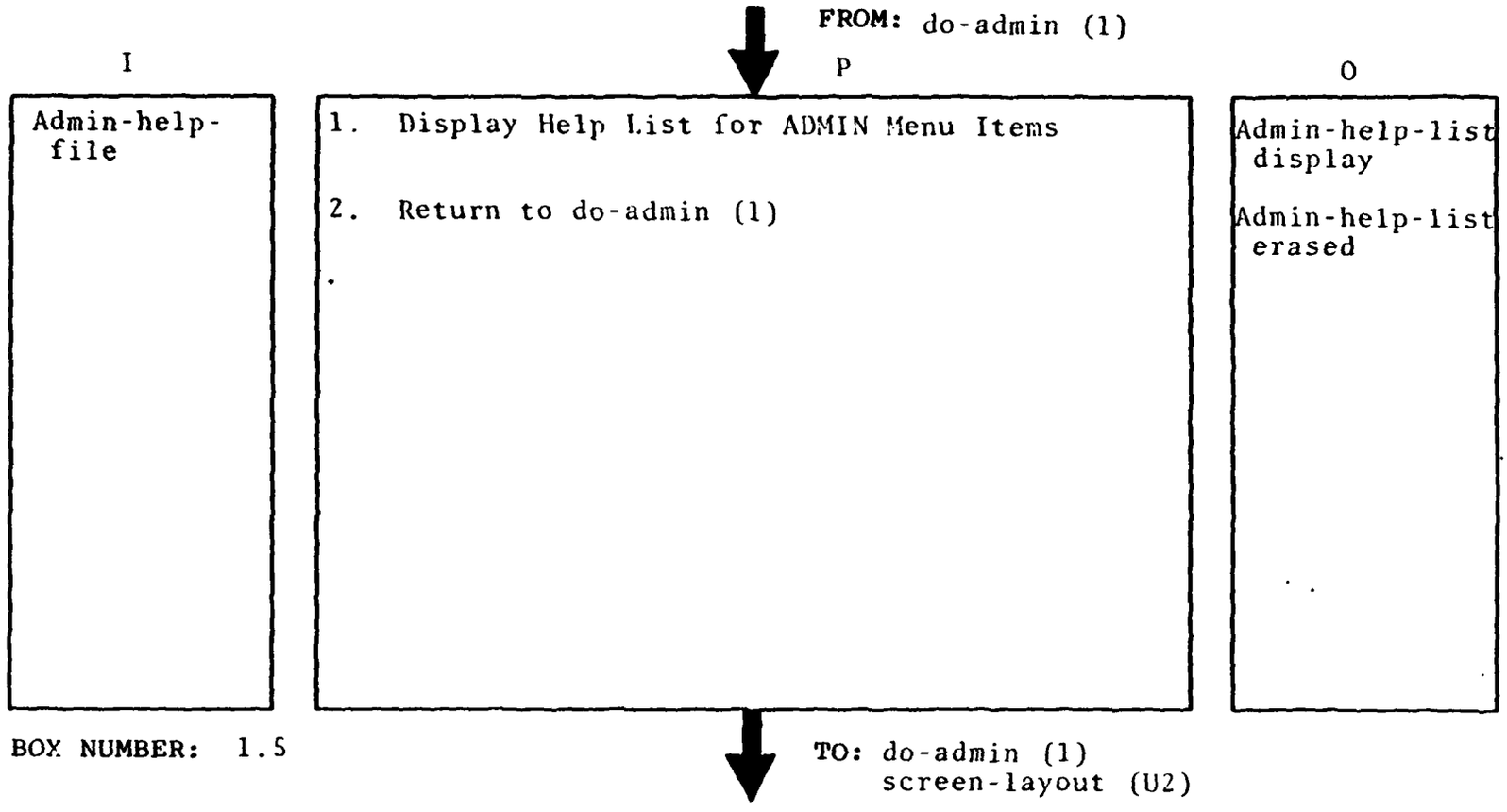


Figure 3.23 Ad-Help

HIPO DETAIL DIAGRAM
for AD-HELP (1.5)

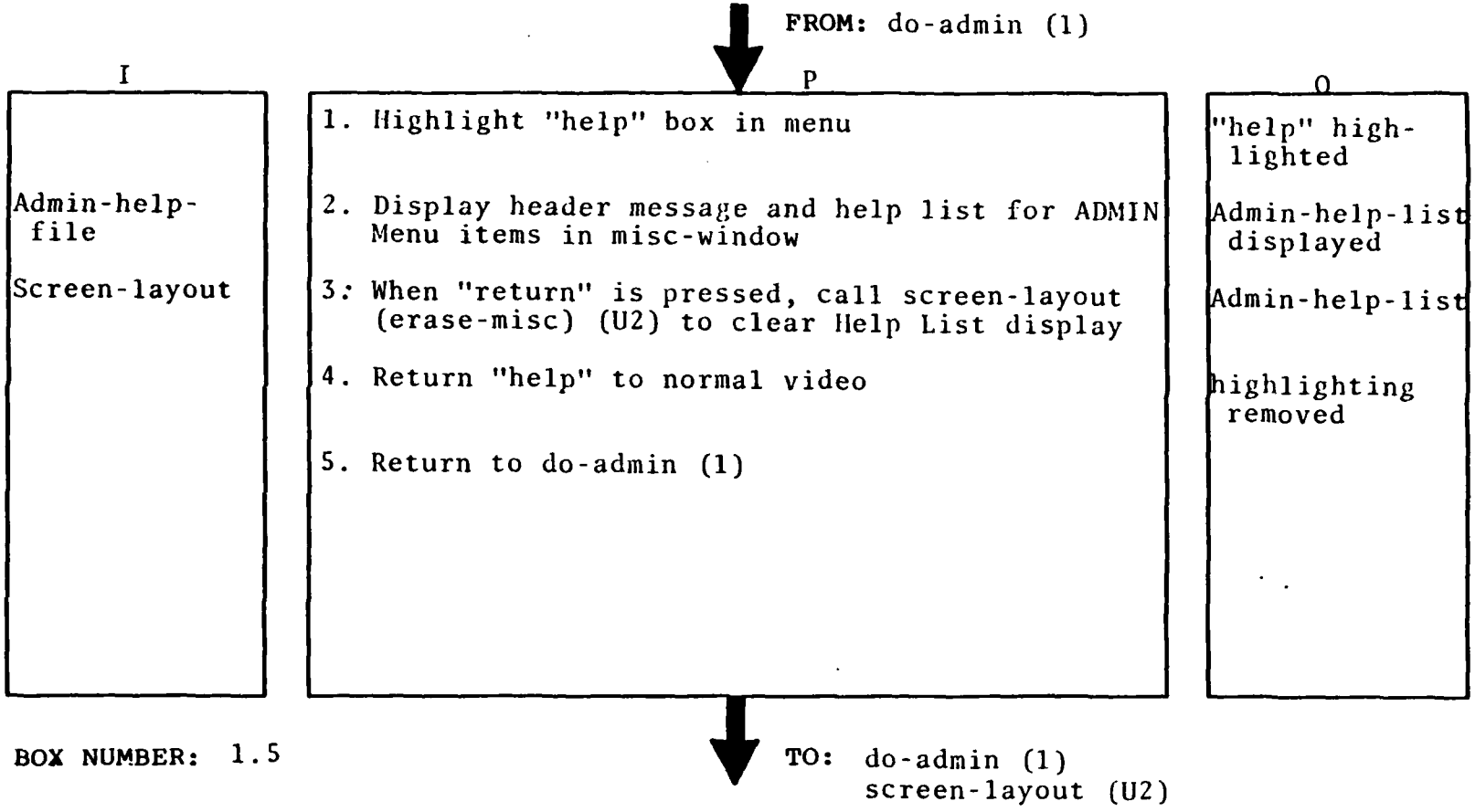


Figure 3.24 Ad-Help

HIPO NOTES
for AD-HELP (1.5)

Step	Note
	This module is currently designed to provide a simple list to the user. Each menu item is described along with the types of actions which will be required of the user. It might be more useful to prompt the user for a menu item he wants described, and present only that item. Or, better yet, allow the user to choose the "help" option anywhere during the execution of the menu or submenu items, and provide very specific guidance regarding the actions expected of the user at that point. Deciding which of the first two methods to design was simply a matter of personal style. A decision to use the last method should be made only after much deliberation, because it implies that a help message be prepared for every possible user situation (very difficult indeed).
3.	Any keystroke can be used here to symbolize completion. I recommend that the particular symbol be chosen to coincide with the one used on the implementation system to indicate a "continue" selection.

Figure 3.25 Ad-Help (1.5)

HIPO OVERVIEW DIAGRAM
for DESCR-OBJ (1.3.1)

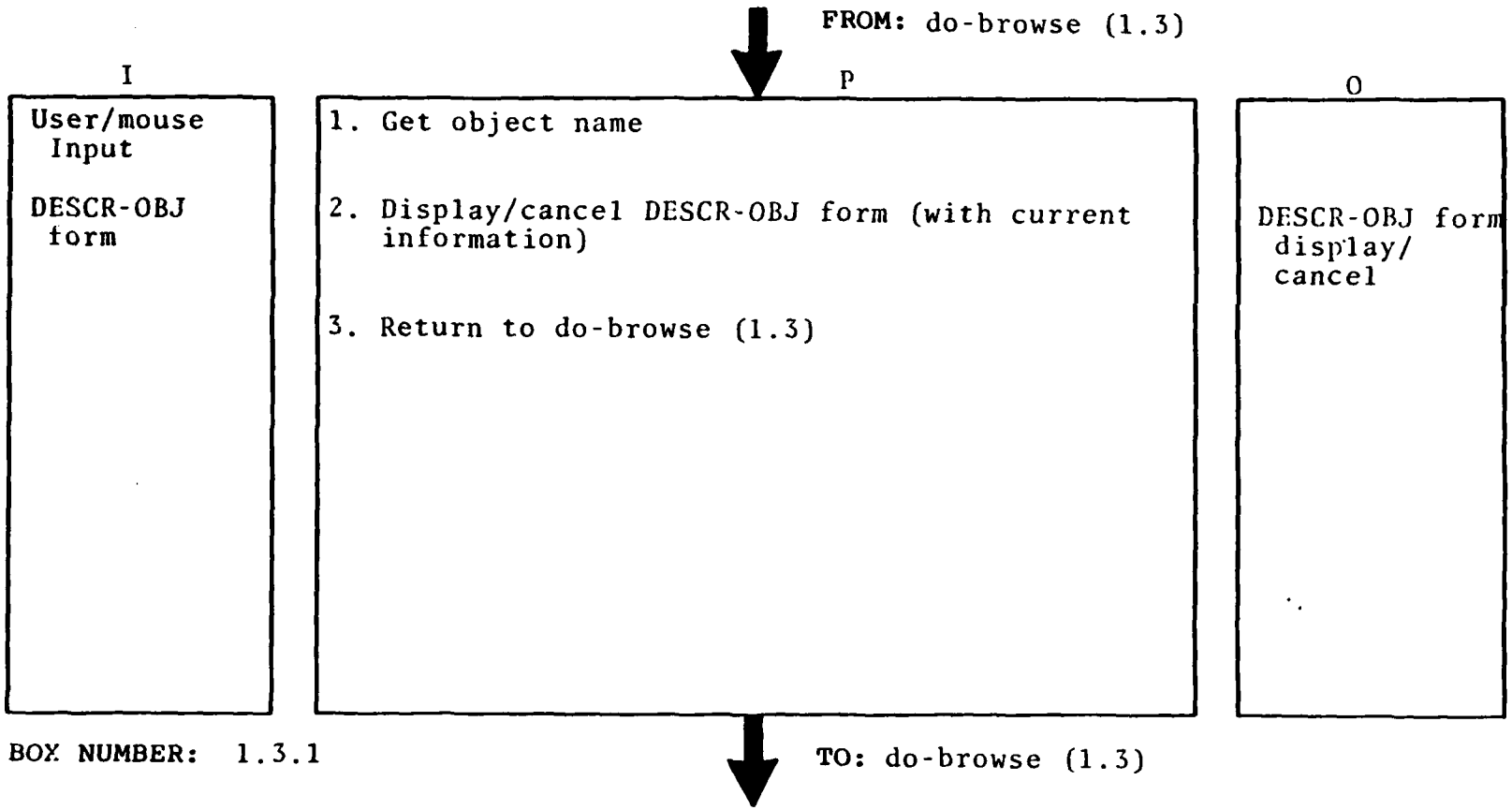


Figure 3.26 Descr-Obj

HIPO DETAIL DIAGRAM
for DESCR-OBJ (1.3.1)

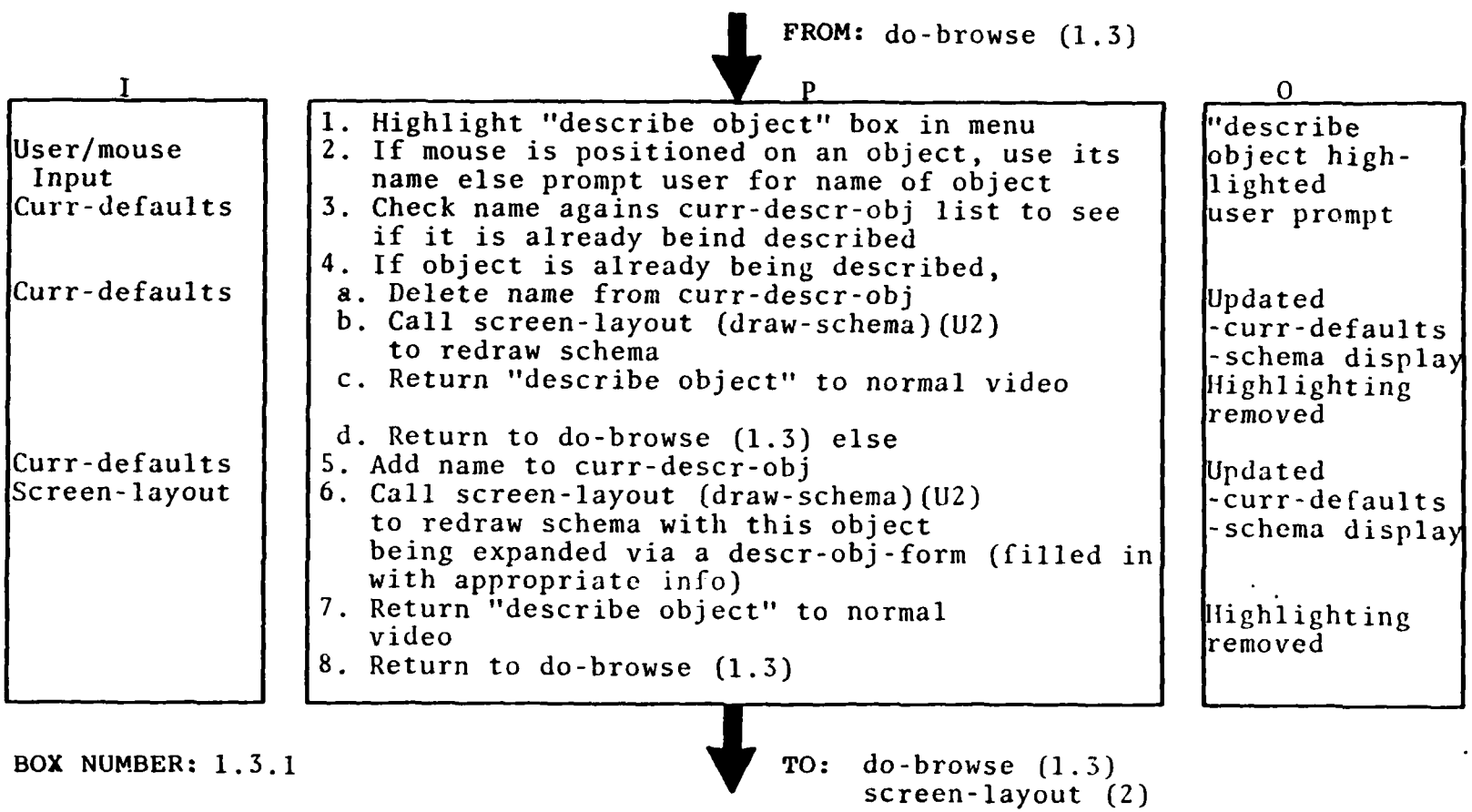


Figure 3.27 Descr-Obj

HIPO NOTES

for DESCR-OBJ (1.3.1)

Step	Note
3.	Curr-descr-obj is a list of objects which are being described in the schema presentation.
4-8	This "describe object" selection should act as a toggle for each object. If the object is not being described, selecting "describe object" will cause it to be described. Conversely, if it is already being described, this selection causes the description to be cancelled and the schema to be rewritten without it
4-6	The method of display for an object description is as follows: -> use a small window format which will overwrite the object in the schema display -> this window will have standard boxes which will be filled in from information in the object file, such as name, fields, etc. This action is actually performed in the screen-layout module, but is introduced here for clarification.

Figure 3.28 Descr-Obj

HIPO OVERVIEW DIAGRAM
for UPDATE-OBJ (1.3.2)

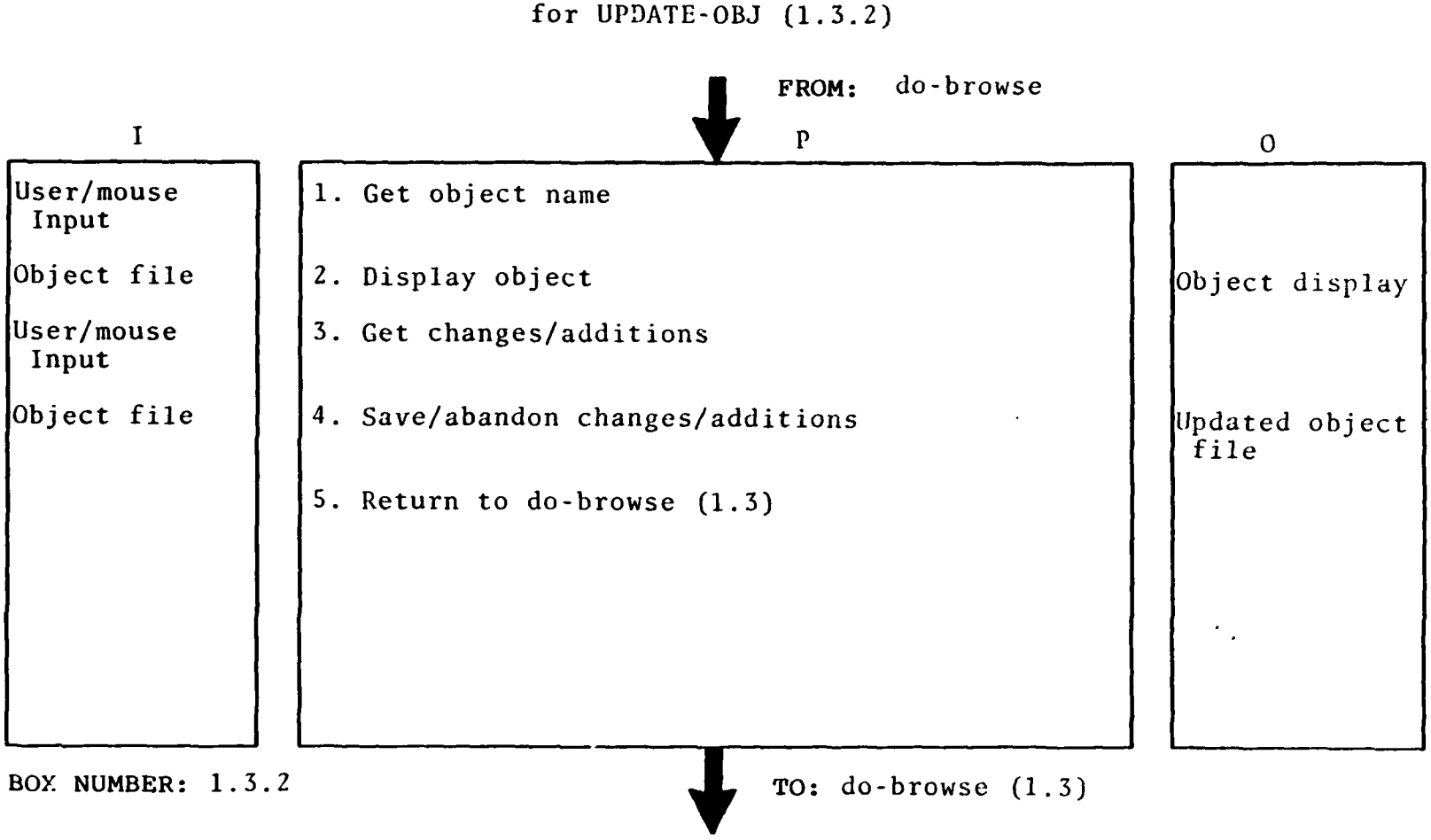


Figure 3.29 Update-Obj

HIPO DETAIL DIAGRAM
for UPDATE-OBJ (1.3.2)

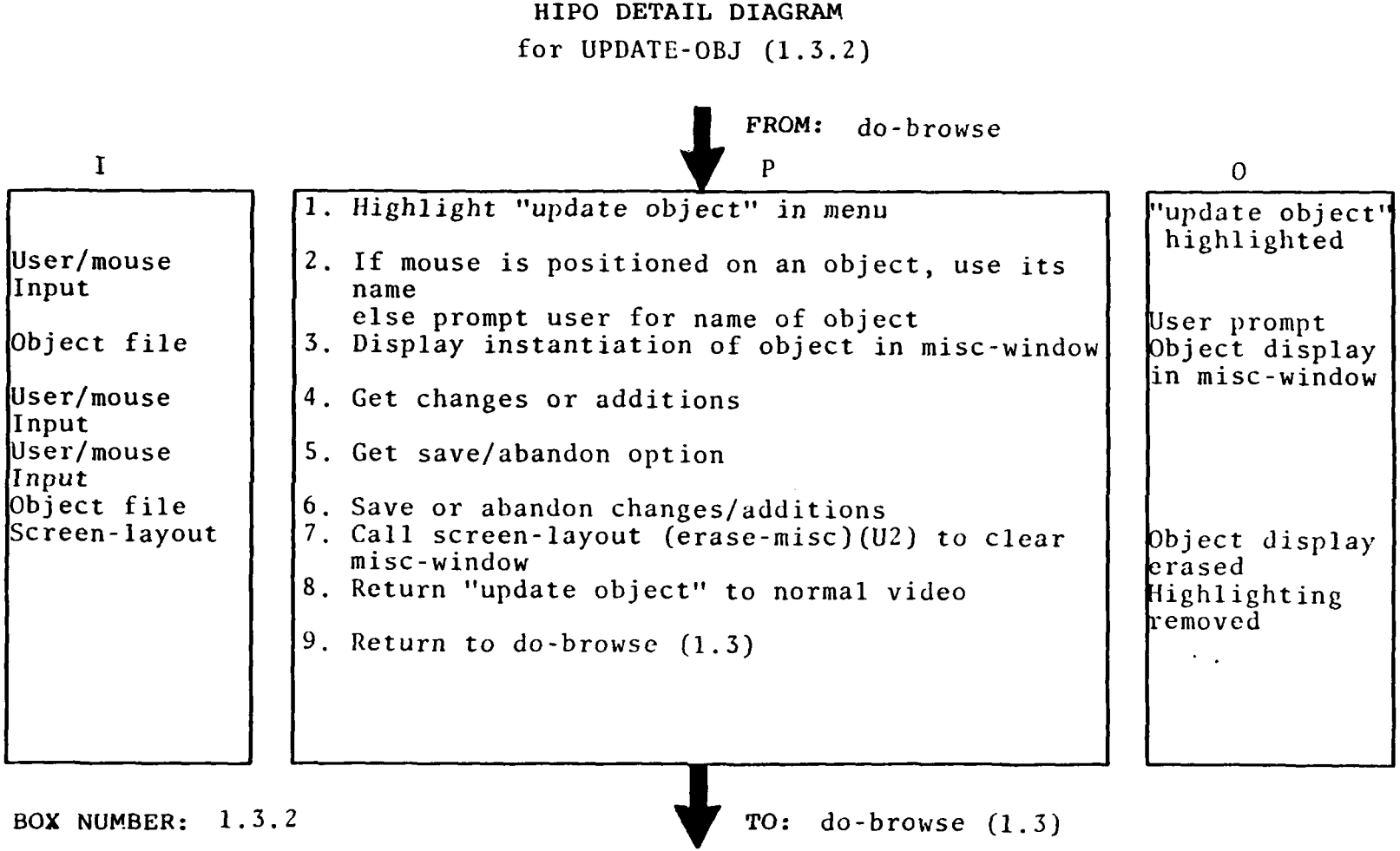


Figure 3.30 Update-Obj

HIPO NOTES

for UPDATE-OBJ (1.3.2)

Step	Note
3-4.	This display and edit function should be done in a standard edit format that the normal user would be familiar with. It is not necessary to have an elaborate editor since this is not the primary function of the program, but it should be effective and understandable.
5.	Save and Abandon should be small icons located in the top corners of the misc-window. By having them displayed there, the user will always be comfortable that he knows how to end the update function at any time (and for any reason for the "abandon" option).

Figure 3.31 Update-Obj (1.3.2)

HIPO OVERVIEW DIAGRAM
for PRINT-OBJ (1.3.3)

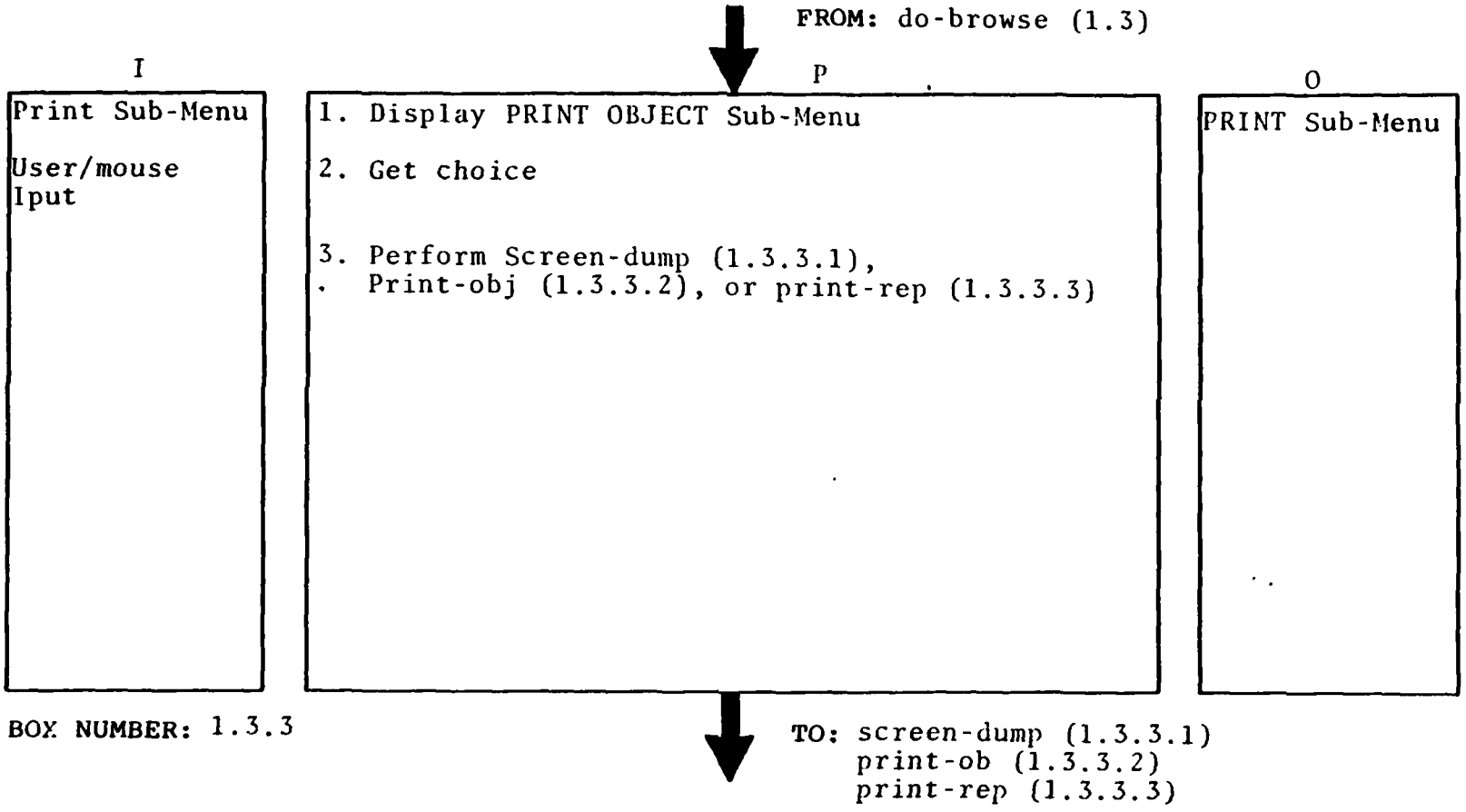


Figure 3.32 Print-Obj

HIPO DETAIL DIAGRAM

for PRINT-OBJ (1.3.3)

FROM: do-browse (1.3)

I	P	O
User/mouse Input	1. Highlight "print object" box in menu 2. Display PRINT OBJECT Sub-Menu on sub-menu line 3. Wait for input 4. If mouse curser is not positioned on one of the menu items, a. Error msg, return to step 3 else 5. Call screen-dump (1.3.3.1), print-ob (1.3.3.2), or print-rep (1.3.3.3) as determined by position of mouse	"print object" highlighted PRINT OBJECT Sub-Menu display error message

BOX NUMBER: 1.3.3

TO: error-msgs (U1)
screen-dump (1.3.3.1)
print-ob (1.3.3.2)
print-rep (1.3.3.3)

Figure 3.33 Print-Obj

HIPO NOTES
for PRINT-OBJ (1.3.3)

Step	Note
2.	This sub-menu is actually composed of just the three choices printed out on the line immediately below the menu box.
4.	Deciding whether the mouse is positioned on one of the items entails noting the curser position at the start of each choice and counting spaces

Figure 3.34 Print-Obj (1.3.3)

HIPO OVERVIEW DIAGRAM
for SCREEN-DUMP (1.3.3.1)

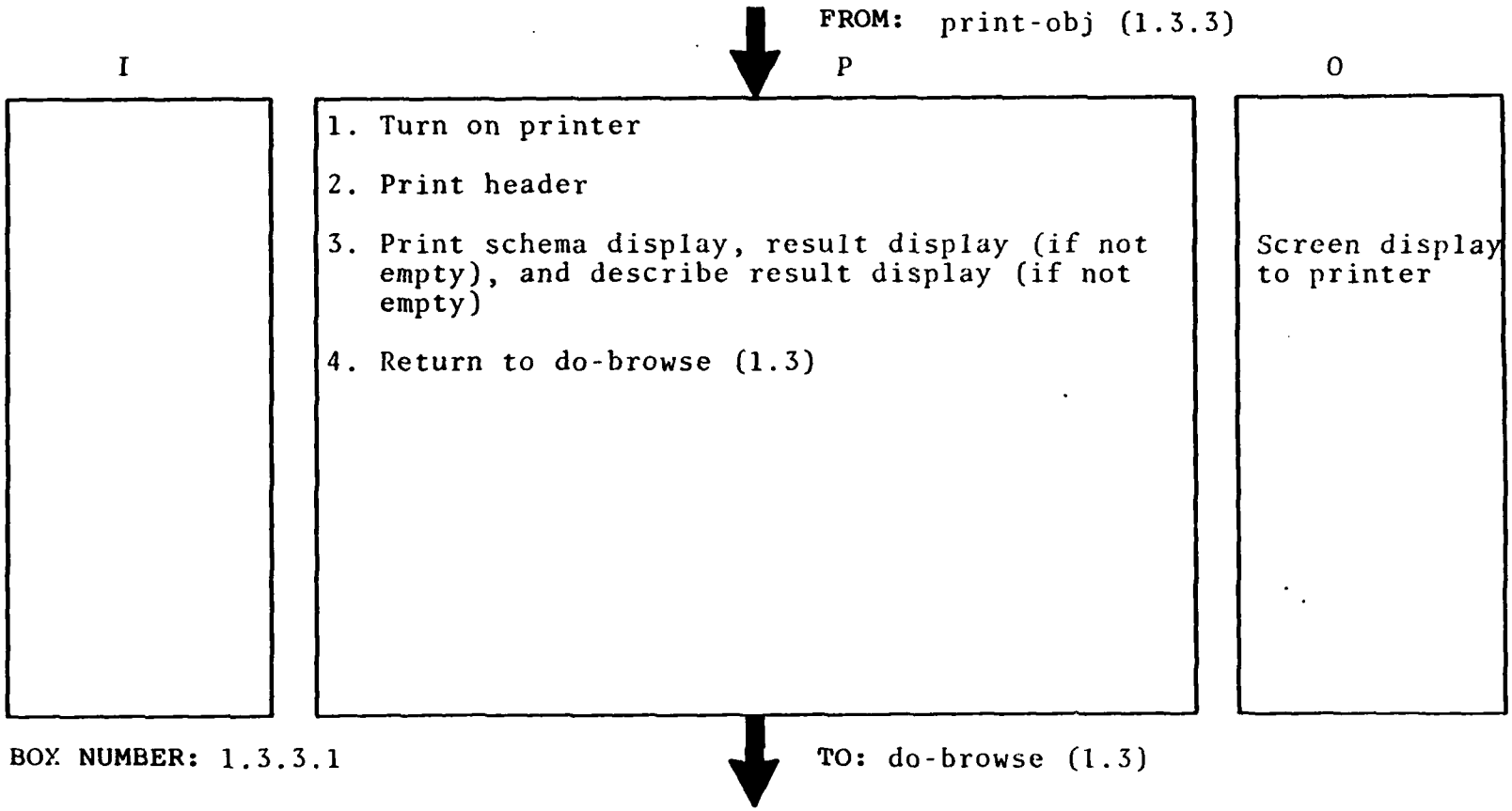


Figure 3.35 Screen-Dump

HIPO DETAIL DIAGRAM
for SCREEN-DUMP (1.3.3.1)

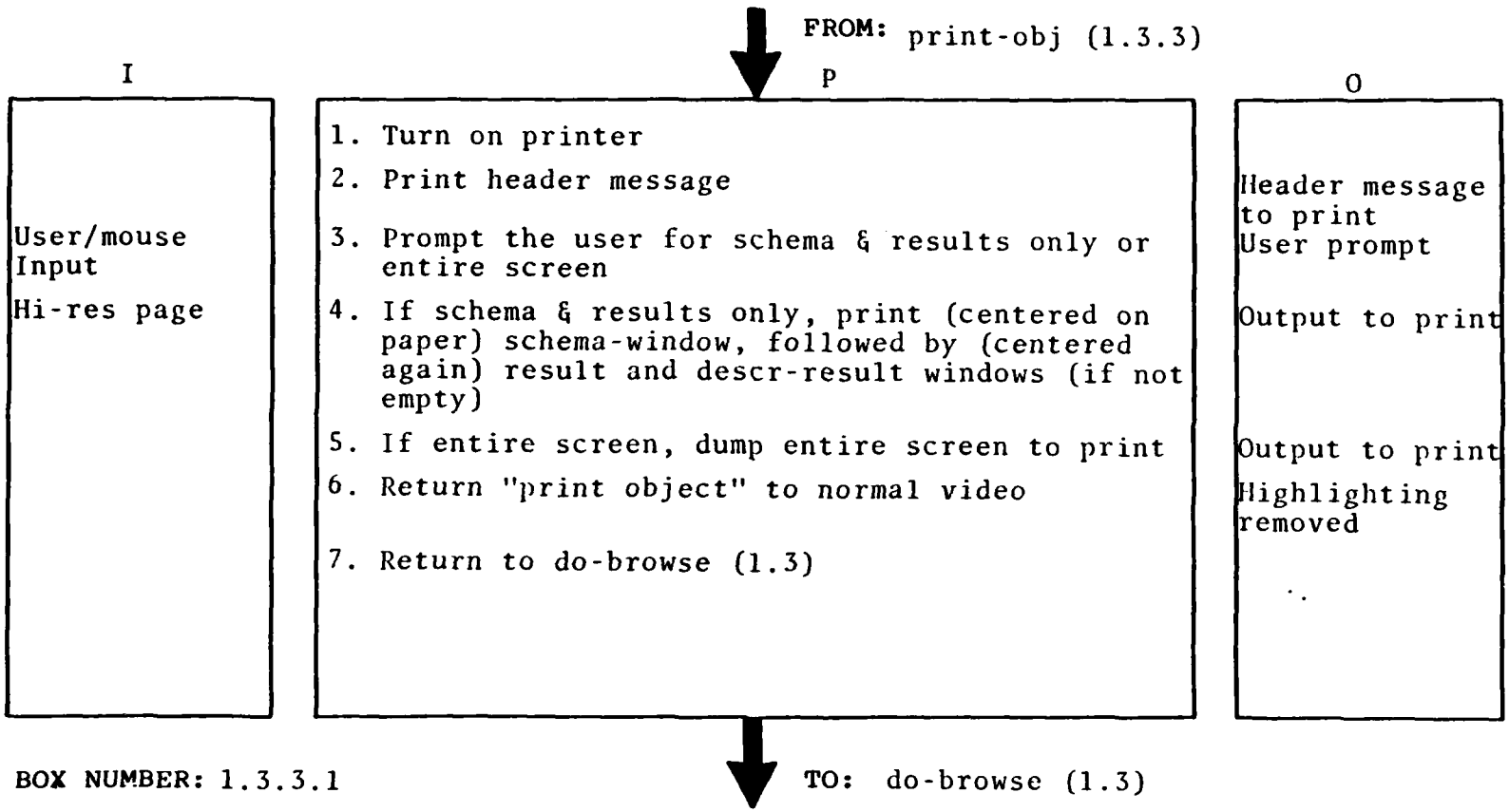


Figure 3.36 Screen-Dump

HIPO NOTES
for SCREEN-DUMP (1.3.3.1)

Step	Note
2.	This header message will probably include things such as database name, current date, and user access level (if used)
4.	It is anticipated that the user will usually want to print what the schema looks like, so we give him this option which will be quicker and use less paper
6.	In this, and the two following PRINT sub-items, highlighting is turned off in a module that does not turn it on. While this is usually not a desirable characteristic, in this case it is justified since the modules and logically connected and it speeds execution

Figure 3.37 Screen-Dump

HIPO OVERVIEW DIAGRAM
for PRINT-OB (1.3.3.2)

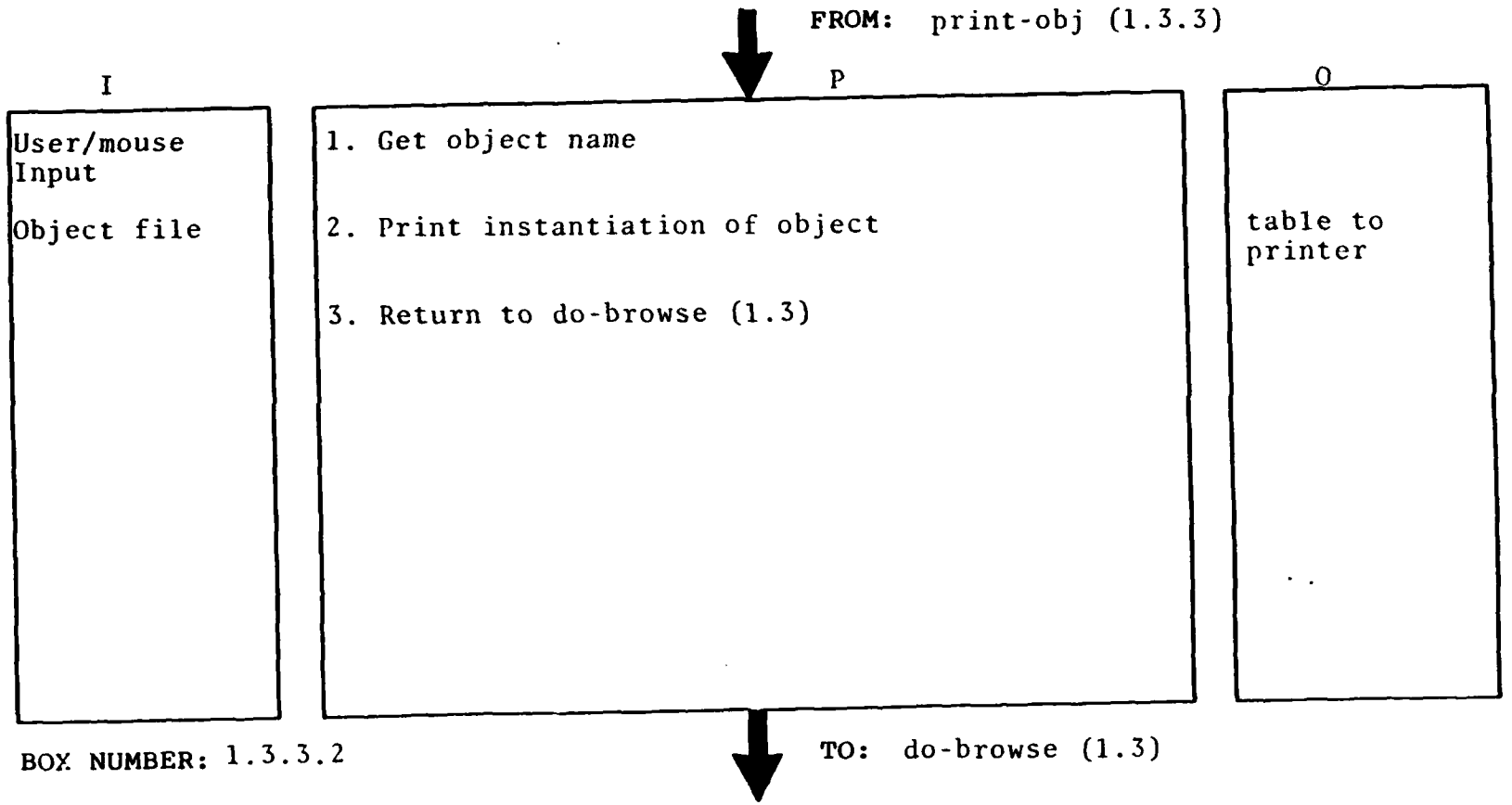


Figure 3.38 Print-Ob

HIPO DETAIL DIAGRAM
for PRINT-OB (1.3.3.2)

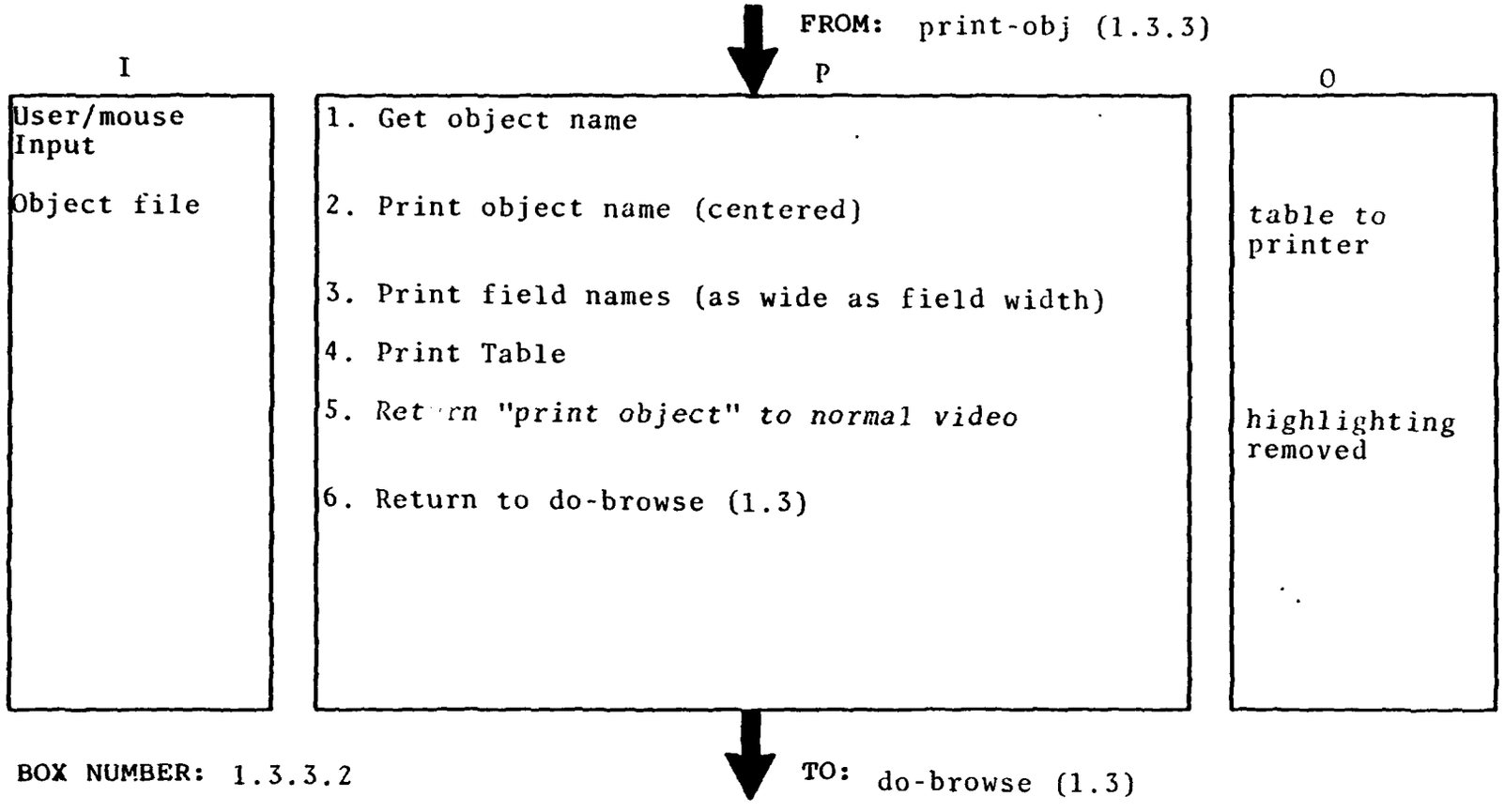


Figure 3.39 Print-Ob

HIPO NOTES

for PRINT-OB (1.3.3.2)

Step	Note
1.	The object name can be typed in, or the user can position the mouse over the desired object and press the "select" button.
3.	The number of field names (and table values) to be printed will be determined by the width of the fields in the records and the width of the platen on the printer. Stop at the last field name (and value set) which will fit completely.

Figure 3.40 Print-Ob

HIPO OVERVIEW DIAGRAM
for PRINT-REP (1.3.3.3)

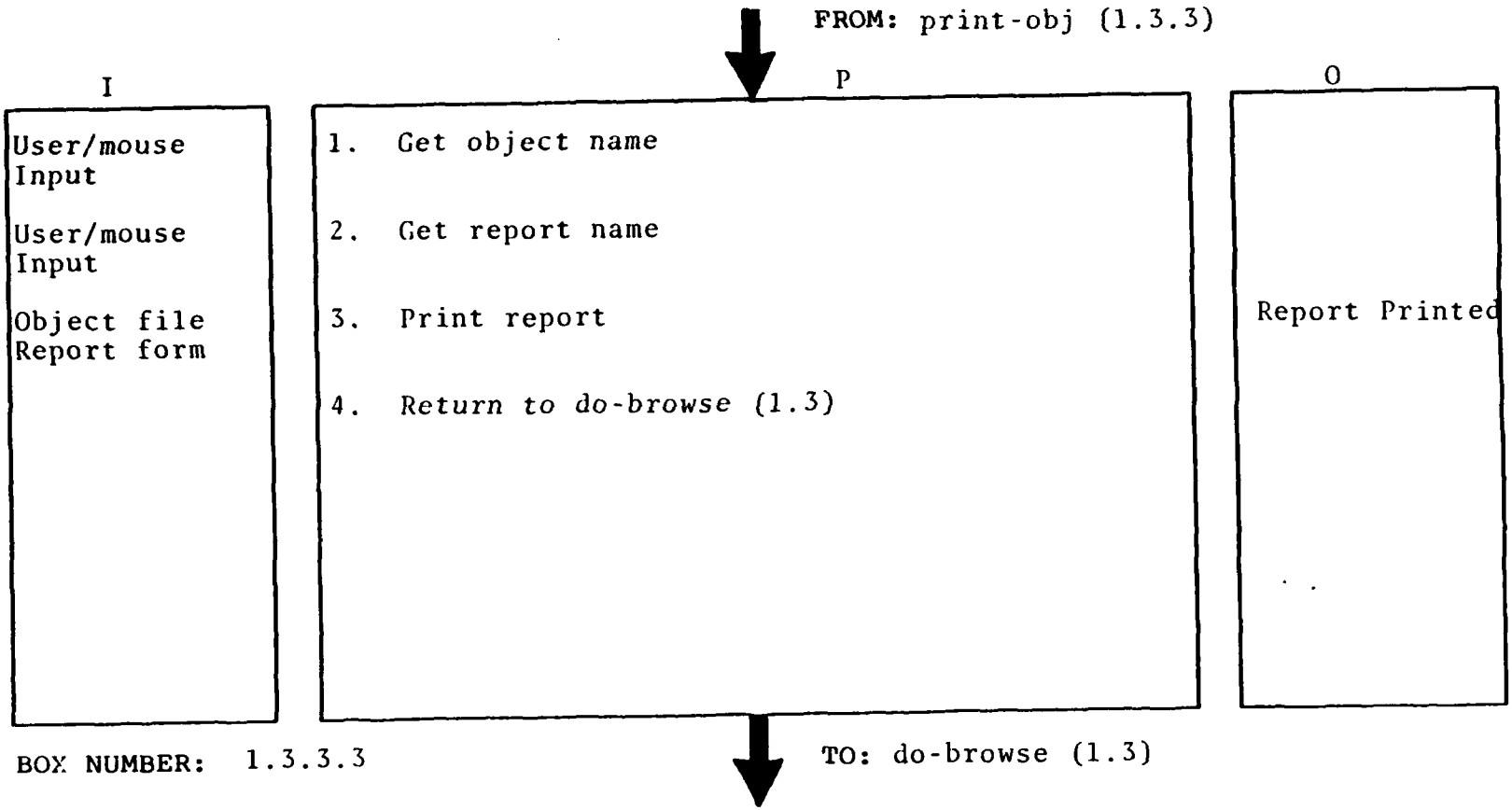


Figure 3.41 Print-Rep

HIPO DETAIL DIAGRAM
for PRINT-REP (1.3.3.3)

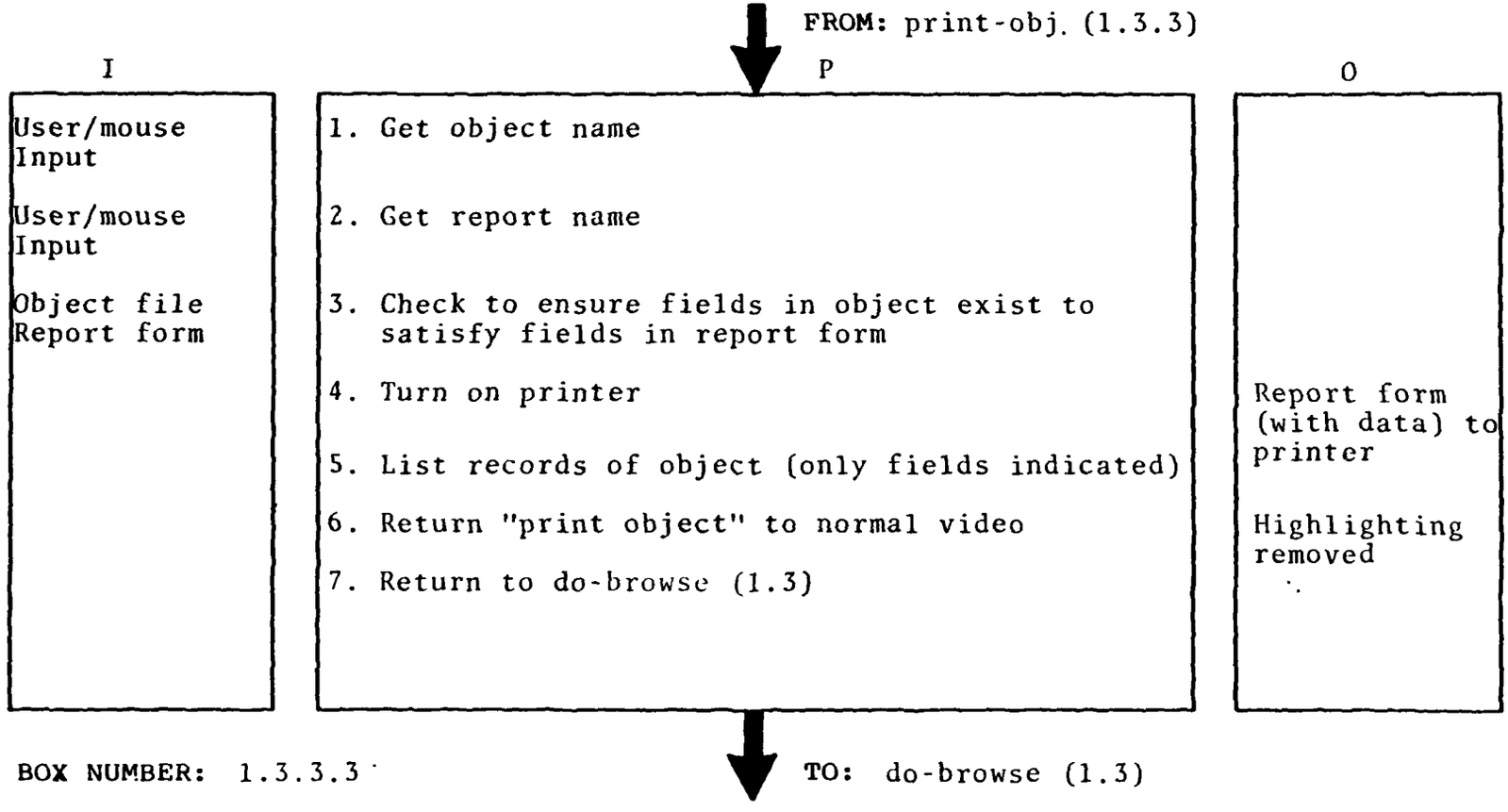


Figure 3.42 Print-Rep

HIPO NOTES
for PRINT-REP (1.3.3.3)

Step	Note
3.	All fields in report form must exist in the object. Otherwise, you cannot know where the inconsistency is.
5.	This will be done as indicated in the report form directives (i.e. the header, field names, and widths will be specified)

Figure 3.43 Print-Rep

HIPO OVERVIEW DIAGRAM

for QUERY (1.3.4 prep)

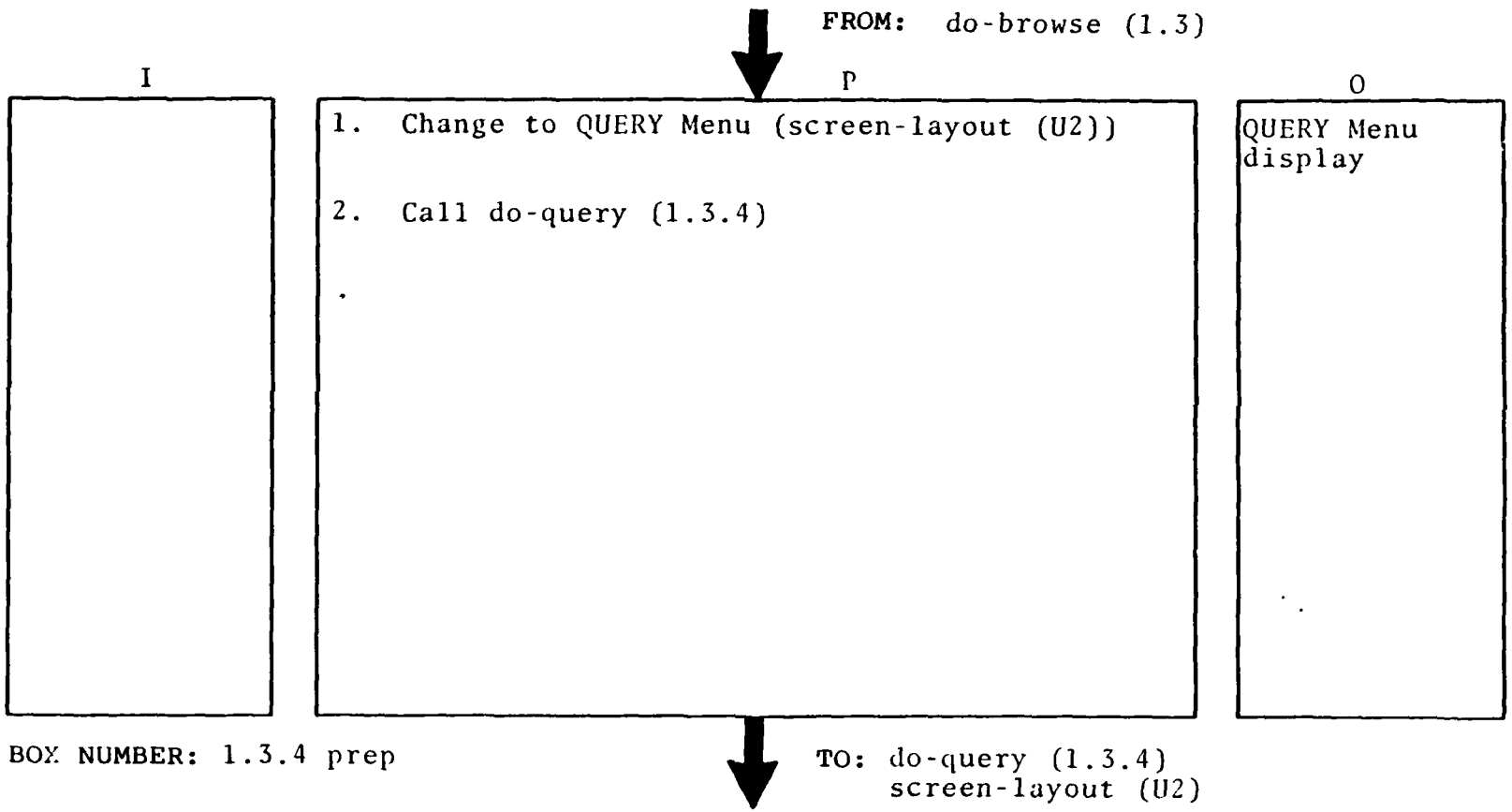
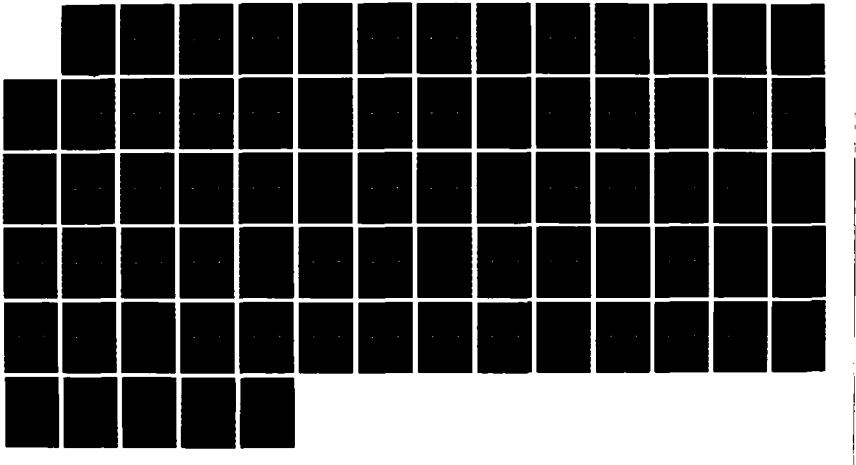
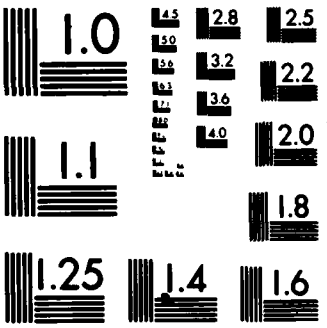


Figure 3.44 Query

AD-A171 393 DESIGN OF A GRAPHICS USER INTERFACE FOR A DATABASE MANAGEMENT SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J K ADCOCK JUN 86 2/2

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

HIPO DETAIL DIAGRAM
for QUERY (1.3.4 prep)

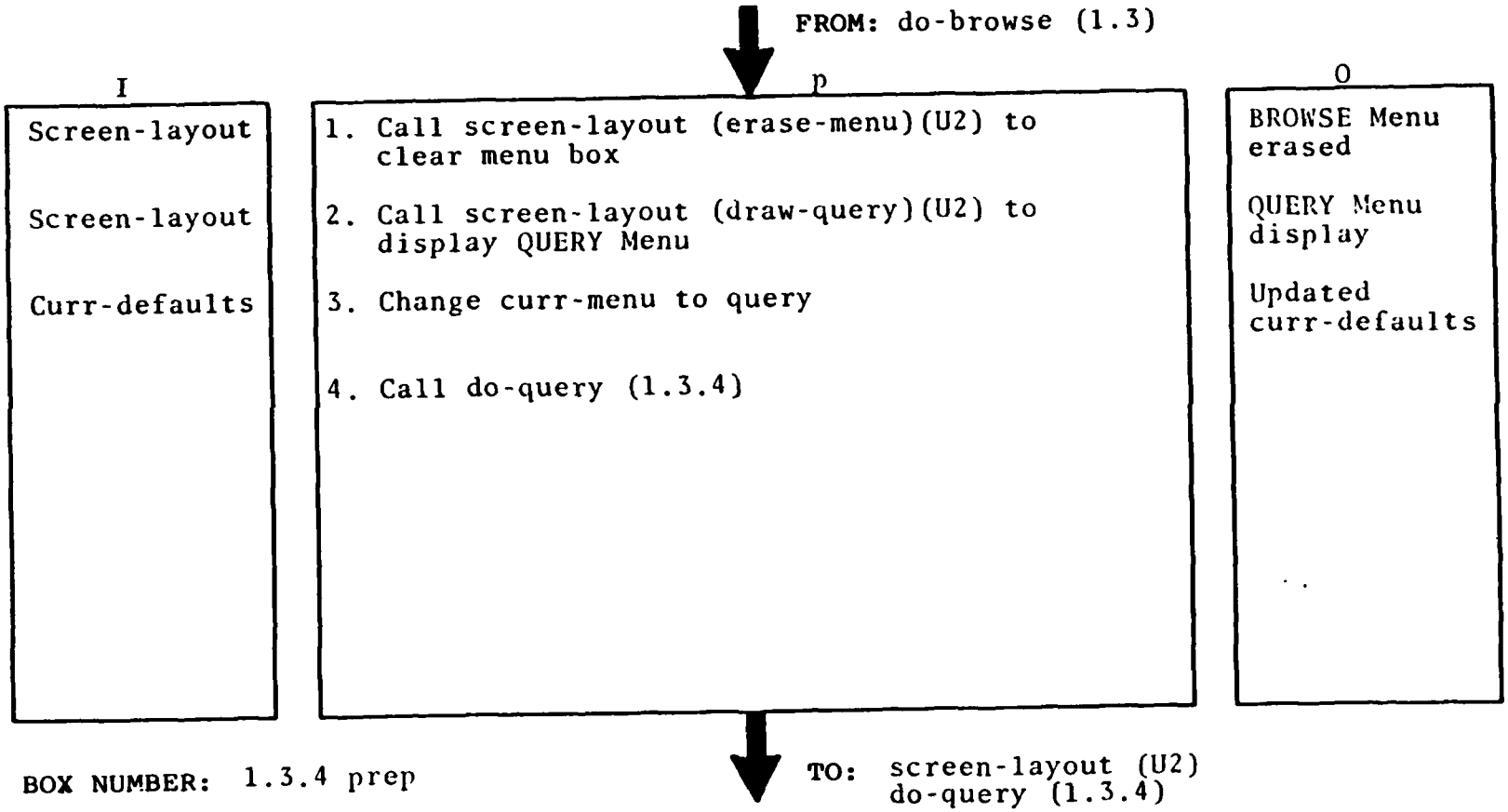


Figure 3.45 Query

HIPO OVERVIEW DIAGRAM
for DO-QUERY (1.3.4)

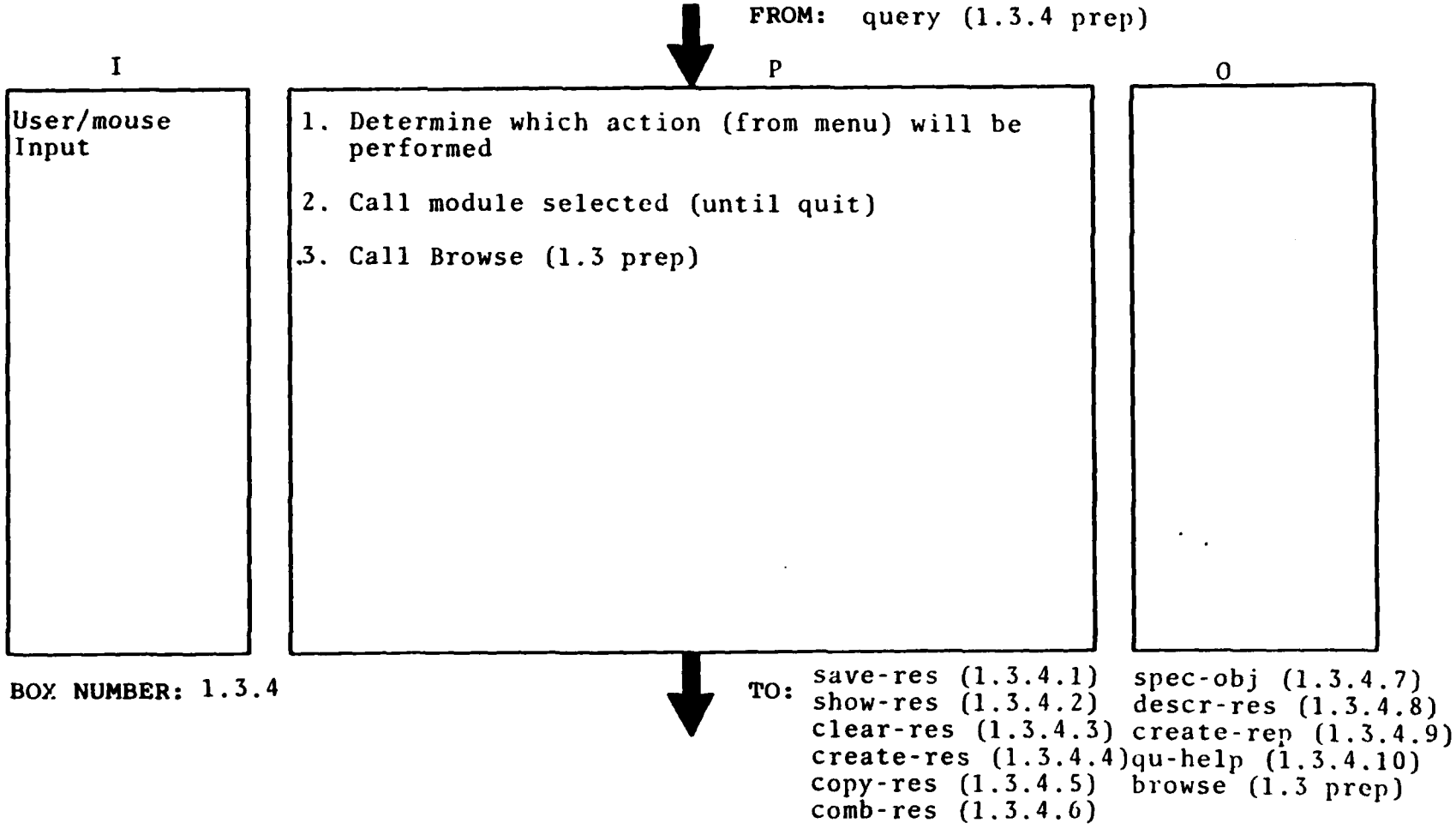


Figure 3.46 Do-Query

HIPO DETAIL DIAGRAM
for DO-QUERY (1.3.4)

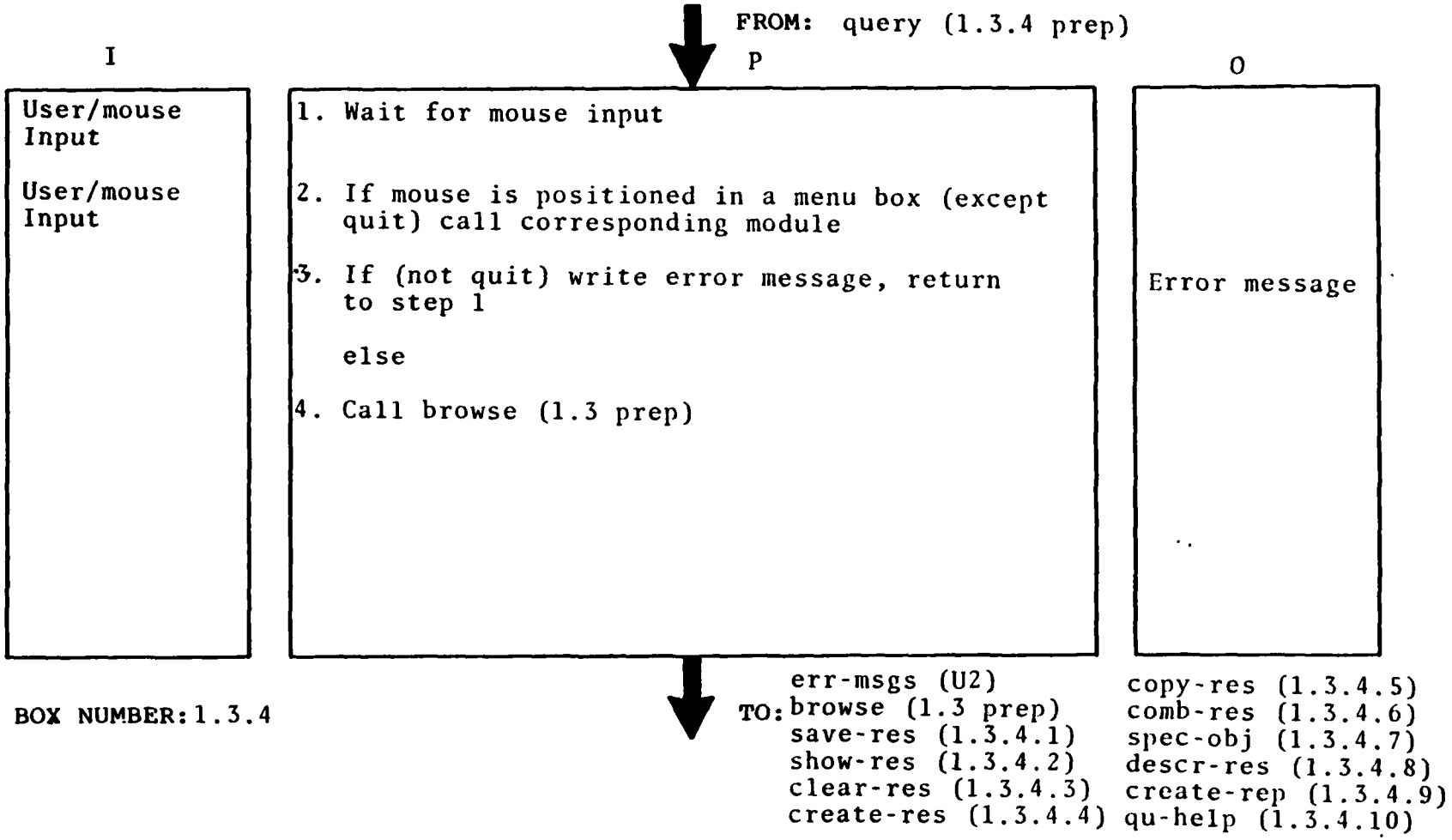


Figure 3.47 Do-Query.

HIPO NOTES
for DO-QUERY (1.3.4)

Step	Note
4.	The call at the end of this module is different than the one at the end of do-browse (1.3). The reason is that do-browse has a simple lead-in module that can be used (recall that the lead-in module to do-admin is start (0) which does more than simply change the menu display and the curr-menu variable).

Figure 3.48 Do-Query.

HIPO OVERVIEW DIAGRAM
for EXPAND-OBJ (1.3.5)

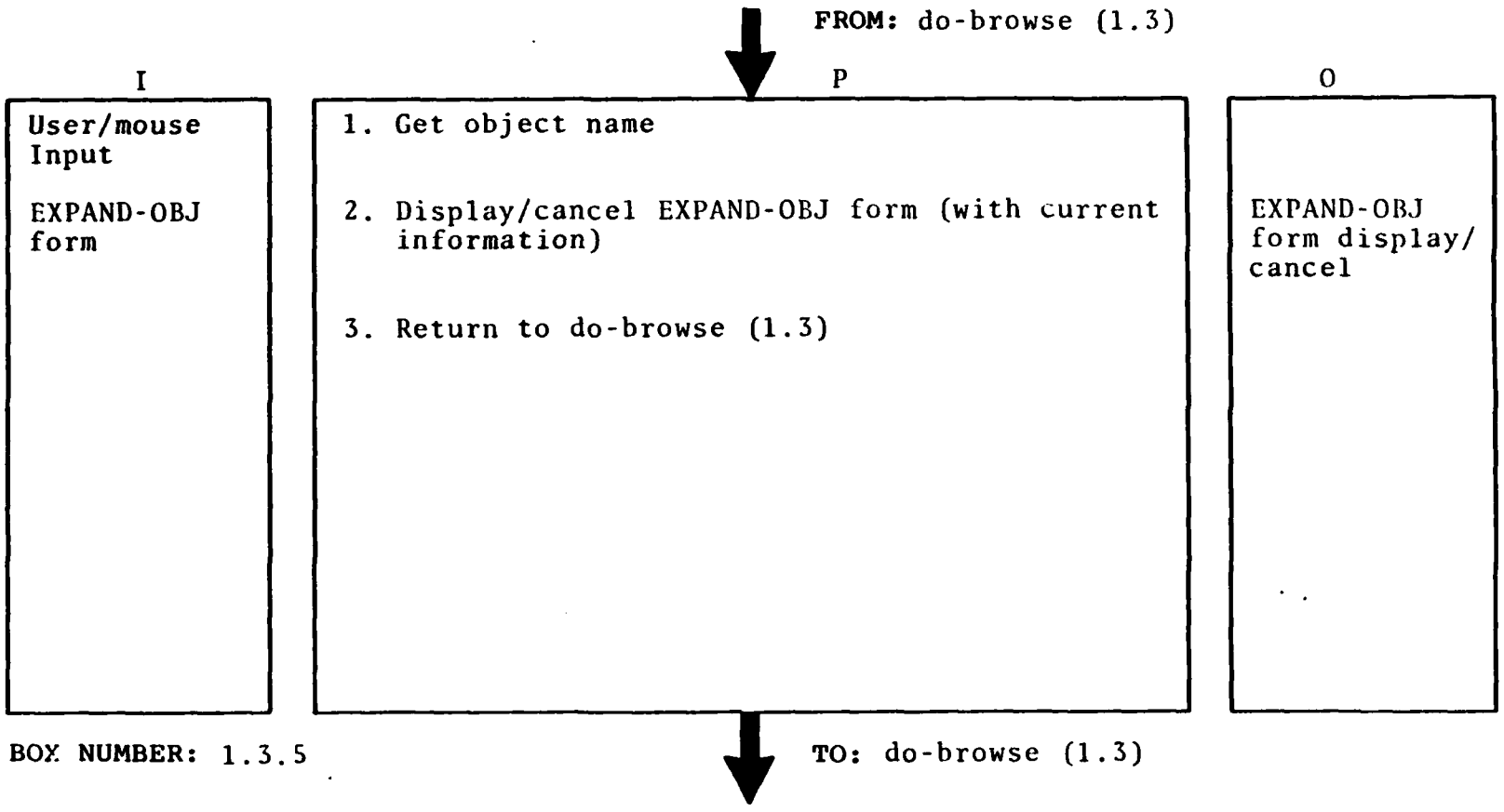


Figure 3.49 Expand-Obj.

HIPO DETAIL DIAGRAM

for EXPAND-OBJ (1.3.5)

FROM: do-browse (1.3)

I	P	O
User/mouse Input	1. Highlight "expand object" box in menu	"expand object" highlighted
User/mouse Input	2. If mouse is positioned on an object, use its name else prompt user for name of object	User prompt
Curr-defaults	3. Check name against curr-exp-obj list to see if is already being expanded	
	4. If object is already being expanded, a. Delete name from curr-exp-obj	Updated -curr defaults
Screen-layout	b. Call screen-layout (draw-schema)(U2) to redraw schema	-schema display
	c. Return "expand object" to normal video	Highlighting
	d. Return to do-browse (1.3) else	removed
Curr-defaults Screen-layout	5. Add name to curr-exp-obj 6. Call screen-layout (draw-schema)(U2) to redraw schema with this object being expanded via a exp-obj-form (filled in with appropriate info)	Updated -curr defaults -schema dis-play
	7. Return "expand object" to normal video	Highlighting removed
	8. Return to do-browse (1.3)	

BOX NUMBER: 1.3.5

TO: do-browse (1.3)
screen-display (U2)

Figure 3.50 Expand-Obj.

HIPO NOTES
for EXPAND-OBJ (1.3.5)

Step	Note
	This module should be implemented in a manner almost identical to that used for descr-obj. They are similar functions, so they should look very similar. The only difference between the two modules is the actual layout of the forms (small windows) that will overwrite the schema during screen-layout (draw-schema).

Figure 3.51 Expand-Obj.

HIPO OVERVIEW DIAGRAM

for LIST-MEM (1.3.6)

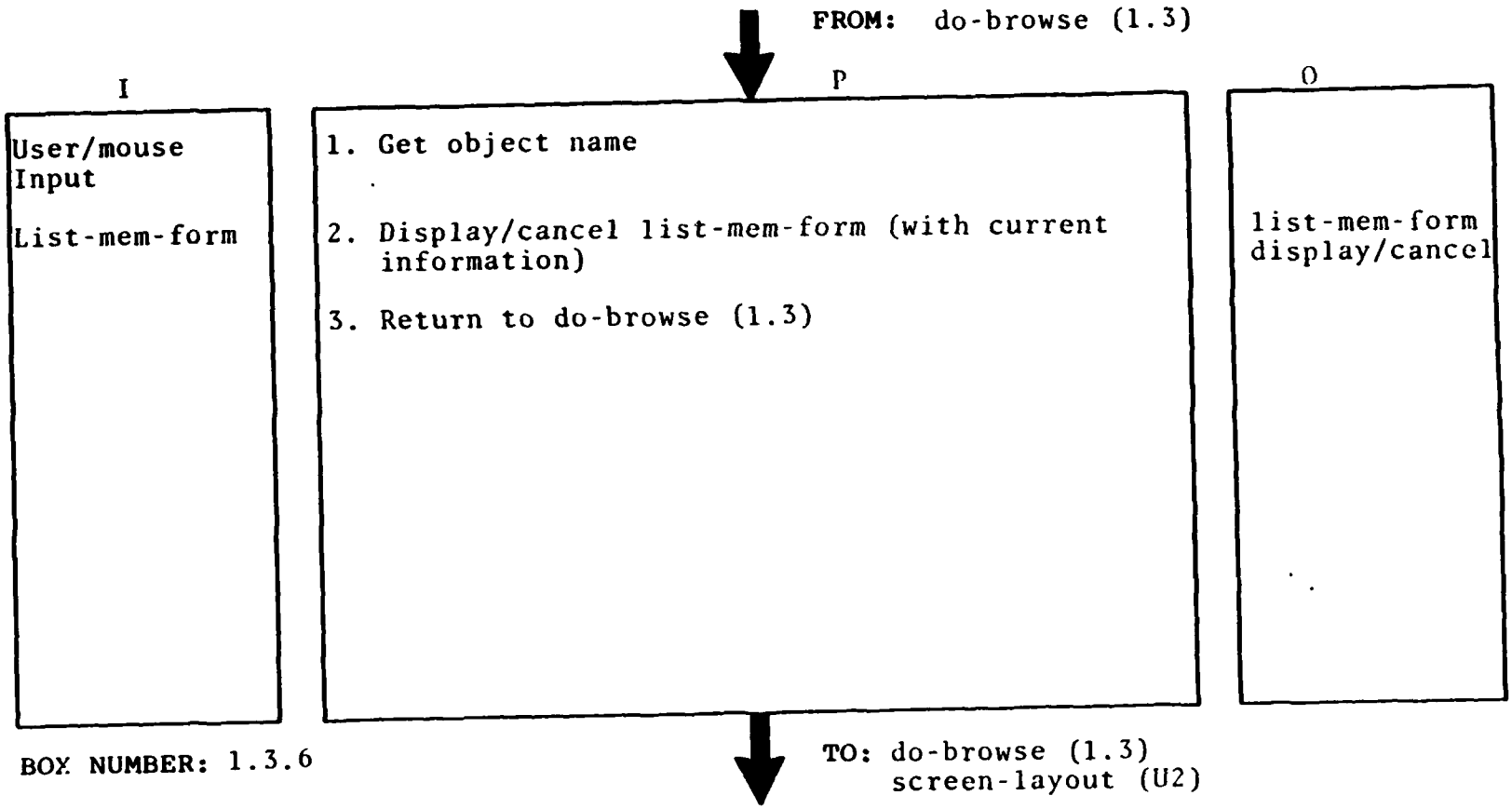


Figure 3.52 List-Mem.

HIPO DETAIL DIAGRAM

for LIST-MEM (1.3.6)

FROM: do-browse (1.3)

I	P	O
	1. Highlight "list members" box in menu	"list members"
User/mouse Input	2. If mouse is positioned on an object, use its name	highlighted
User/mouse Input	3. else prompt user for name of object	User prompt
Curr-defaults	3. Check name against curr-lst-mem list to see if it is already being listed	
	4. If object is already being listed,	Updated
Curr-defaults	a. Delete name from curr-lst-mem	Updated -curr-defaults
Screen-layout	b. Call screen-layout (draw-schema)(U2) to redraw schema	-schema display
	c. Return "list member" to normal video	Highlighting
	d. Return to do-browse (1.3)	removed
	else	
Curr-defaults	5. Add name to curr-lst-mem	Updated
Screen-layout	6. Call screen-layout (draw-schema)(U2) to redraw schema with this object being expanded via a lst-mem-form (filled in with appropriate info)	-curr-defaults -schema display
	7. Return "list member"to normal video	lighlighting
	8. Return to do-browse (1.3)	removed

BOX NUMBER: 1.3.6

TO: do-browse (1.3)
screen-layout (U2)

Figure 3.53 List-Mem.

HIPO NOTES

for LIST-MEM (1.3.6)

Step	Note
	This module is the third in the series of object "expansion" options. It should be implemented the same way as the other two (descr-obj and expand-obj). The use of "forms" in all three of these cases should simplify their implementation and enable the implementor to achieve conformity and regularity in their use.

Figure 3.54 List-Mem.

HIPO OVERVIEW DIAGRAM

for CENTER-OBJ (1.3.7)

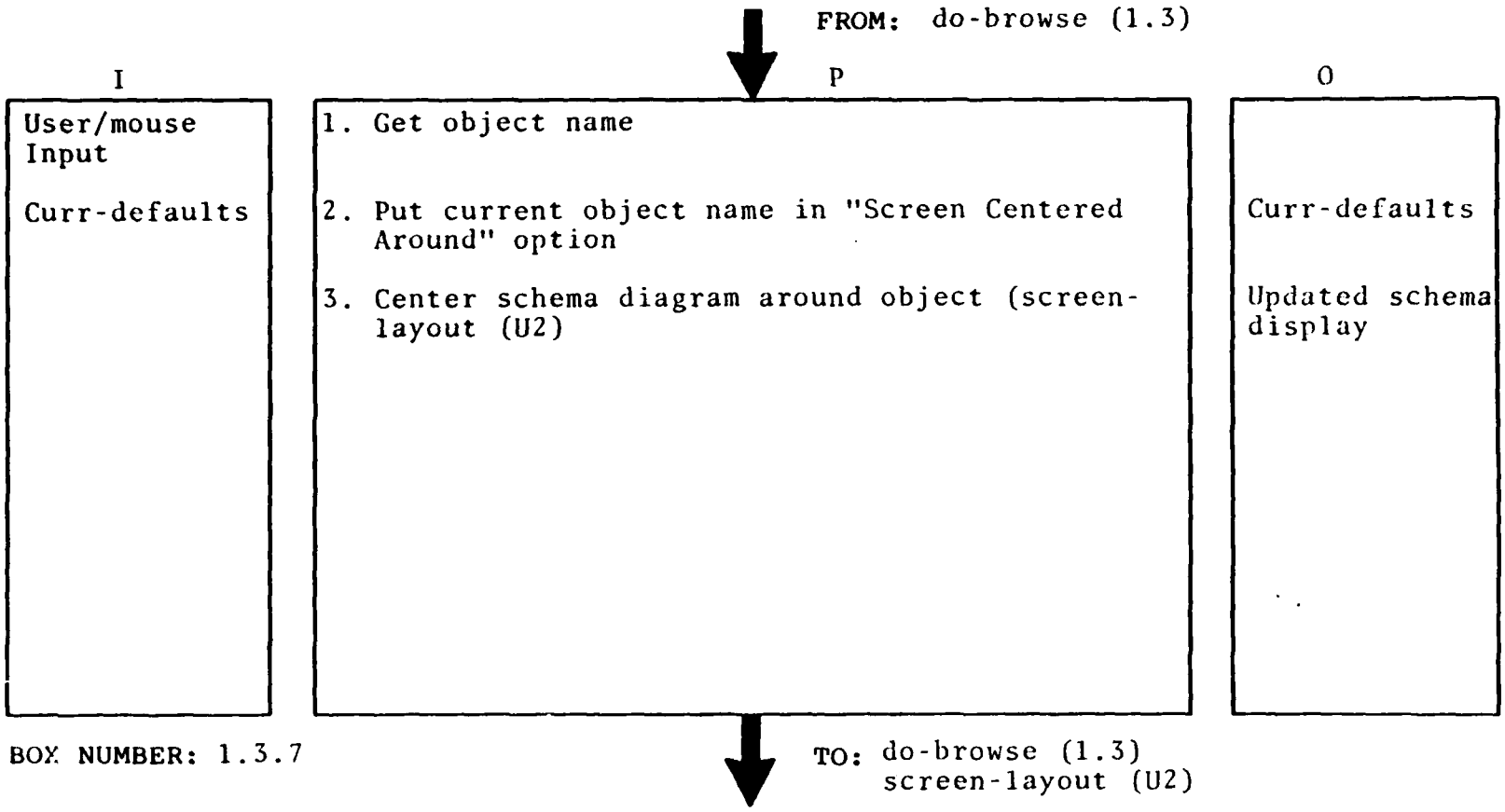


Figure 3.55 Center-Obj.

HIPO DETAIL DIAGRAM
for CENTER-OBJ (1.3.7)

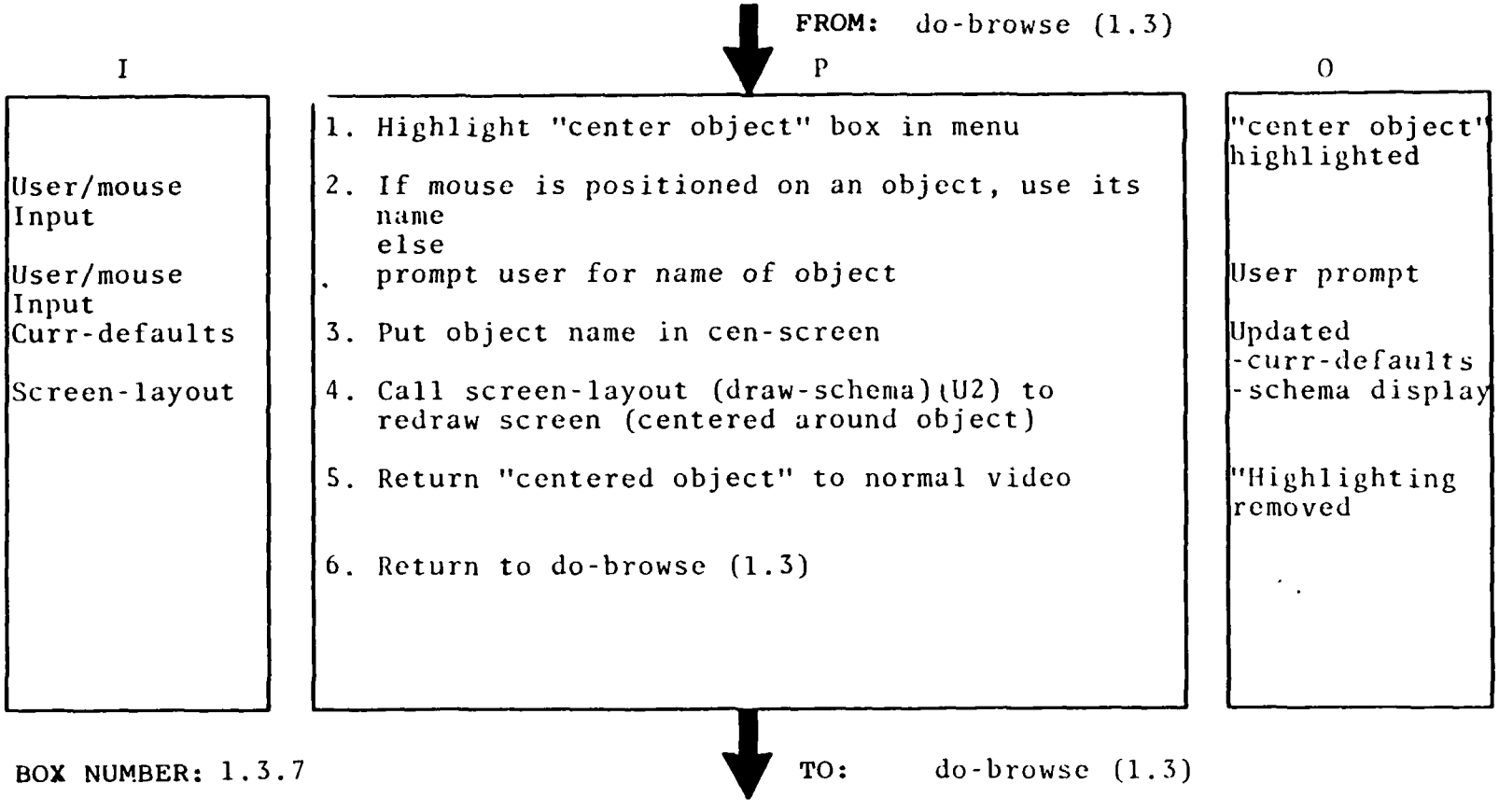


Figure 3.56 Center-Obj.

HIPO NOTES
for CENTER-OBJ (1.3.7)

Step	Note
	This module is fairly straightforward except for the case in which the user wants to get out of having the schema display centered around any object. There are two ways to accomplish this: (1) the user selects "center object" without having the mouse positioned on any object, and presses "carriage return" when prompted for an object name or (2) the user selects "center object" without having the mouse positioned on any object, and types a symbolic name (such as "home" or "none") when prompted for an object name. The advantage to option (1) is that it is faster and easier. The advantage of option (2) is that it precludes the user from "accidentally" selecting the "none" option. However, he would need to be reminded of the symbolic name for "none", because we certainly do not want to force him to memorize an arbitrary command code. The choice between the two options is really one of style, since either would accomplish the task.

Figure 3.57 Center-Obj.

HIPO OVERVIEW DIAGRAM

for BR-HELP (1.3.8)

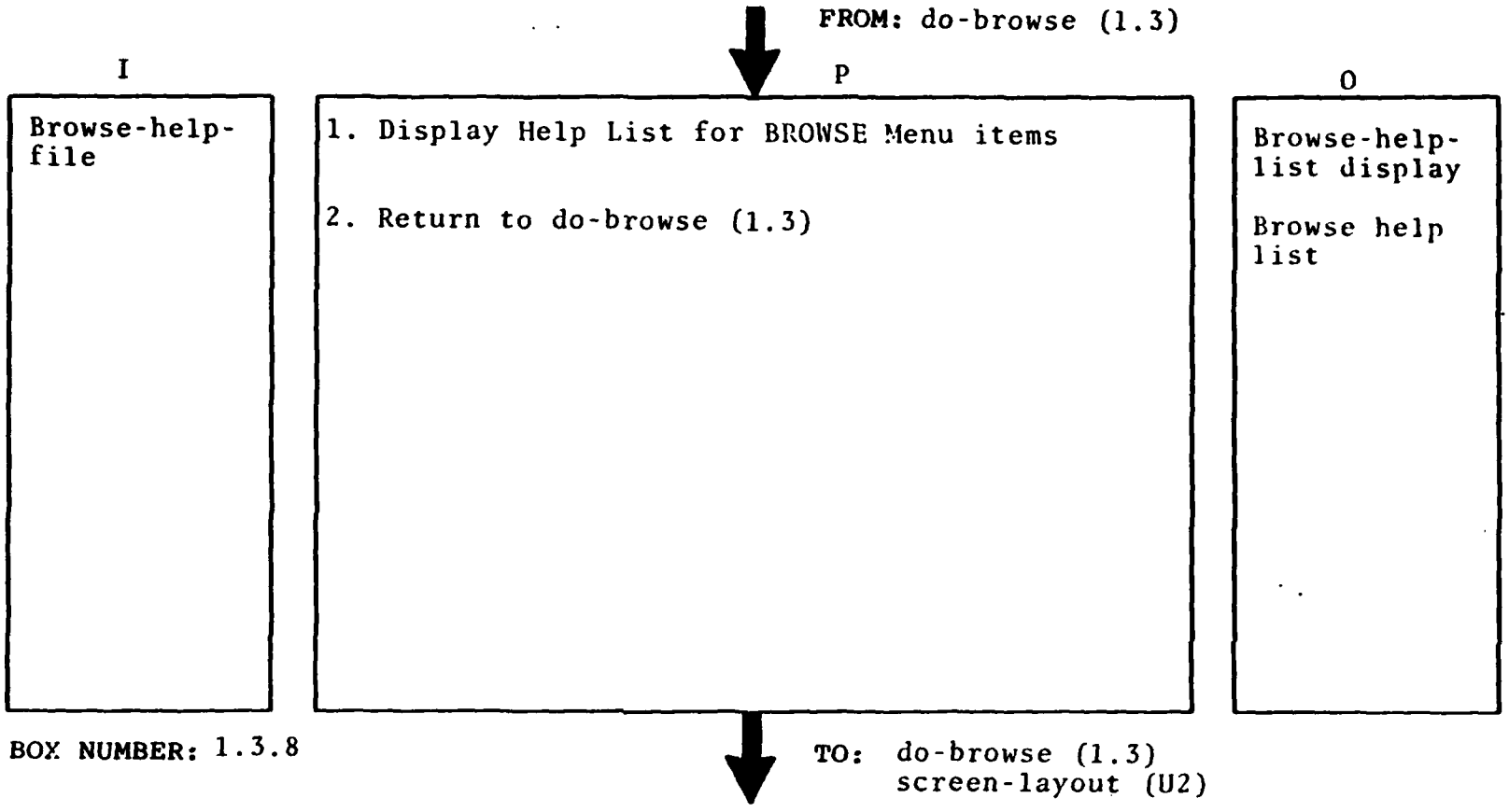


Figure 3.58 Br-Help.

HIPO DETAIL DIAGRAM

for BR-HELP (1.3.8)

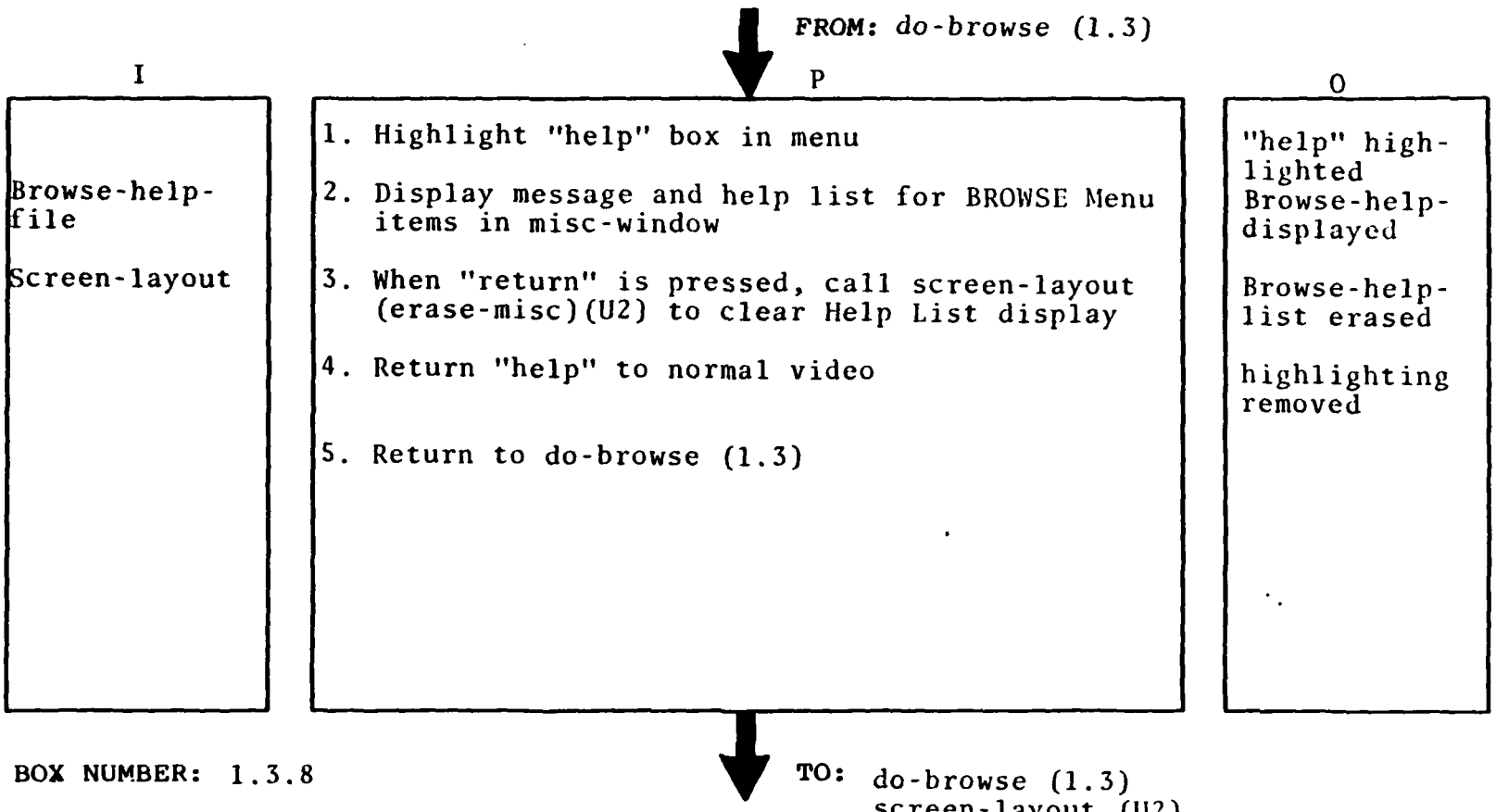


Figure 3.59 Br-Help.

HIPO OVERVIEW DIAGRAM
for SAVE-RES (1.3.4.1)

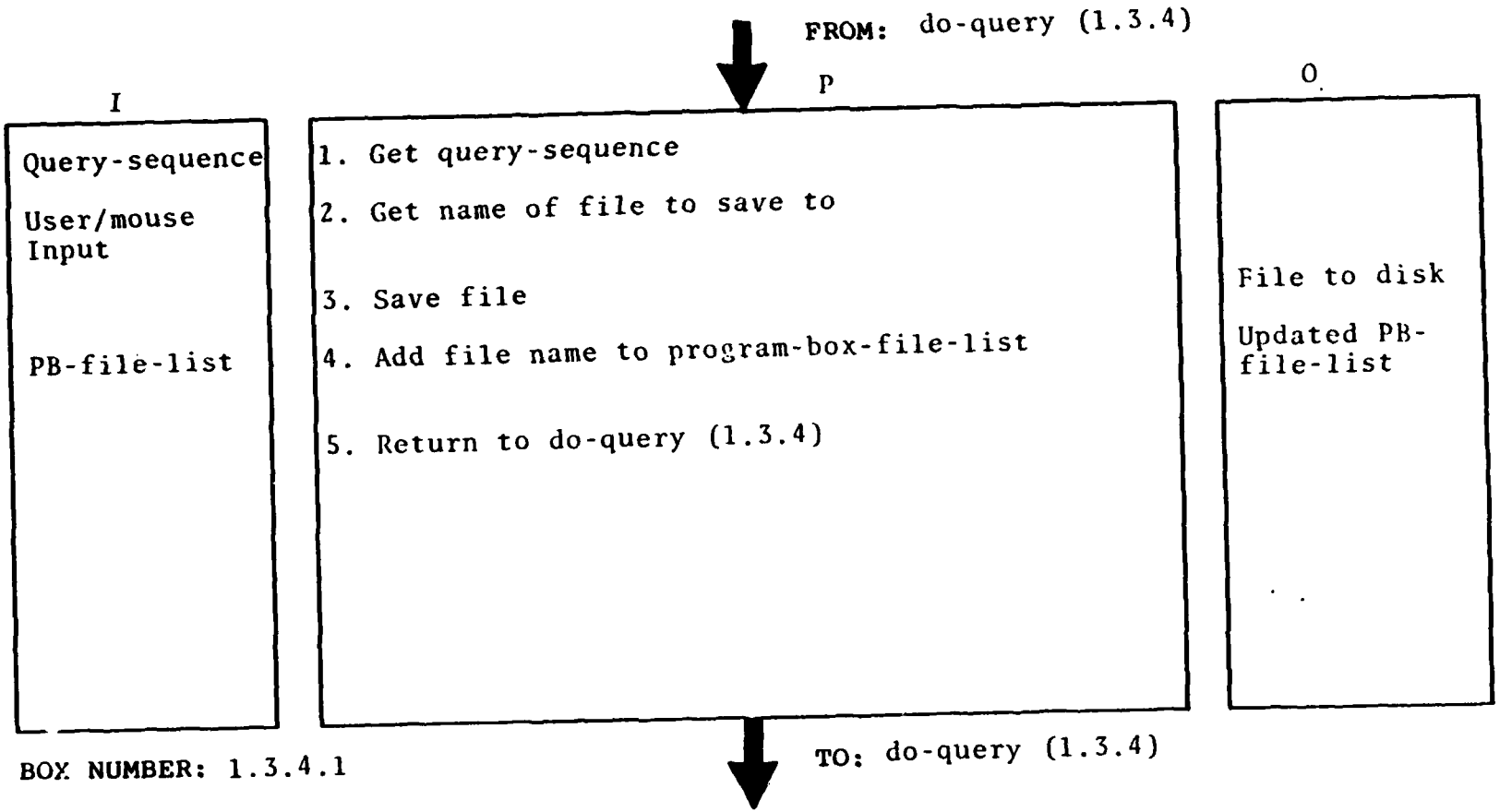


Figure 3.60 Save-Res.

HIPO DETAIL DIAGRAM
for SAVE-RES (1.3.4.1)

FROM: do-query (1.3.4)

I	P	O
	1. Highlight "save result" box in menu	"save result" highlighted
User/mouse Input	2. If there is only one result formed, use it else if mouse is positioned on a result box, use else prompt user for name of result	User prompt
User/mouse Input	3. Prompt user for name of file to save to	User prompt
PB-file-list	4. Check name against those in program-box-file-list	
User/mouse Input	5. If name already exists, prompt user for "overwrite" or "select another name"	User prompt
PB-file-list	6. Add file name to program-box-file-list	Updated PB-file-list
	7. Return "save result" to normal video	Highlighting removed
	8. Return to do-query (1.3.4)	

BOX NUMBER: 1.3.4.1

TO: do-query (1.3.4)

Figure 3.61 Save-Res.

HIPO NOTES
for SAVE-RES (1.3.4.1)

Step	Note
2.	The query sequence to be saved here is one of the sequences the user created himself earlier by selecting "create result". It is important to note here that the result itself may not have been created at all, and all we are dealing with here is the sequence that creates the result.
4.	This PB-file-list is a list of file names that contain certain query sequences, repetitive actions, or command "programs" that the user or database administrator has created. These are included to make the program easier/faster to use for the experienced user, and perhaps easier to learn for the novice user (in the case where the database administrator creates command programs).

Figure 3.63 Save-Res.

HIPO OVERVIEW DIAGRAM
for SHOW-RES (1.3.4.2)

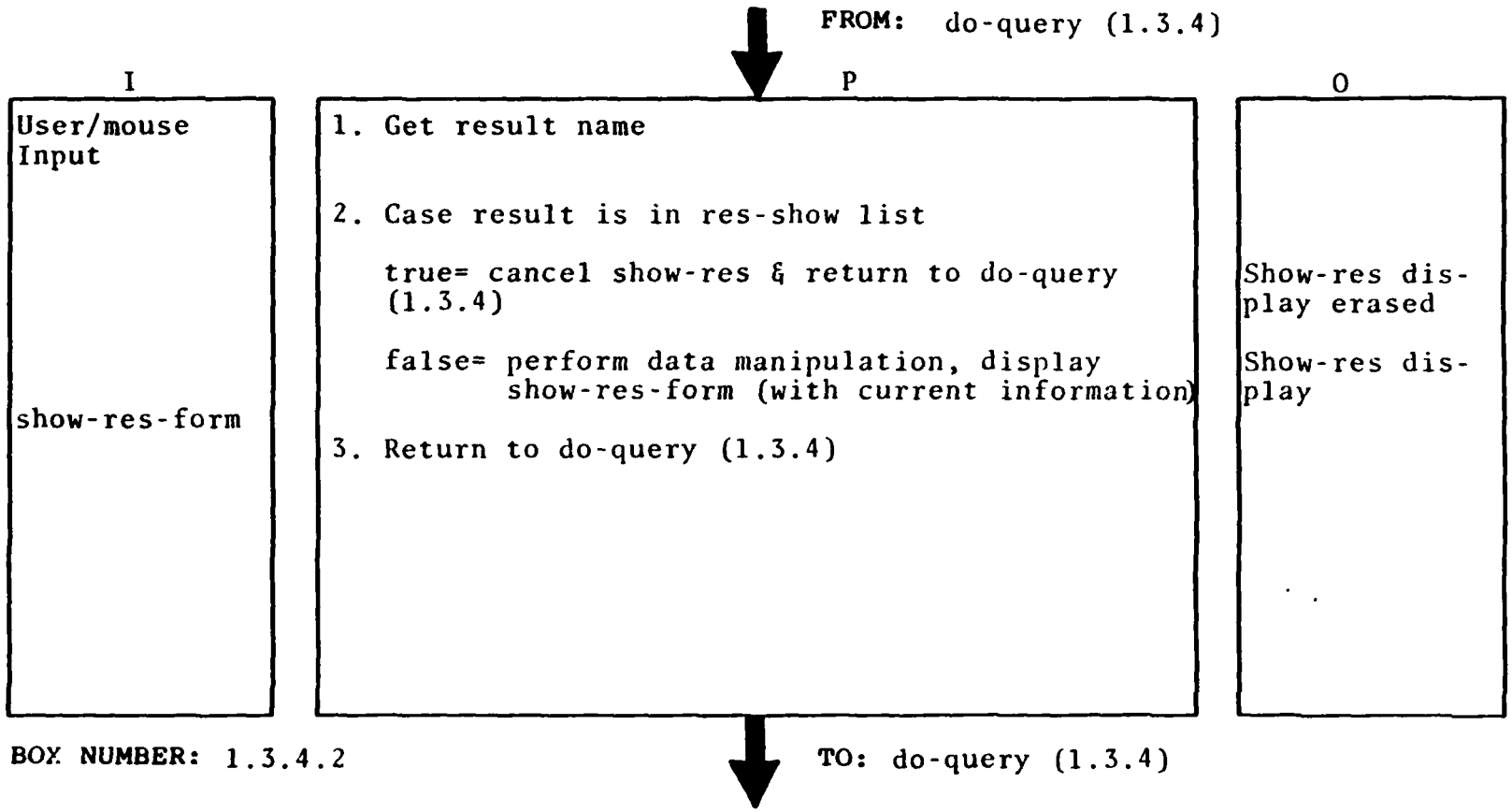


Figure 3.63 Show-Res.

HIPO DETAIL DIAGRAM

for SHOW-RES (1.3.4.2)

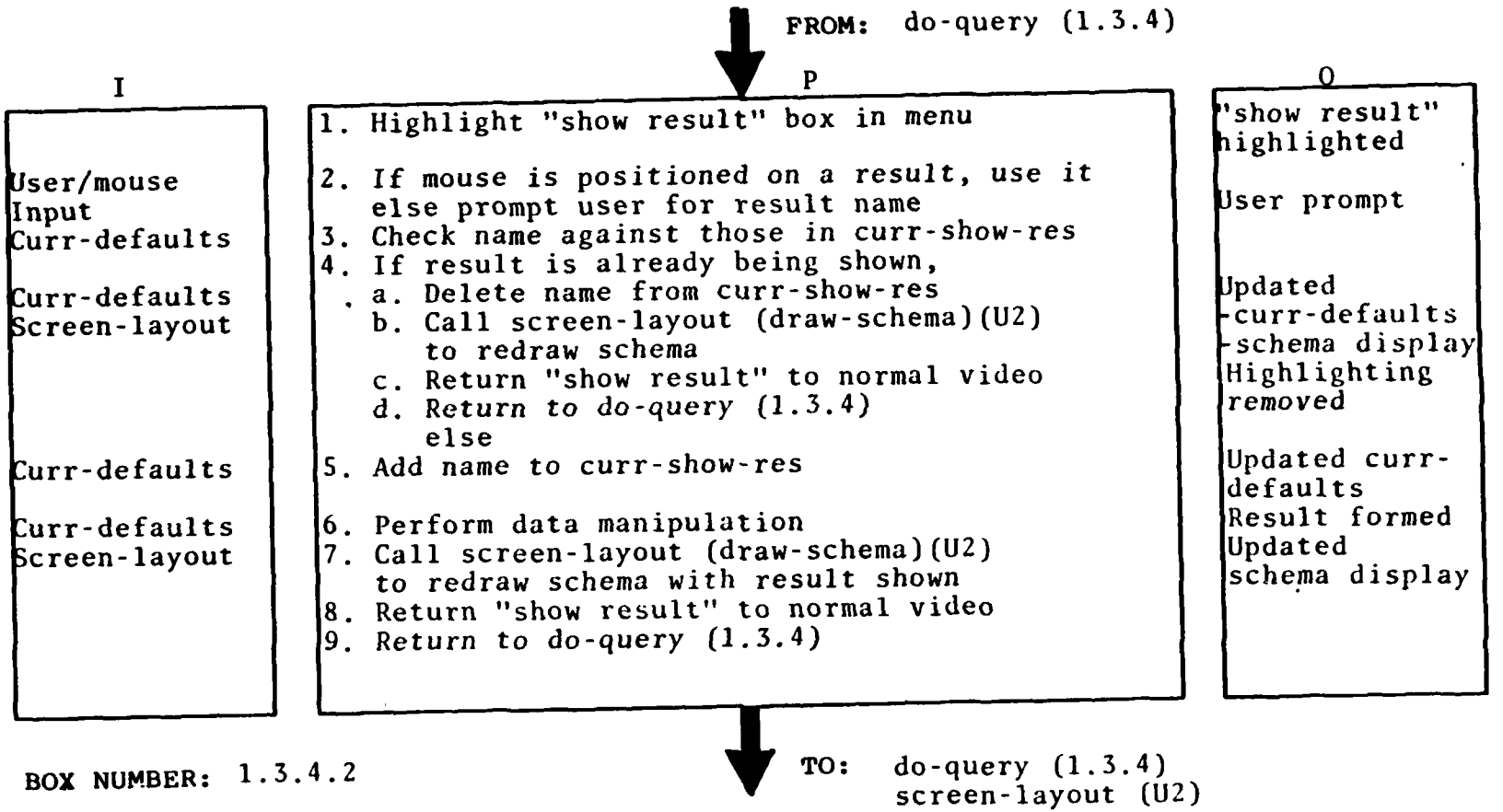


Figure 3.64 Show-Res.

HIPO NOTES
for SHOW-RES (1.3.4.2)

Step	Note
6.	This instruction may prove to be the most time-consuming of the entire program. Until this time, the result has not actually been formed: only the query sequence has been saved. Now it is necessary to actually execute that sequence.
7.	The show-res-form to be used here will be essentially identical to the one used for list members. The difference is that it is to be displayed below the result box rather than overwriting it as we did with list-mem. The reason we do not want to lose the information content provided by its shading. (Recall that the result box and all objects used to generate the result are identically shaded.)

Figure 3.65 Show-Res.

HIPO OVERVIEW DIAGRAM
for CLEAR-RES (1.3.4.3)

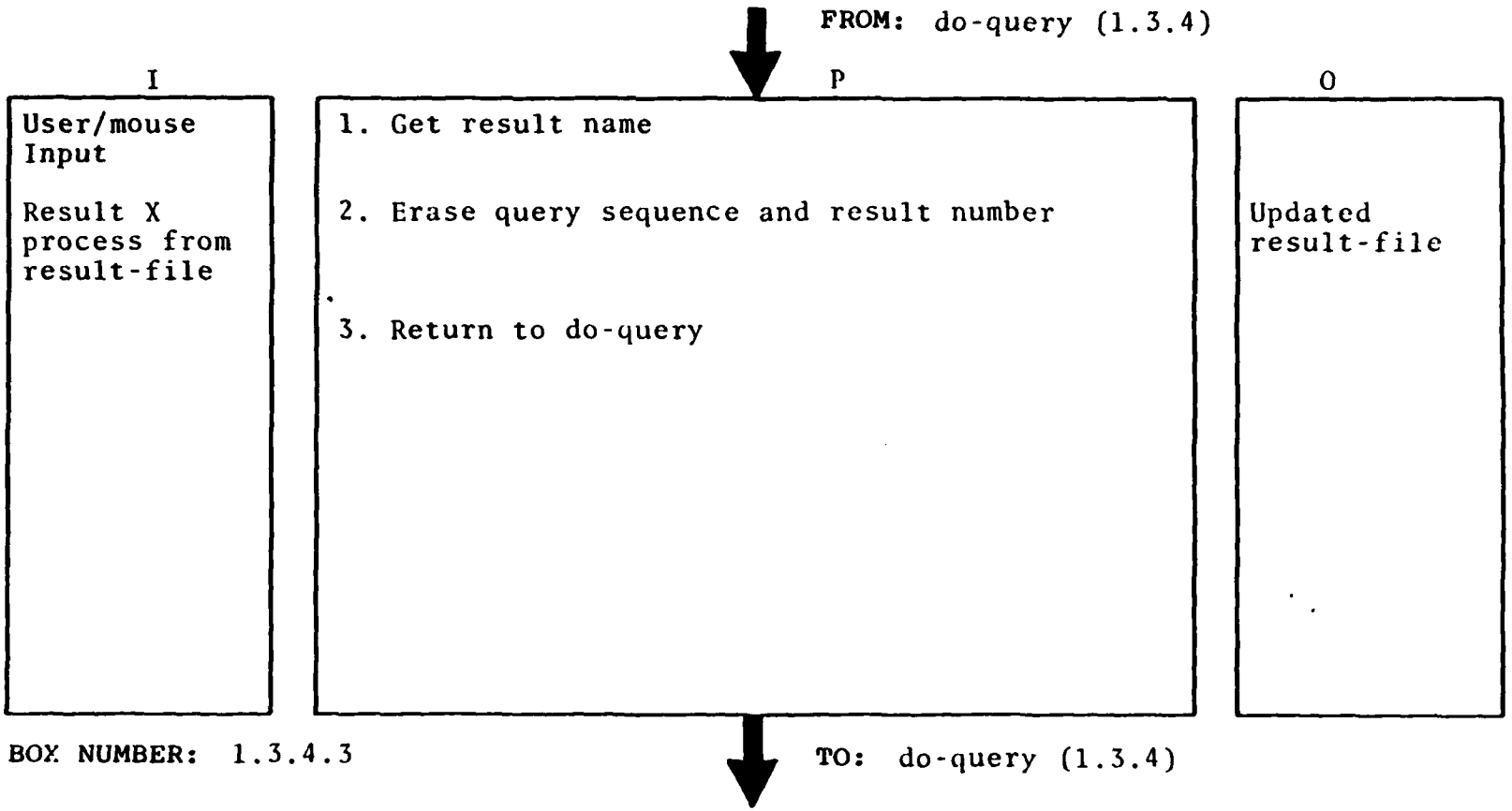


Figure 3.66 Clear-Res.

HIPO DETAIL DIAGRAM

for CLEAR-RES (1.3.4.3)

FROM: do-query (1.3.4)

I	P	O
User/mouse Input Result-file Obj-shade-list Curr-descr-res Screen-layout	1. Highlight "clear result" box in menu 2. If mouse is positioned on a result, use it else prompt user for result name 3. Delete Result-X from result-file 4. Delete Result-X obj_shade-list 5. Check curr-descr-res and curr-show-res lists to see if result is being described or shown and delete if so 6. Call screen-layout (draw-schema)(U2) to redraw schema 7. Return "clear result" to normal video 8. Retrun to do-query (1.3.4)	"clear result" highlighted User prompt Updated result-file Updated obj-shade-list Updated -curr-descr-res -curr-show-res -schema display Highlighting removed

BOX NUMBER: 1.3.4.3

TO: do-query (1.3.4)
screen-layout (U2)

Figure 3.67 Clear-Res.

HIPO NOTES

for CLEAR-RES (1.3.4.3)

Step	Note
3.	This result-file is the one created by create-res and includes filenames and query sequences. The "X" refers to the integer used by the numbering system for that particular result sequence.
4.	This obj-shade-list is the objects which were used in the result formulation. It is referenced by the particular result sequence.

Figure 3.68 Clear-Res.

HIPO OVERVIEW DIAGRAM
for CREATE-RES (1.3.4.4)

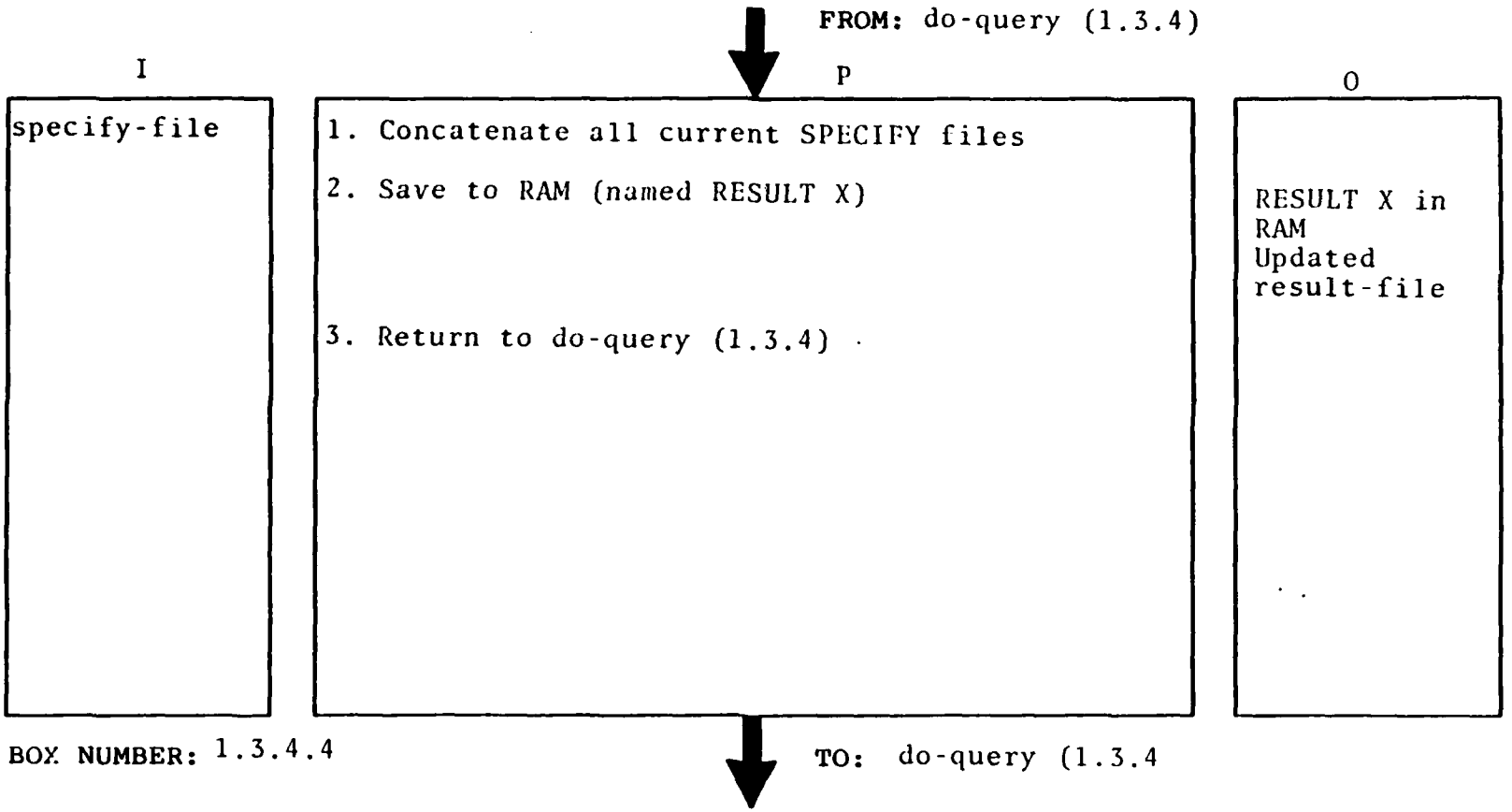


Figure 3.69 Create-Res.

HIPO DETAIL DIAGRAM
for CREATE-RES (1.3.4.4)

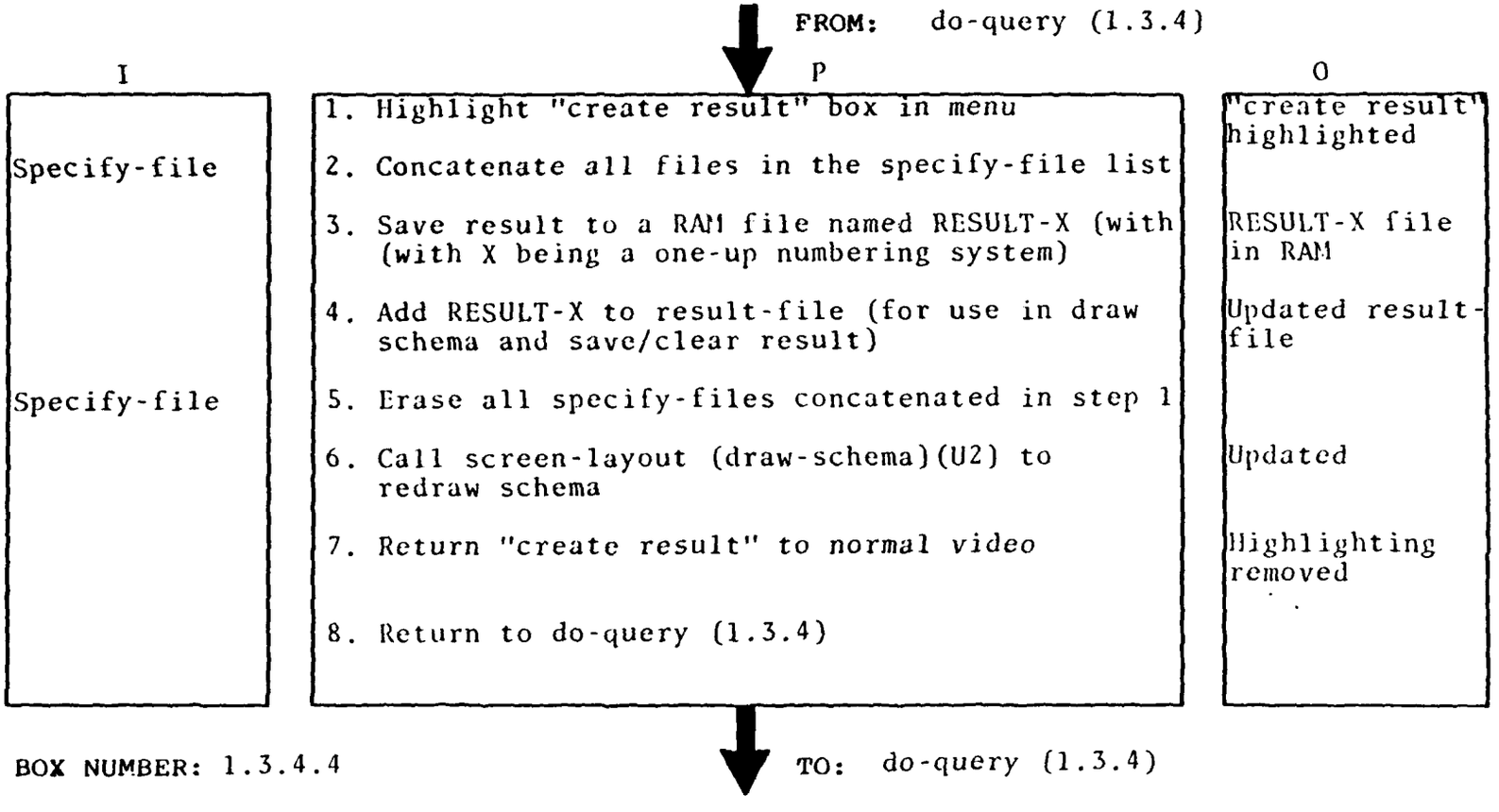


Figure 3.70 Create-Res.

HIPO NOTES
for CREATE-RES (1.3.4.4)

Step	Note
2.	This step will result in a file which contains all the steps previously identified by spec-obj
4.	This creates/adds to a list of results which the user has created during this session. It will be used to determine how many results are shown in the schema display, along with other user options, such as save/show/clear result.
5.	Once the new file has been created, all current entries in specify-file are cleared so the user can begin his next query formulation.

Figure 3.71 Create-Res.

HIPO OVERVIEW DIAGRAM

for COPY-RES (1.3.4.5)

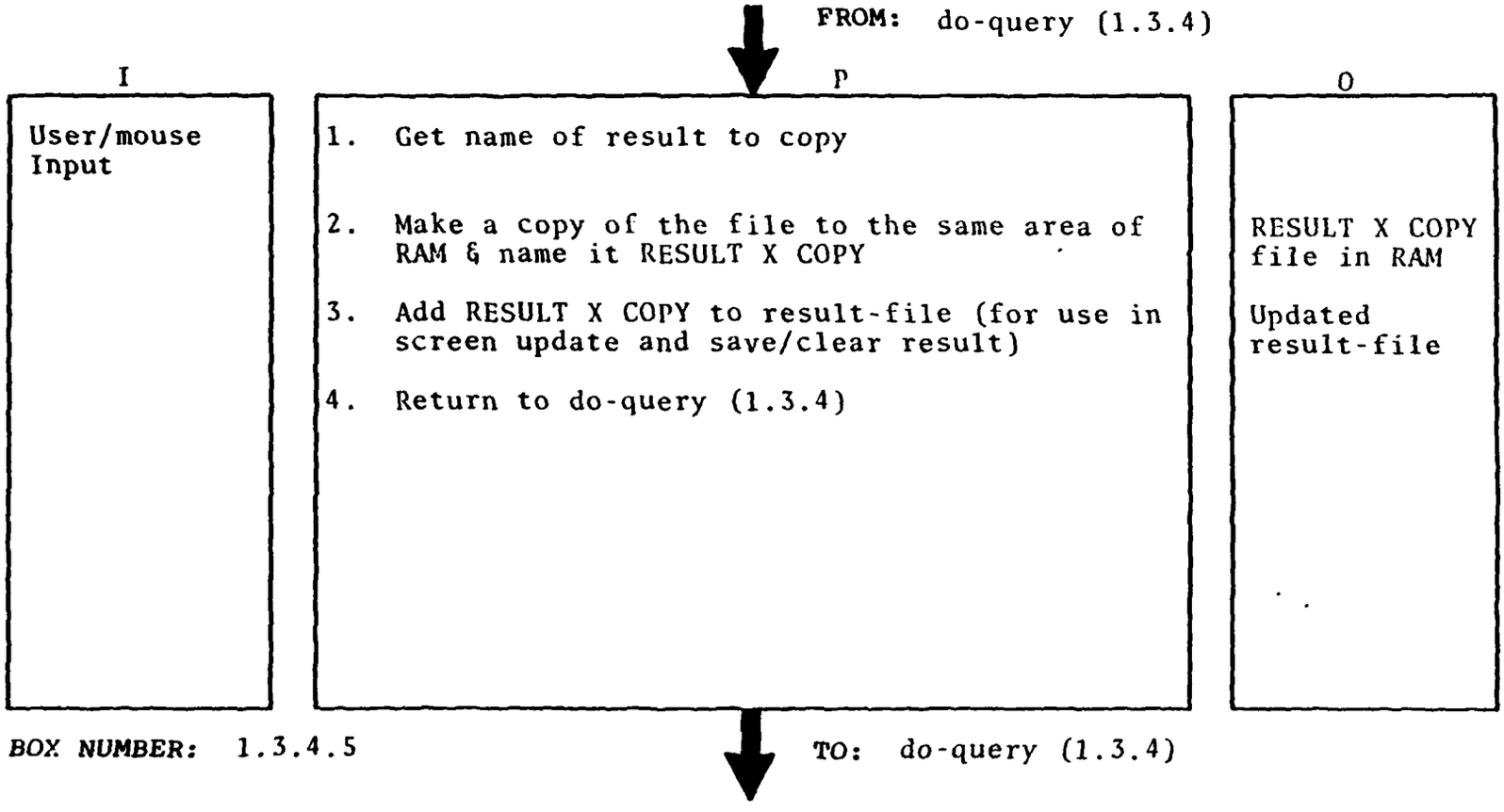
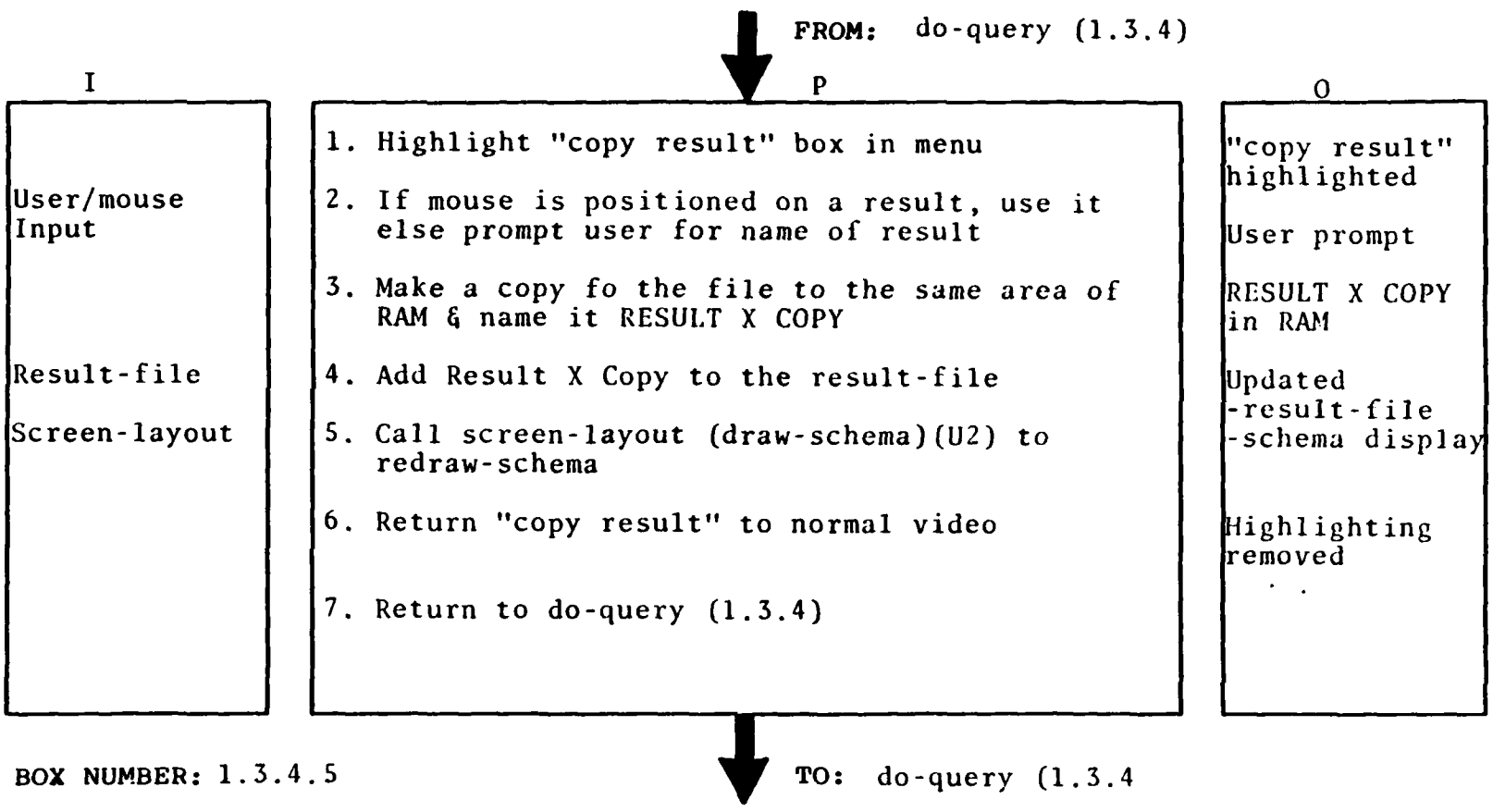


Figure 3.72 Copy-Res.

HIPO DETAIL DIAGRAM

for COPY-RES (1.3.4.5)

FROM: do-query (1.3.4)



BOX NUMBER: 1.3.4.5

TO: do-query (1.3.4

Figure 3.73 Copy-Res.

HIPO OVERVIEW DIAGRAM
for COMB-RES (1.3.4.6)

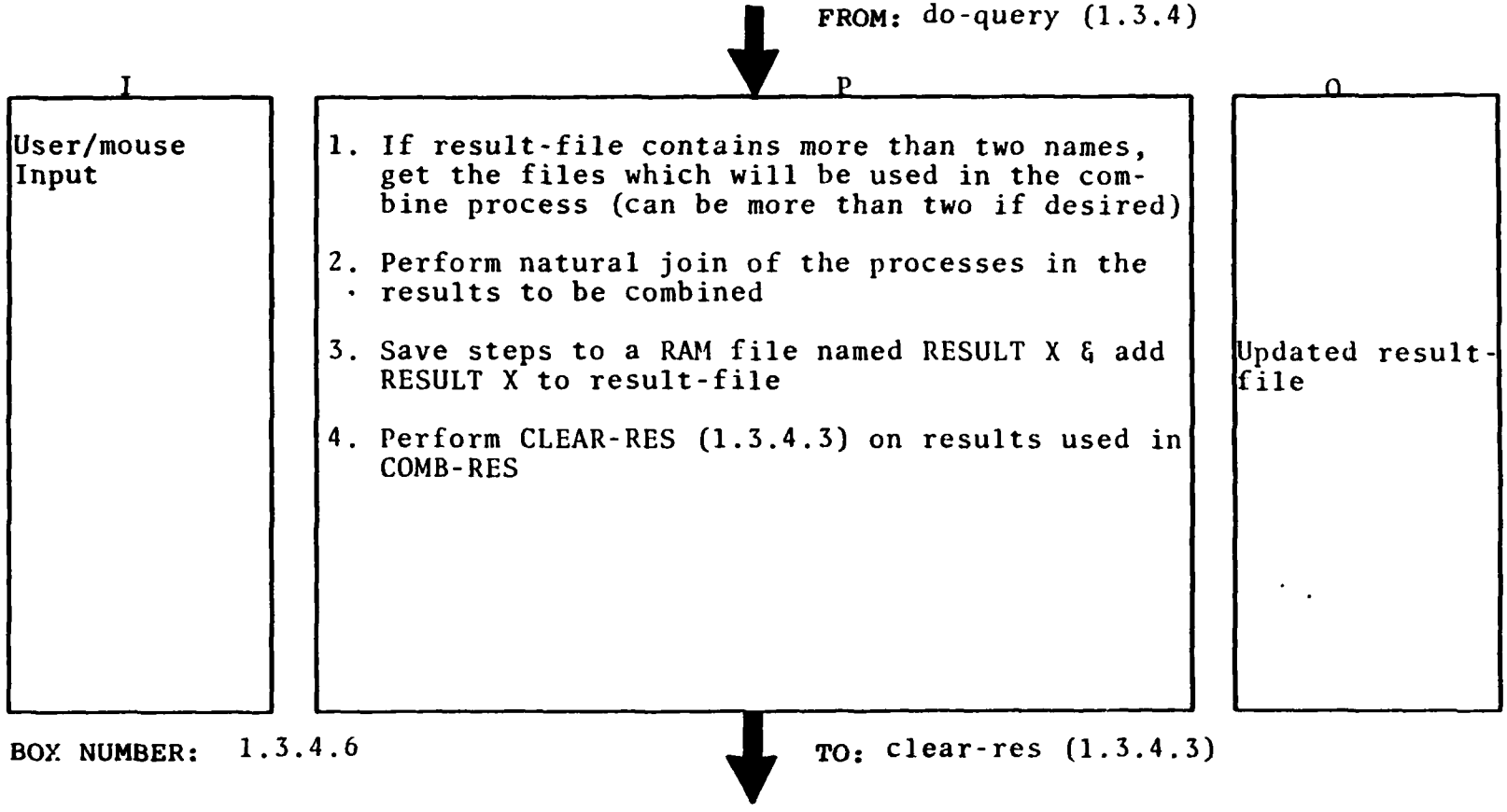


Figure 3.74 Comb-Res.

HIPO DETAIL DIAGRAM
for COMB-RES (1.3.4.6)

FROM: *do-query* (1.3.4)

I	P	O
	1. Highlight "combine results" box in menu	"combine results" highlighted
User/mouse Input	2. *If result-file contains more than two entries,* prompt user for names of results to be combined	User *prompt*
	3. Concatenate files identified in step 1.	
	4. Save result to a RAM file named RESULT-X	RESULT-X file in RAM
Result-file	5. Delete result names (in step 1) from result-file and add new name	Updated result-file
Obj-Shade-list	6. Combine lists from the results in step 1 to a new list in obj-shade-list with new reference name RESULT-X	Updated obj-shade-list
Screen-layout	7. Call screen-layout (draw-schema)(U2) to re-draw schema	Updated display
	8. *Return "combine results" to normal video*	Highlighting removed
	9. Return to do-query (1.3.4)	

BOX NUMBER: 1.3.4.6

TO: do-query (1.3.4)
screen-layout (U2)

Figure 3.75 Comb-Res.

HIPO NOTES

for COMB-RES (1.3.4.6)

Step	Note
	This module is essentially the same as create-res except that it takes the steps previously defined for multiple queries and combines them into one larger query. Because of that, there are several "house-keeping" steps to straighten out the files abd schema display.

Figure 3.76 Comb-Res.

HIPO OVERVIEW DIAGRAM
for SPEC-OBJ (1.3.4.7)

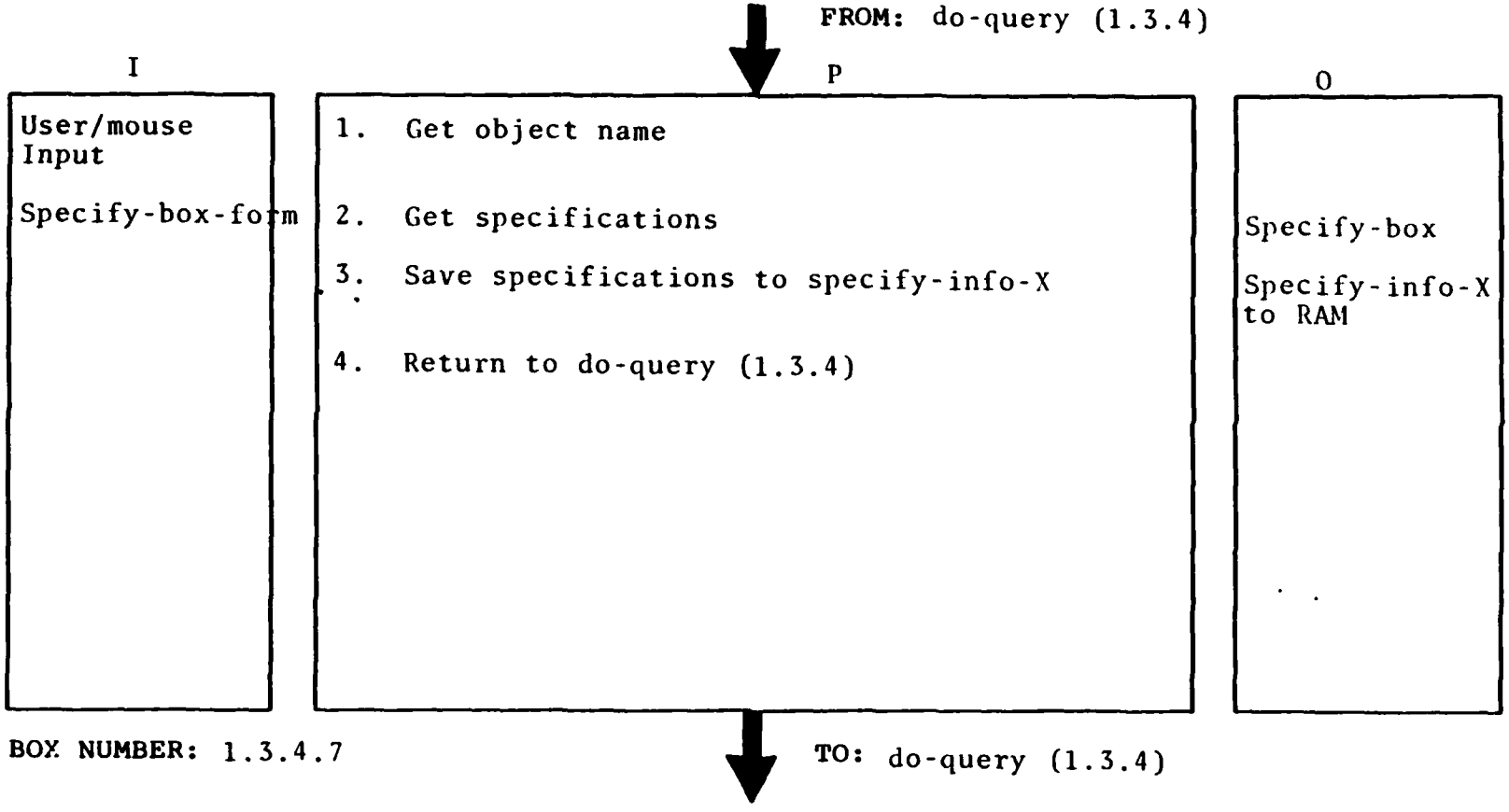


Figure 3.77 Spec-Obj.

HIPO DETAIL DIAGRAM
for SPEC-OBJ (1.3.4.7)

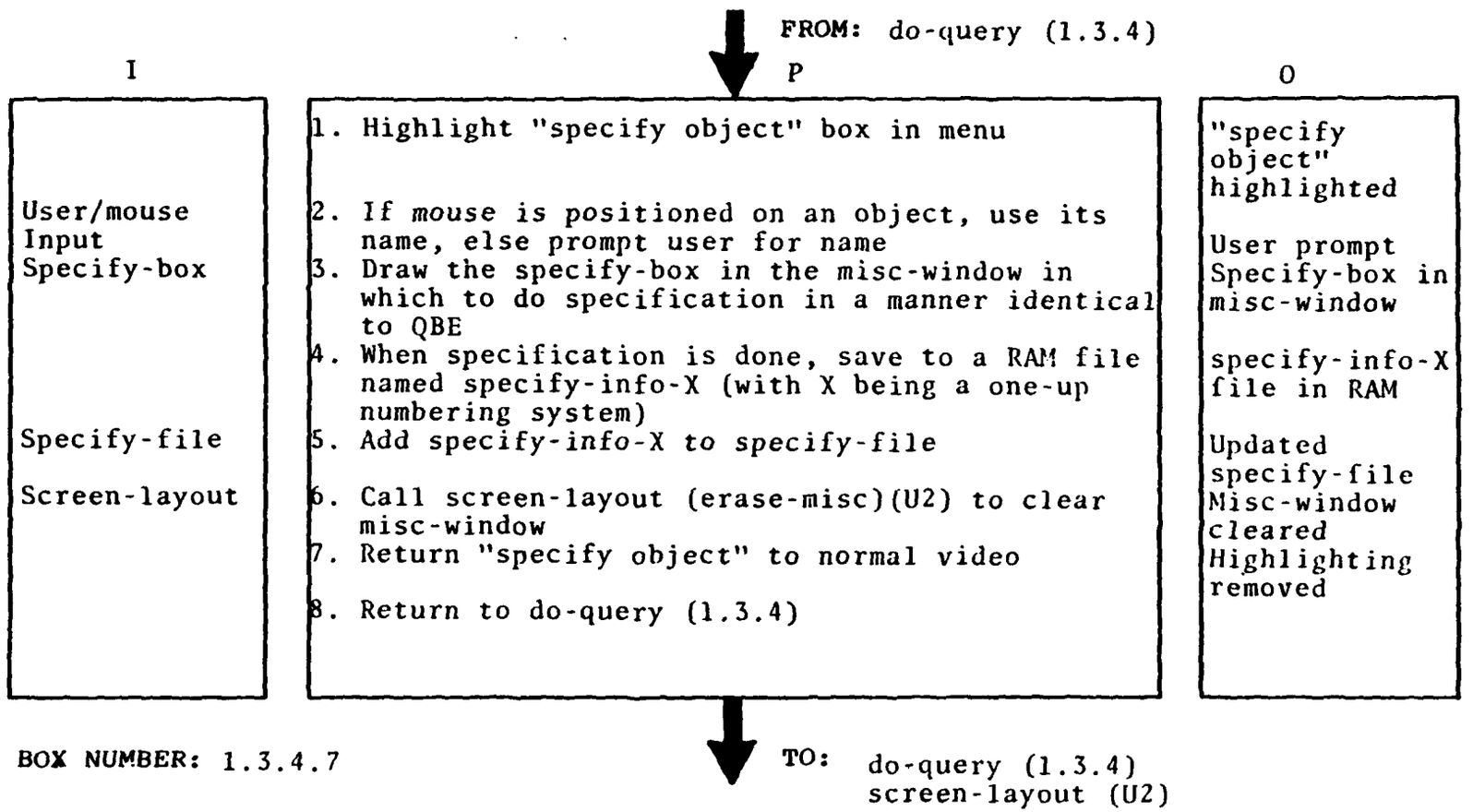


Figure 3.78 Spec-Obj.

HIPO NOTES

for SPEC-OBJ (1.3.4.7)

Step	Note
3.	The commands are not given here for the QBE query formulations. Specific information has not yet been received to provide complete guidance, but such information should be in hand before proceeding with query rules/codes to ensure consistency.
5.	This file contains a list of the names of the specify-info-X files which will be used to create the result.

Figure 3.79 Spec-Obj.

HIPO OVERVIEW DIAGRAM
for DESCR-RES (1.3.4.8)

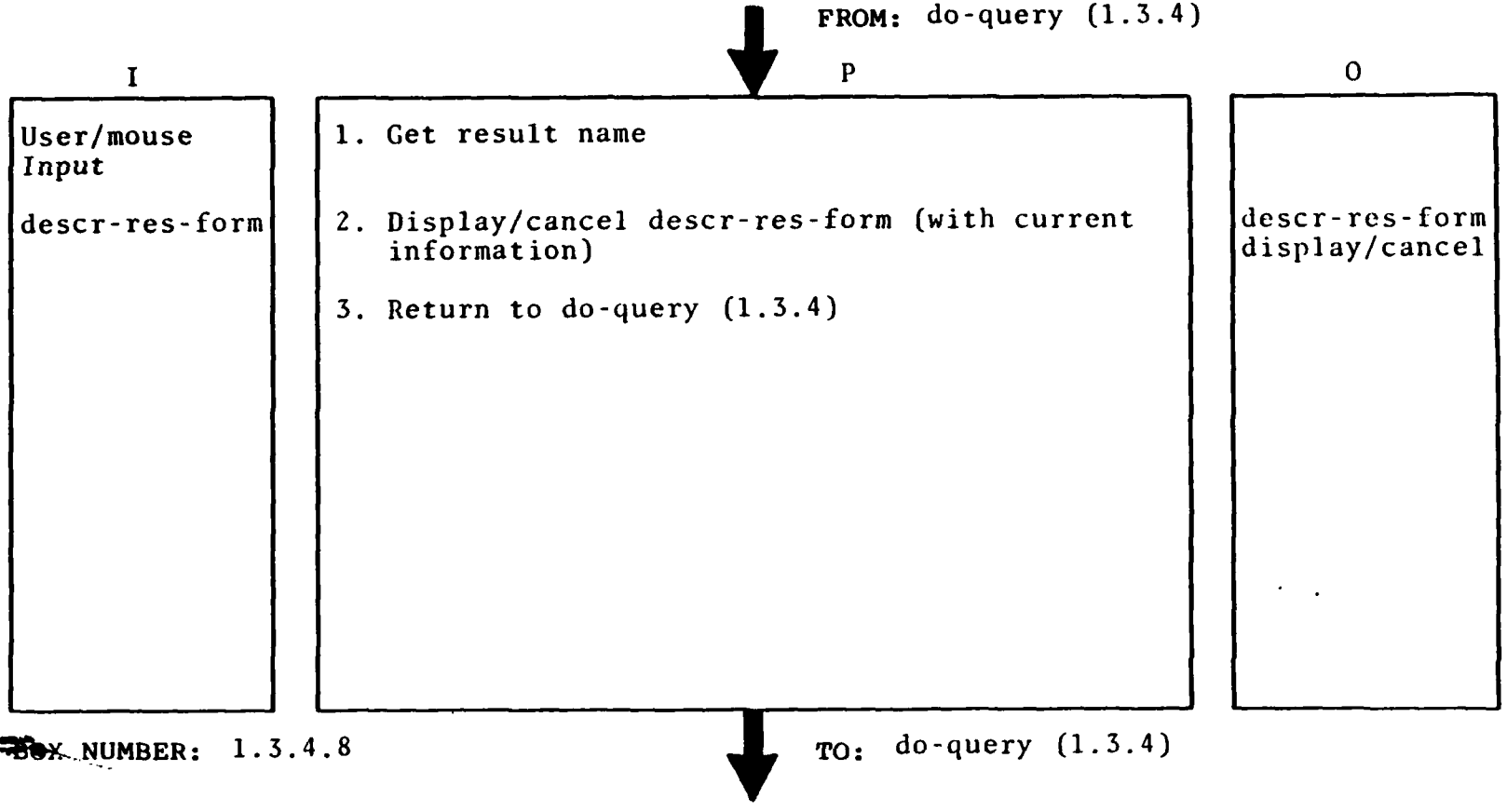


Figure 3.80 Descr-Res.

HIPO DETAIL DIAGRAM
for DESCR-RES (1.3.4.8)

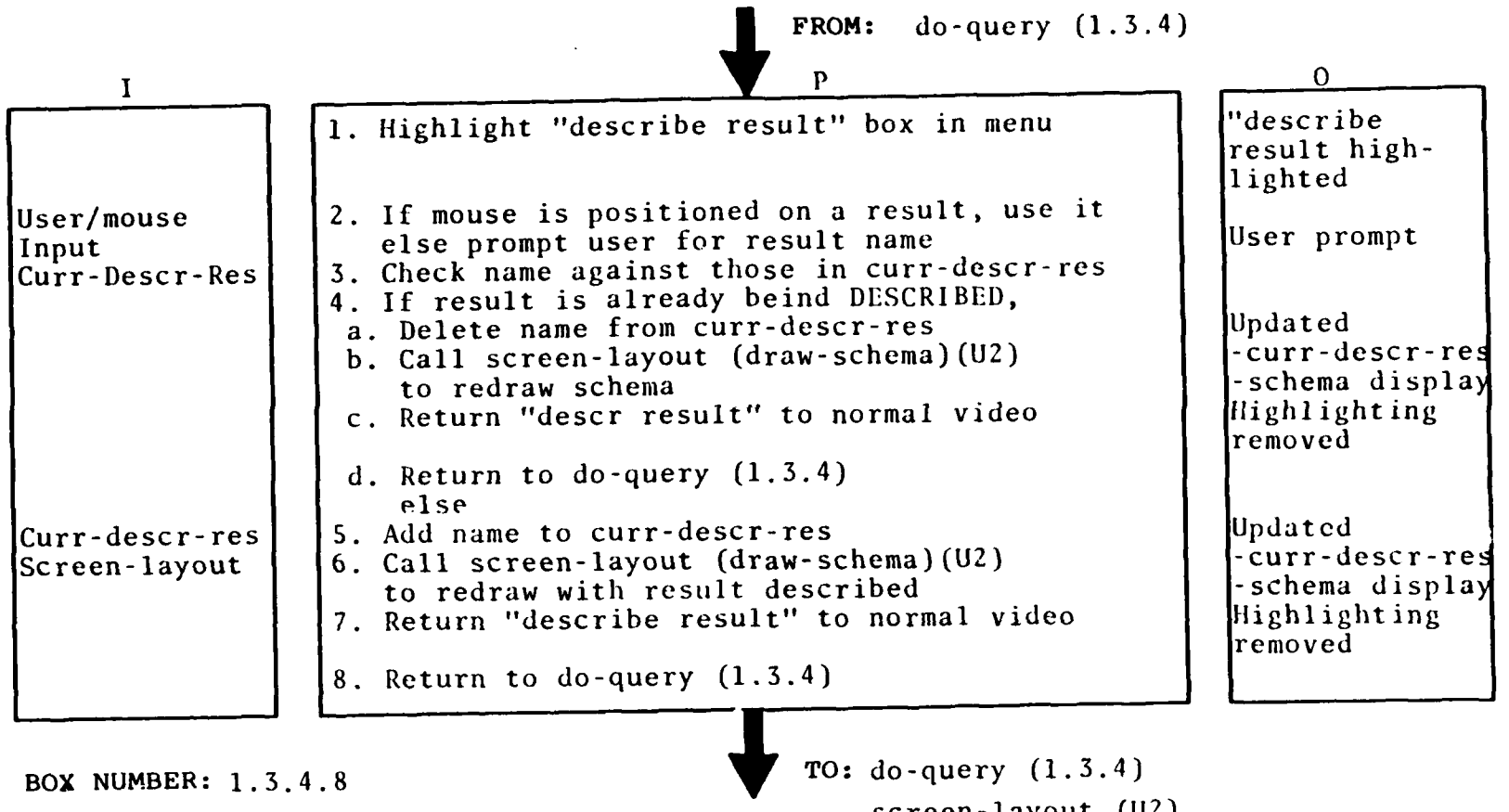


Figure 3.81 Descr-Res.

HIPO OVERVIEW DIAGRAM

for CREATE-REP (1.3.4.9)

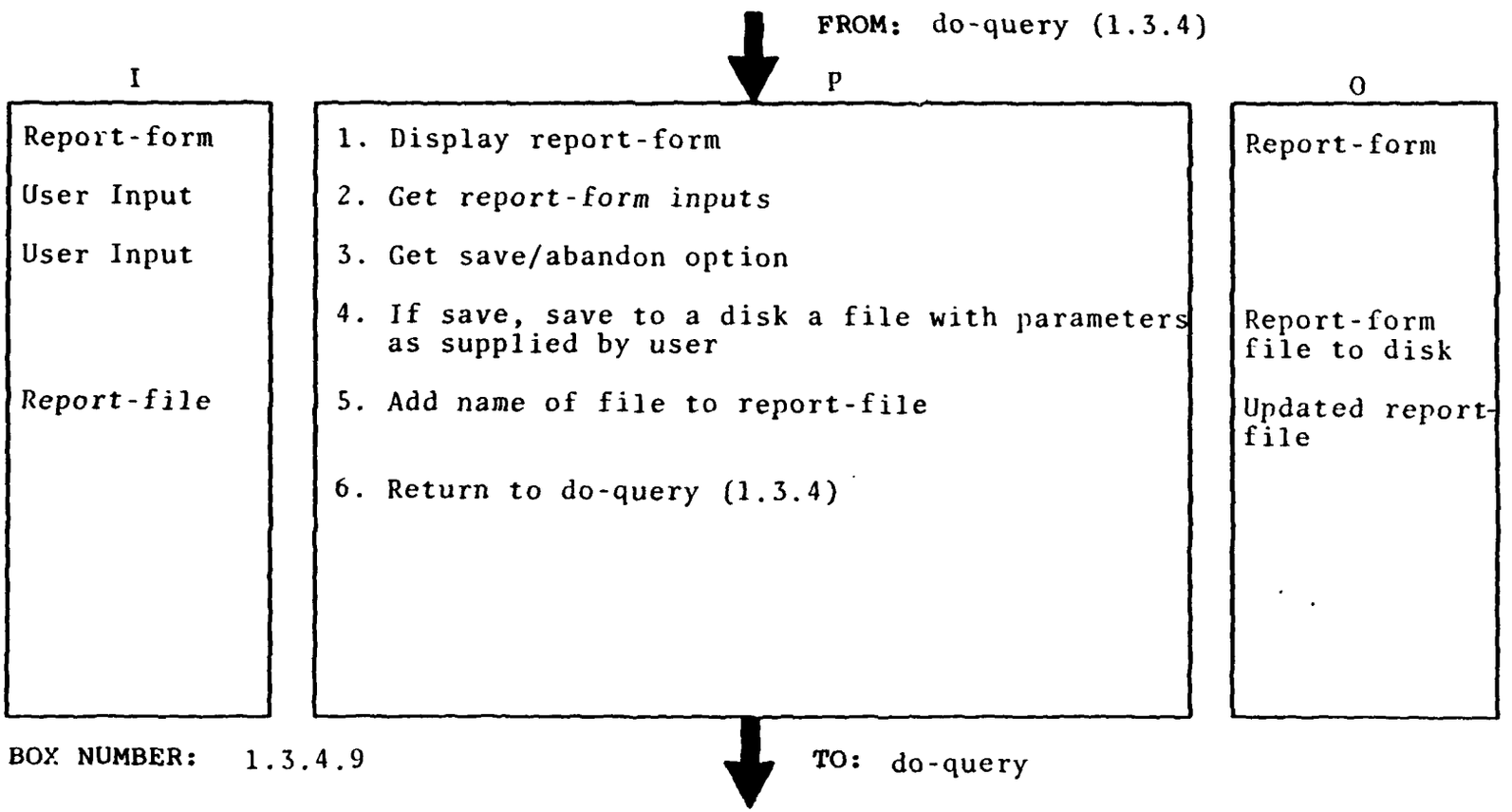


Figure 3.82 Create-Rep.

HIPO DETAIL DIAGRAM
for CREATE-REP (1.3.4.9)

FROM: do-query (1.3.4)

I	P	O
User/mouse Input Report-file Err-msgs Report-form User input User/mouse Input Screen-layout	1. Highlight "create report" box in menu 2. Get name for report 3. Check name against report-file to ensure name is not already being used 4. If name has been used, print error message and return to step 1 else 5. Display report-form in misc-window 6. Get inputs/modifications until done 7. Get save/abandon option 8. If save, save to disk the information in the report-form and add name to report-file 9. Call screen-layout (erase-misc)(U2) to clear misc-window 10. Return "create report" to normal video 11. Return to do-query (1.3.4)	"create-report highlighted Error message Report-form display Report-form info to disk Misc-window cleared Highlighting removed

BOX NUMBER: 1.3.4.9

TO: do-query (1.3.4)
screen-layout (U2)

Figure 3.83 Create-Rep.

HIPO NOTES
for CREATE-REP (1.3.4.9)

Step	Note
	This module should be implemented in a manner very similar to update-obj (1.3.2). It should have a standard form displayed that the user can fill in with his parameters.
3.	This report-file is a file containing the names of reports the user has previously created.
7.	The save/abandon option only affects current work. That is, it cannot be used to call up a previously defined report form and erase it. There should be some device implemented in the form to allow deleting a report file.

Figure 3.84 Create-Rep.

HIPO OVERVIEW DIAGRAM
for QU-HELP (1.3.4.10)

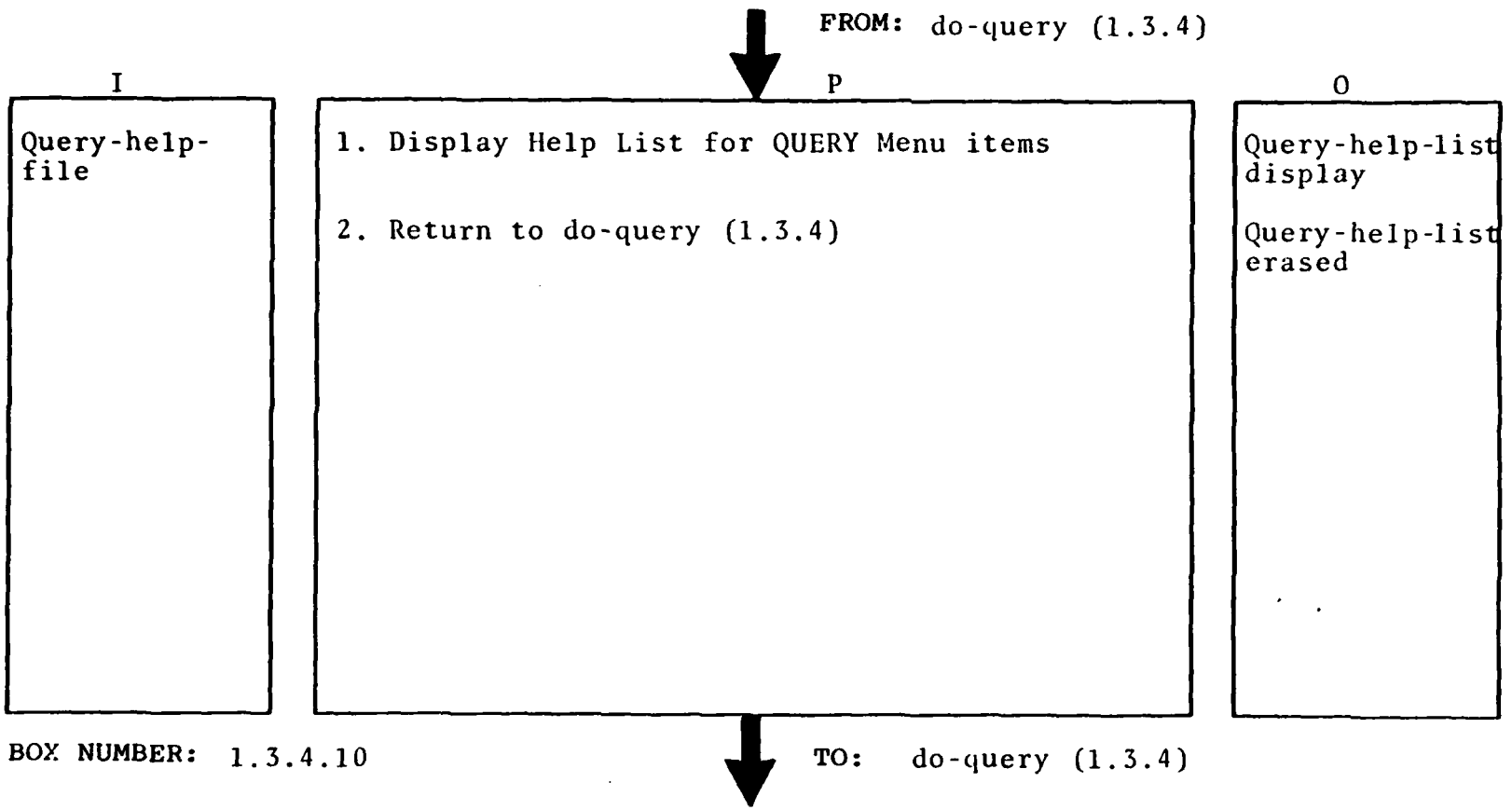


Figure 3.85 *Qu-Help.*

HIPO DETAIL DIAGRAM
for QU-HELP (1.3.4.10)

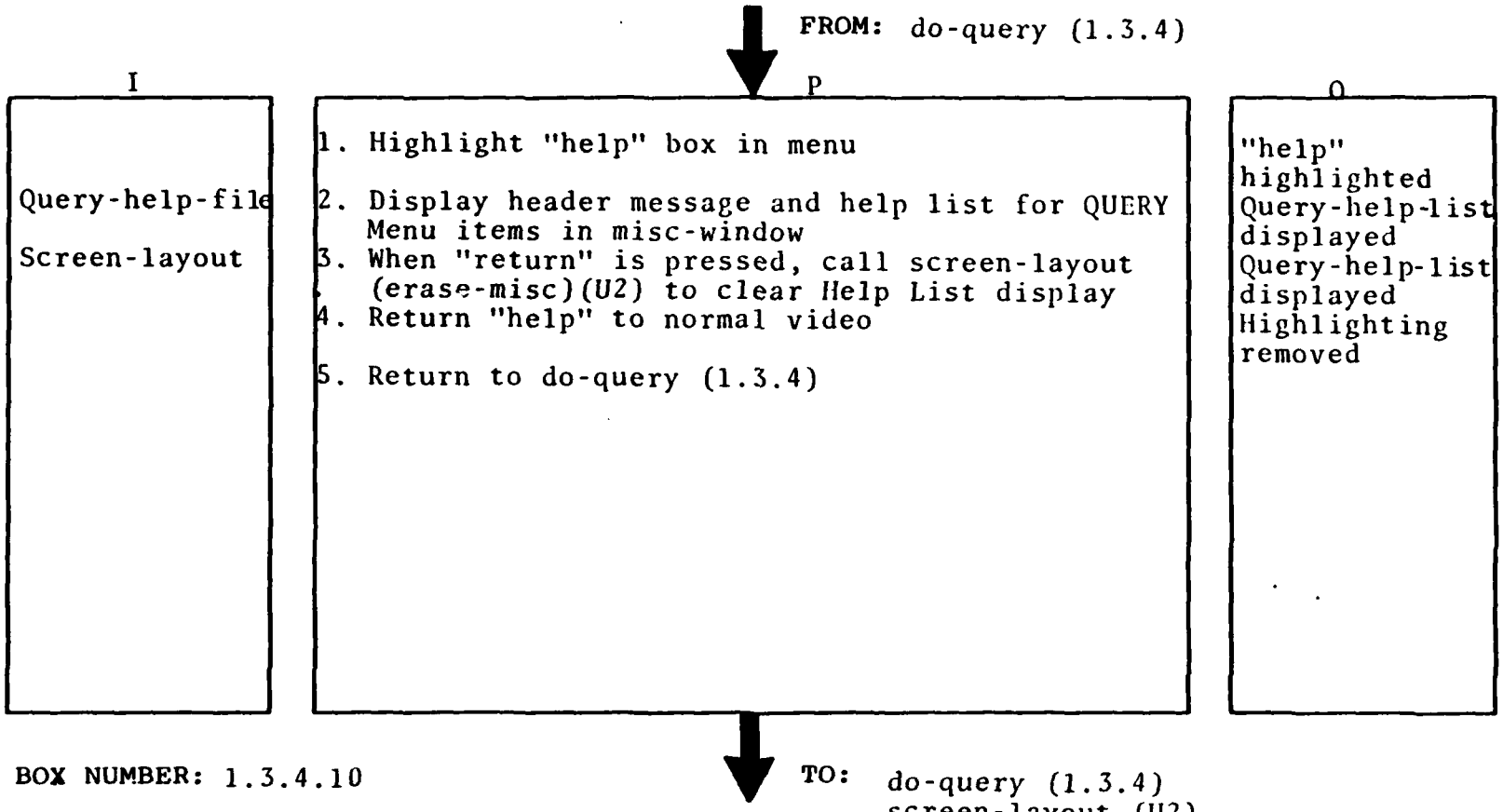


Figure 3.86 *Qu-Help.*

HIPO OVERVIEW DIAGRAM
for ERR-MSGS (U1)

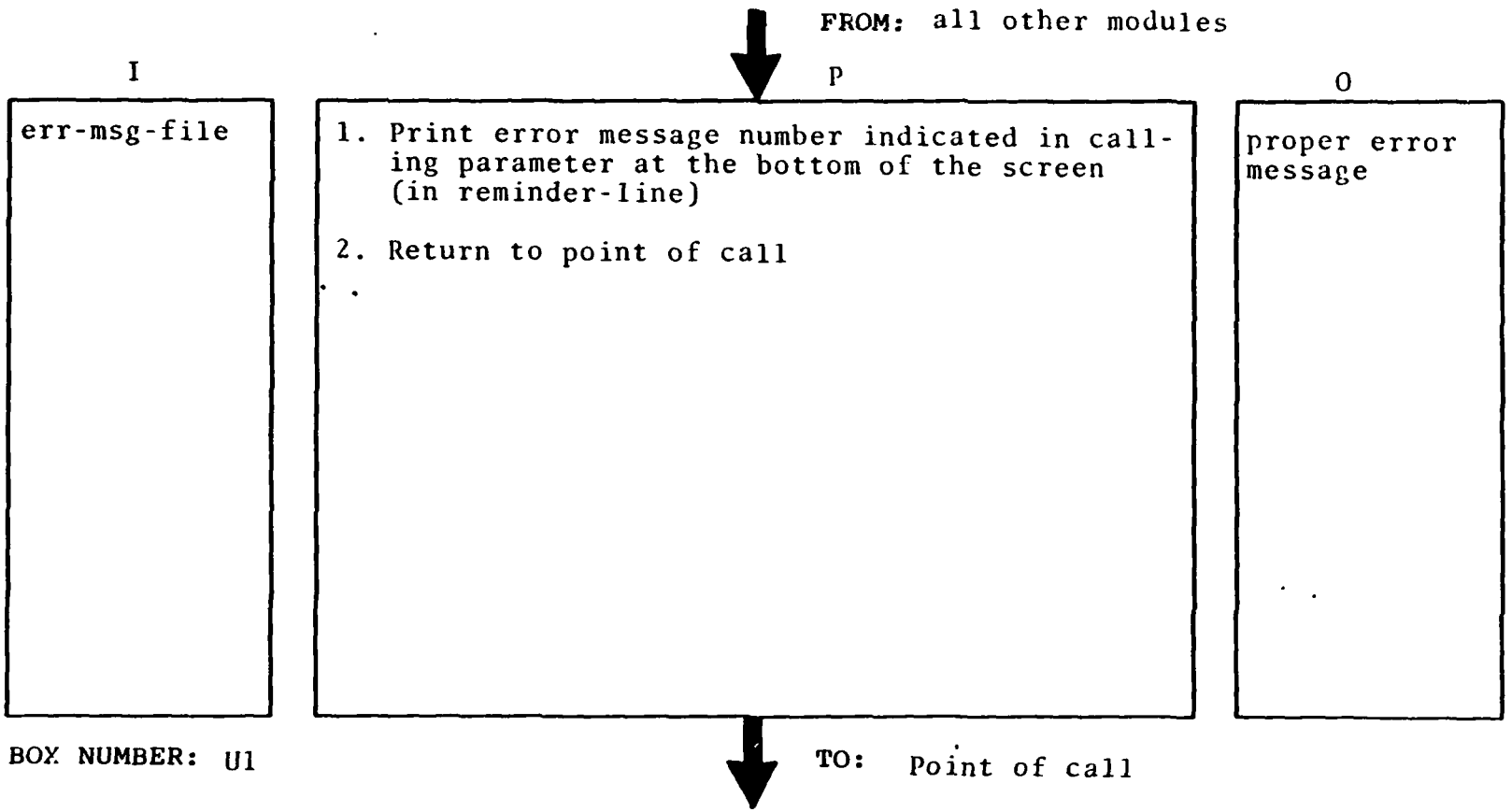


Figure 3.87 Err-Msgs.

HIPO DETAIL DIAGRAM
for ERR-MSGS (U1)

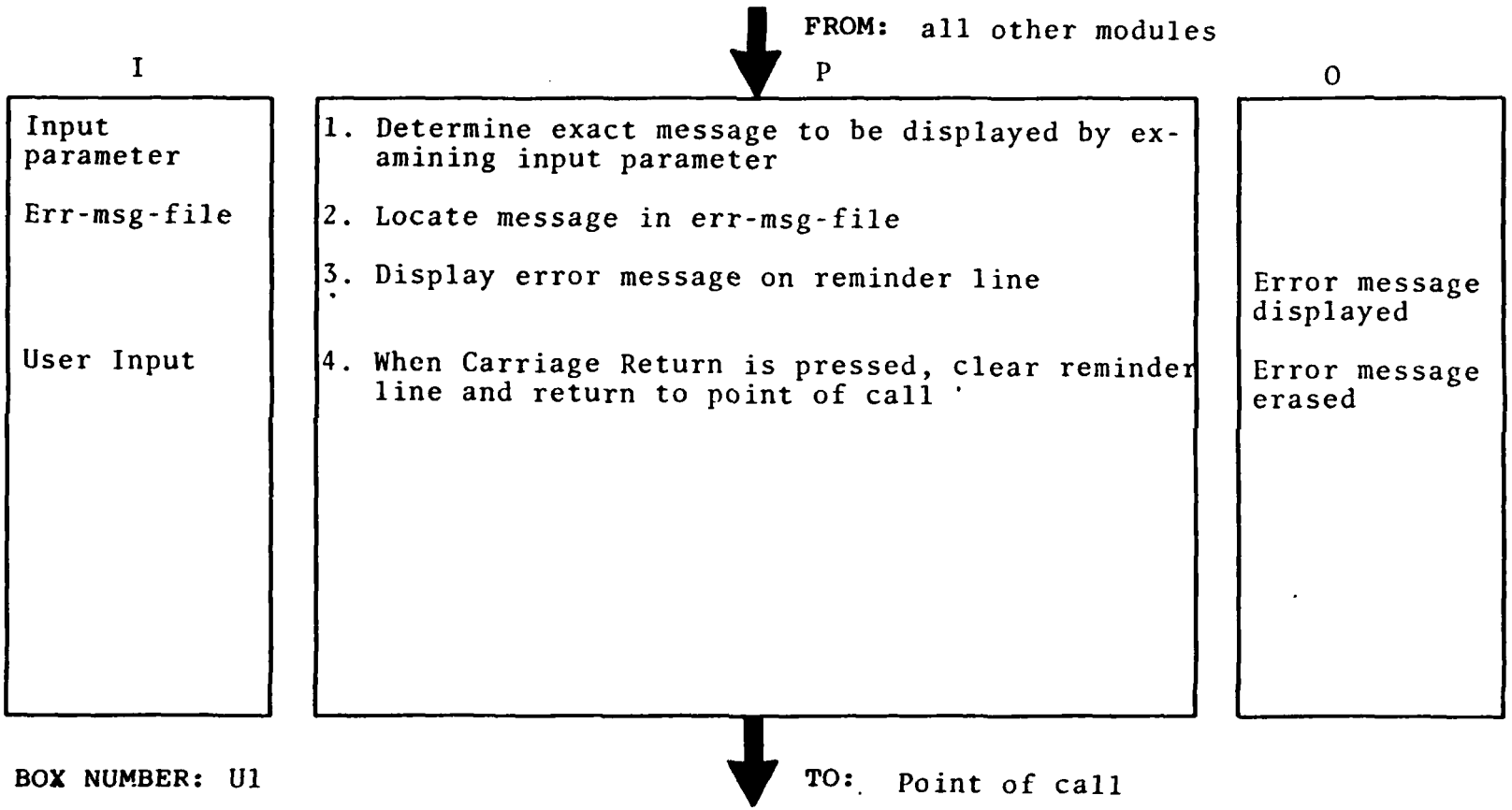


Figure 3.88 Err-Msg.

HIPO NOTES
for ERR-MSGS (U1)

Step	Note
1.	This input parameter should indicate a coding scheme which will identify the class of error message, and the particular error message within that class. By using such a scheme, locating the message will be simplified, messages can be shared by many modules, and the message contents can be standardized (i.e. the same type of error anywhere in the program will get exactly the same error response).
2.	By using a well-devised coding scheme, the search technique to find a particular entry should be trivial.
4.	Forcing the user to press Carriage Return (or any other symbol) to continue implies that we have a novice user that must be led through the process. This may not be necessary, and may annoy the sophisticated user. This is really a question of personal style, though, and can be altered if desired. In fact, one could provide a default option of "Help me through" or "Leave me alone." Another important point to note here is that this is the first module which returns to the point of call rather than simply calling another module. By this I mean that you must return to the step within the calling module rather than the module itself.

Figure 3.89 Err-Msgs.

HIPO OVERVIEW DIAGRAM
for SCREEN-LAYOUT (U2)

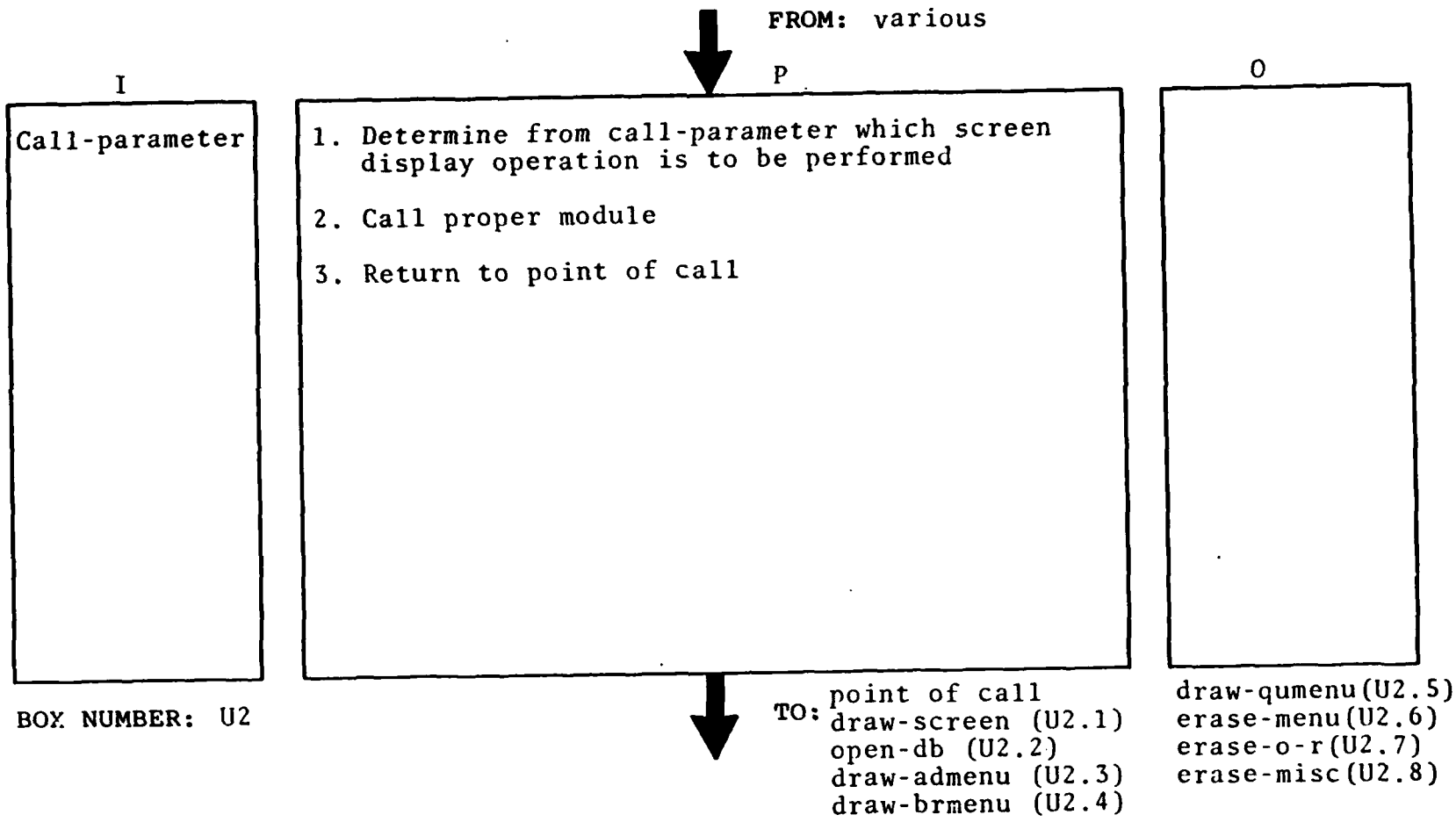


Figure 3.90 Screen-Layout.

HIPO DETAIL DIAGRAM
for SCREEN-LAYOUT (U2)

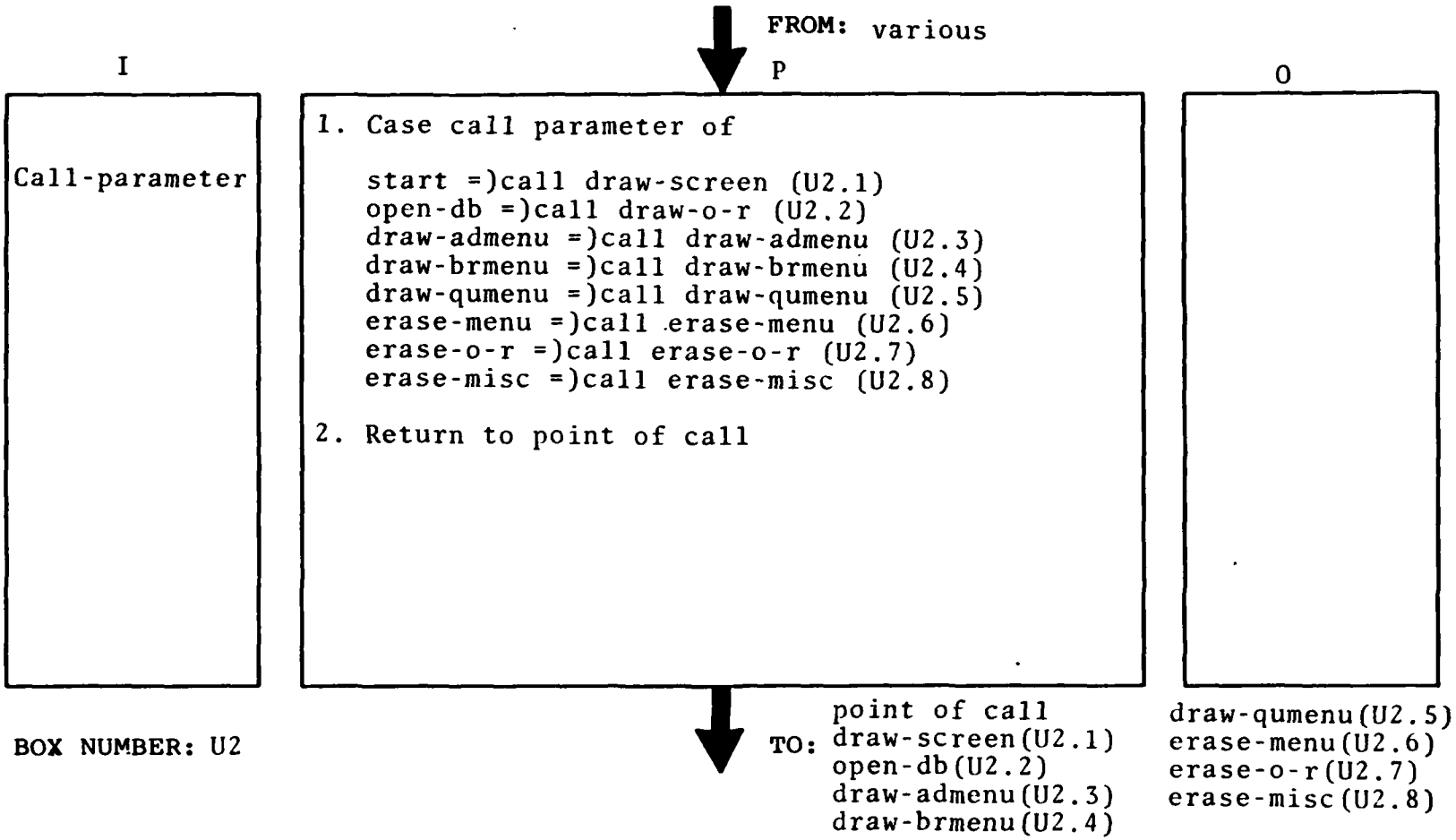


Figure 3.91 Screen-Layout.

HIPO NOTES
for SCREEN-LAYOUT (U2)

Step	Note
1.	This simple case statement uses the input parameter to determine which sub-module should be called
2.	This module, like err-msgs (U1), must return to the particular point of call within the calling module From a strict efficiency point-of-view, this module is not necessary at all. The reason it has been included is that it increases program clarity by providing a single module to interface with the rest of the program. Since this project is being accomplished by more than one person, we do all that is possible to ensure clear meaning and methodology.

Figure 3.92 Screen-Layout.

HIPO OVERVIEW DIAGRAM
for DRAW-SCREEN (U2.1)

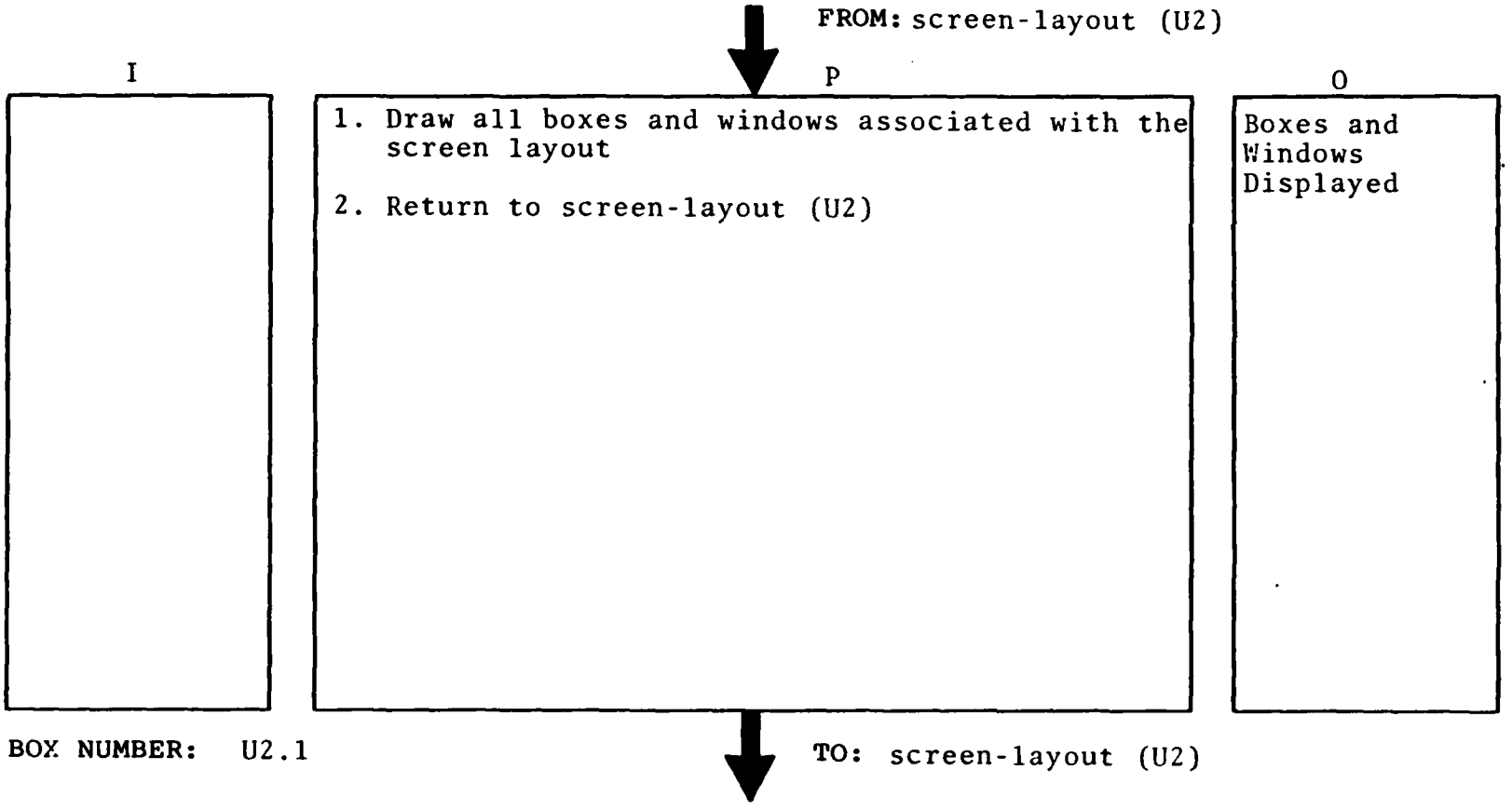


Figure 3.93 Draw-Screen.

HIPO DETAIL DIAGRAM

for DRAW-SCREEN (U2.1)

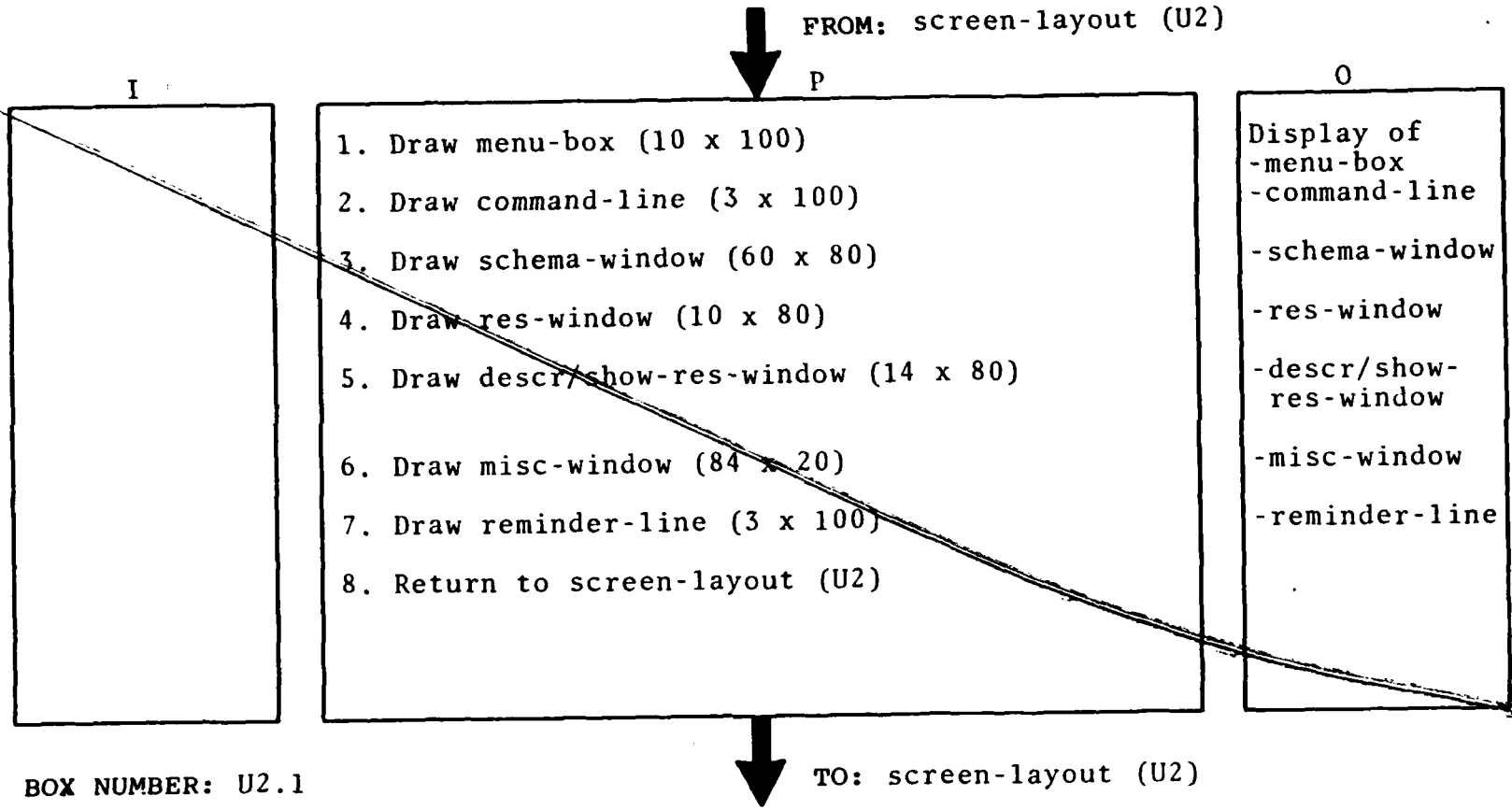


Figure 3.94 Draw-Screen.

HIPO NOTES
for DRAW-SCREEN (U2.1)

Step	Note
	All parenthesized numbers in this module are based on the percentage of the screen that the associated box or window will take. The first number is the percentage of standard text rows, the second number is the percentage of standard text columns. These can, however, be adjusted as necessary during implementation. In fact, some of the curr-defaults could be current window sizes which could be saved for future use. There are three types of drawings to be made by this module. The differentiation is as follows: (1)boxes or things which are not intended to be changed. The information which will be displayed in the boxes should fit and there is no reason that the amount of information will change. Boxes are used for menus. 2)windows must be much more flexible. They are drawn with elevator bars along the bottom and right side to show the user the percentage of information in the file which is currently being displayed. Their size may also be changed by using the left "drag" button of the mouse (this brings up the issue of whether this change will remain after the current operation or not). Windows are used where varying amounts of information will be displayed, such as the schema, 3)lines are really miniature boxes. They will display or accept only one line of information. Lines are used for commands and reminders.

Figure 3.95 Draw-Screen.

HIPO OVERVIEW DIAGRAM
for DRAW-SCHEMA (U2.2)

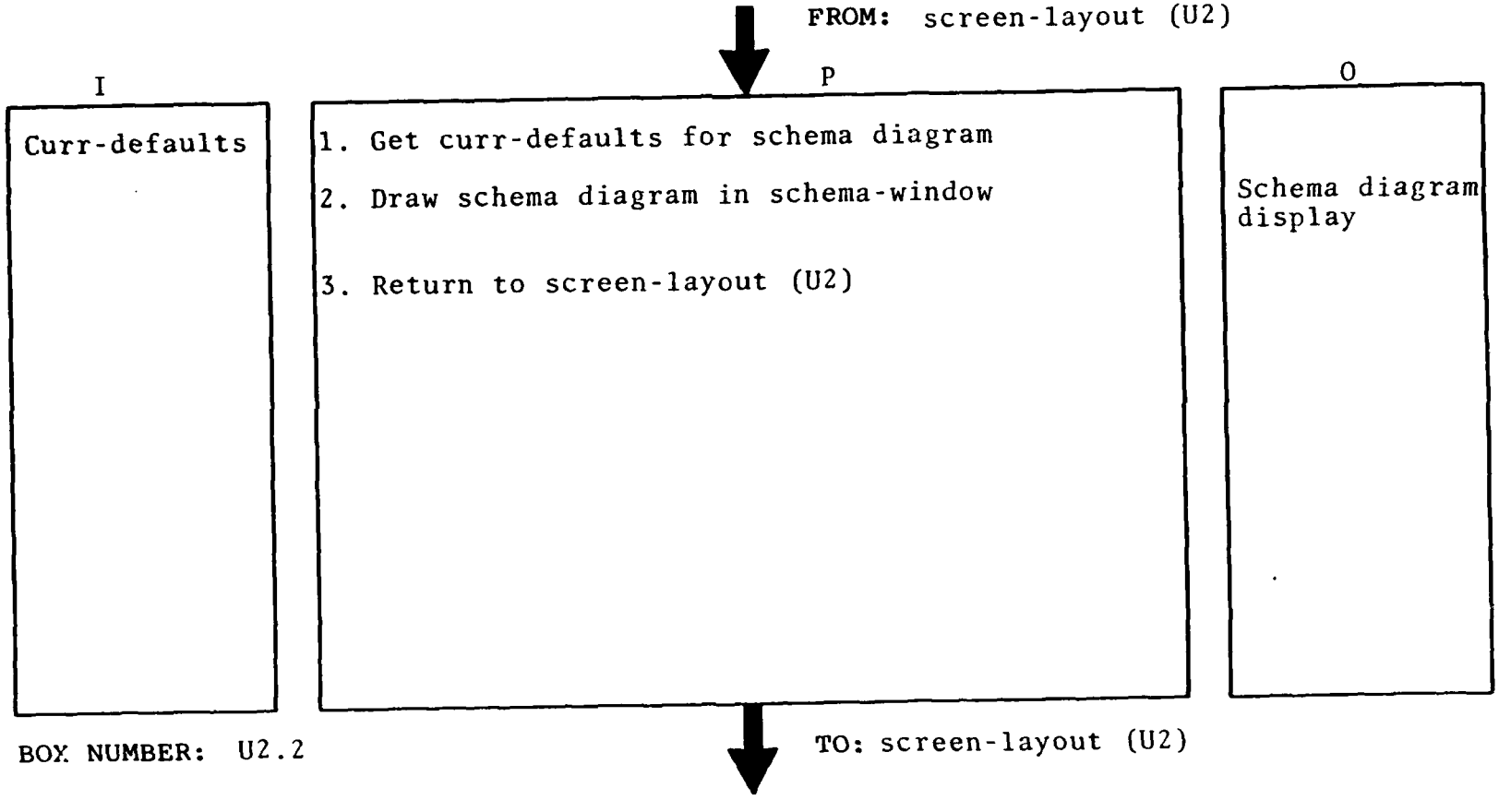


Figure 3.96 Draw-Schema.

HIPO DETAIL DIAGRAM
for DRAW-SCHEMA (U2.2)

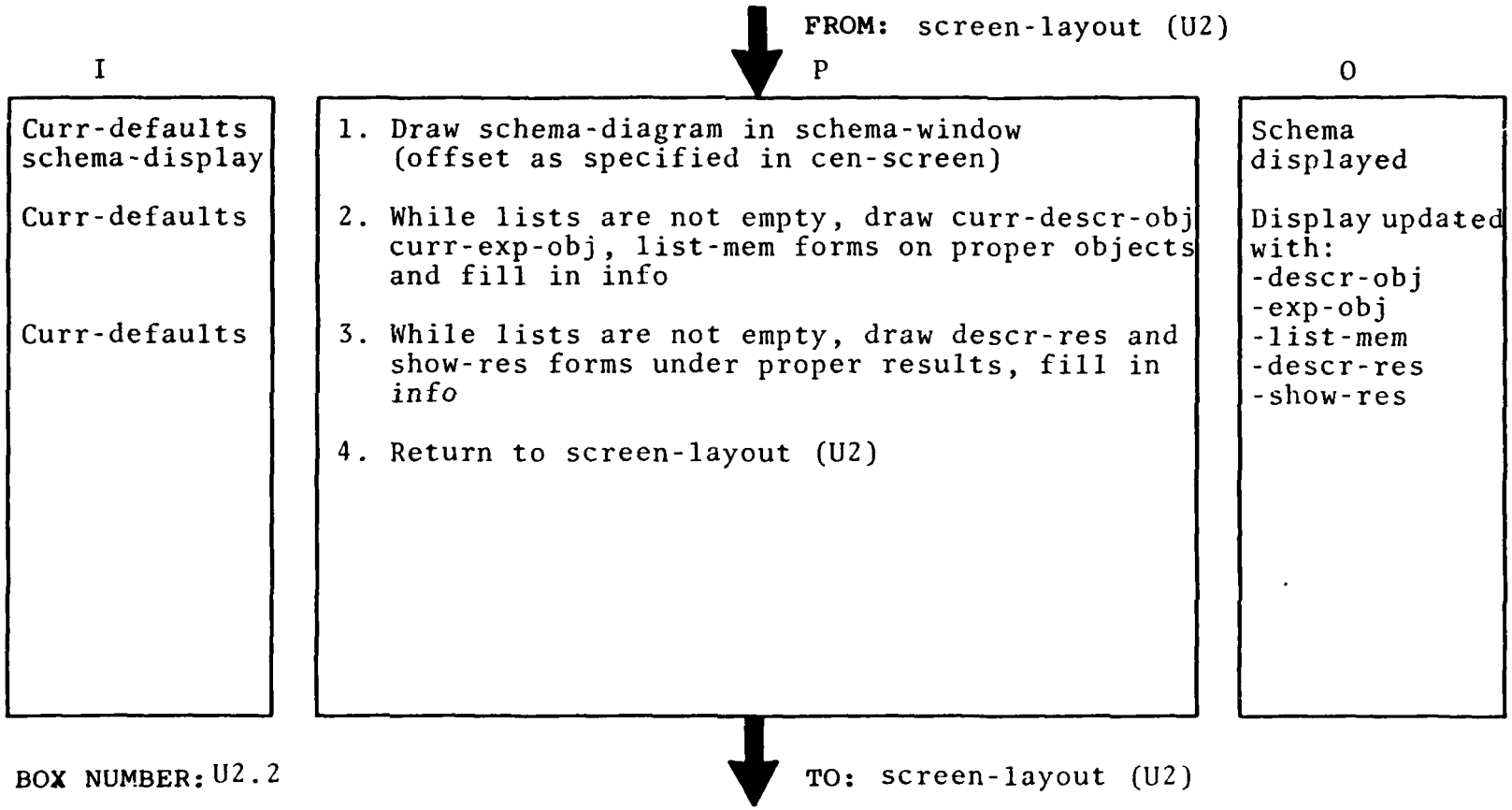


Figure 3.97 Draw-Schema

HIPO NOTES
for DRAW-SCHEMA (U2.2)

Step	Note
1.	This schema-diagram has already been generated for us in the data definition portion of the program. It was built as the database administrator designed the database system. All we have to do here is display it.
2.	These forms are small windows (with varying amounts of information) which will overwrite the object with which they are associated. The way they should be implemented is by a standard form which goes to the file it represents and pulls out the required information to display. It may be a good idea here to save this information the first time it is retrieved and use it each time this module is called provided that it has not been changed (by update-obj).
3.	The same comments as step (2) apply to this step, especially the one about saving the information required to perform this step. In show-res, this is where the data manipulation is actually performed, so it is imperative not to have to rediscover the wheel each time it is ued.

Figure 3.98 Draw-Schema.

HIPO OVERVIEW DIAGRAM
for DRAW-ADMENU (U2.3)

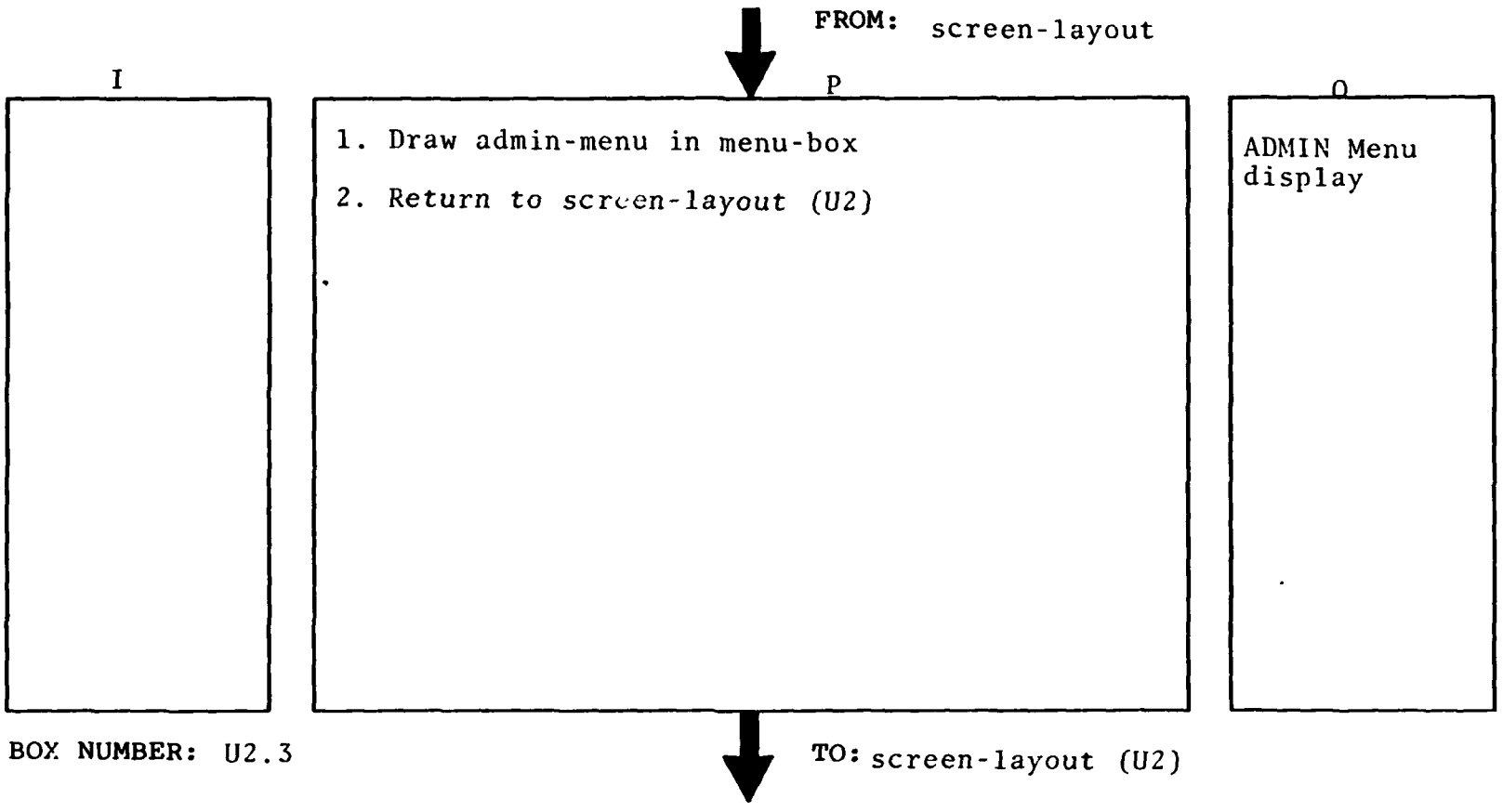


Figure 3.99 Draw-Admenu.

HIPO DETAIL DIAGRAM
for DRAW-ADMENU (U2.3)

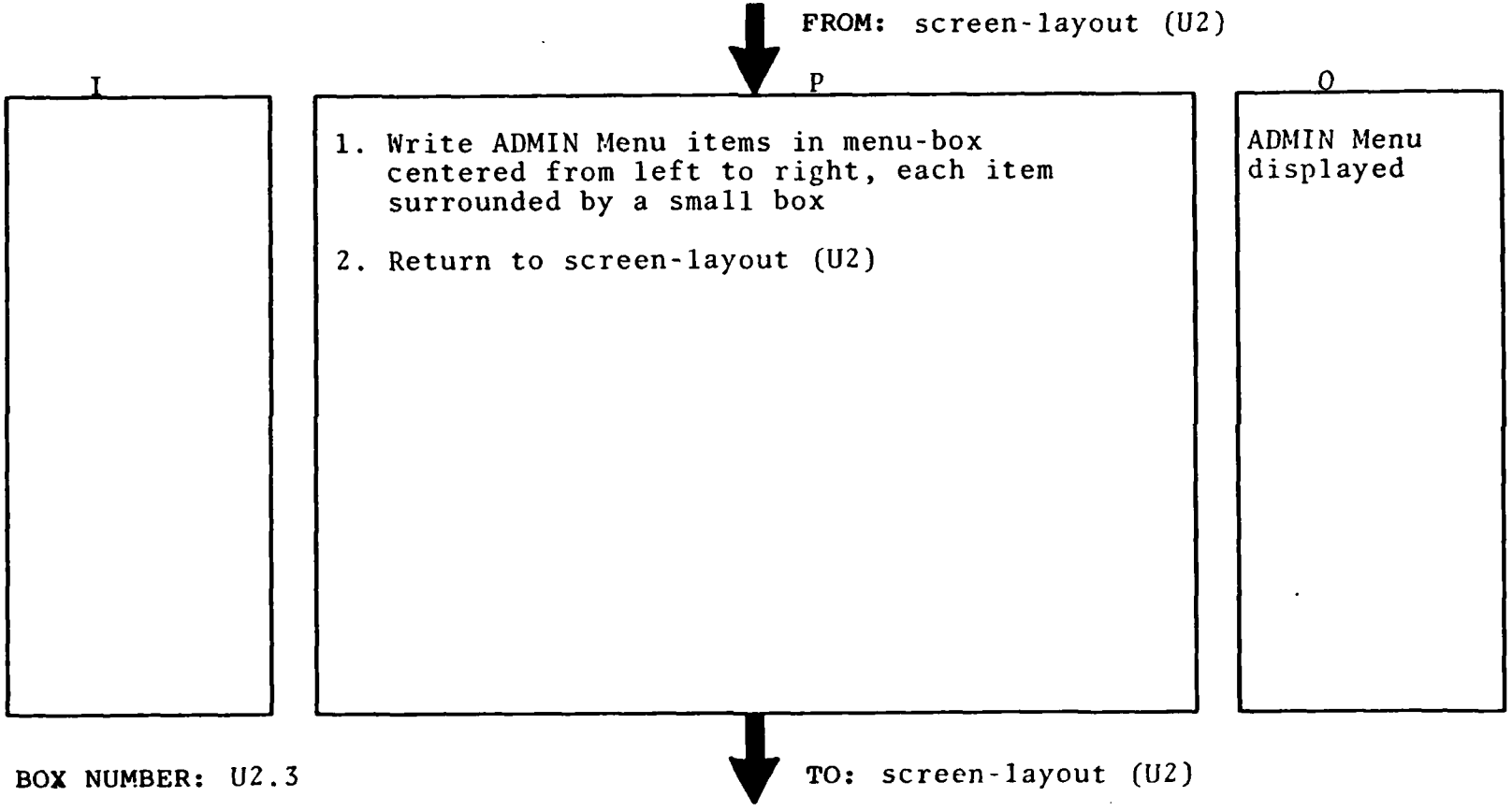


Figure 3.100 Draw-Admenu.

HIPO NOTES
for DRAW-ADMENU (U2.3)

Step	Note
	This module (and the following two) simply write the menu items in the menu-box. It is, however, important to note the boundaries of each item's small box because the user can select the item with a click of the mouse and the program must be able to tell if he is in a menu item box when the mouse is clicked.

Figure 3.101 Draw-Admenu.

HIPO OVERVIEW DIAGRAM
for DRAW-BRMENU (U2.4)

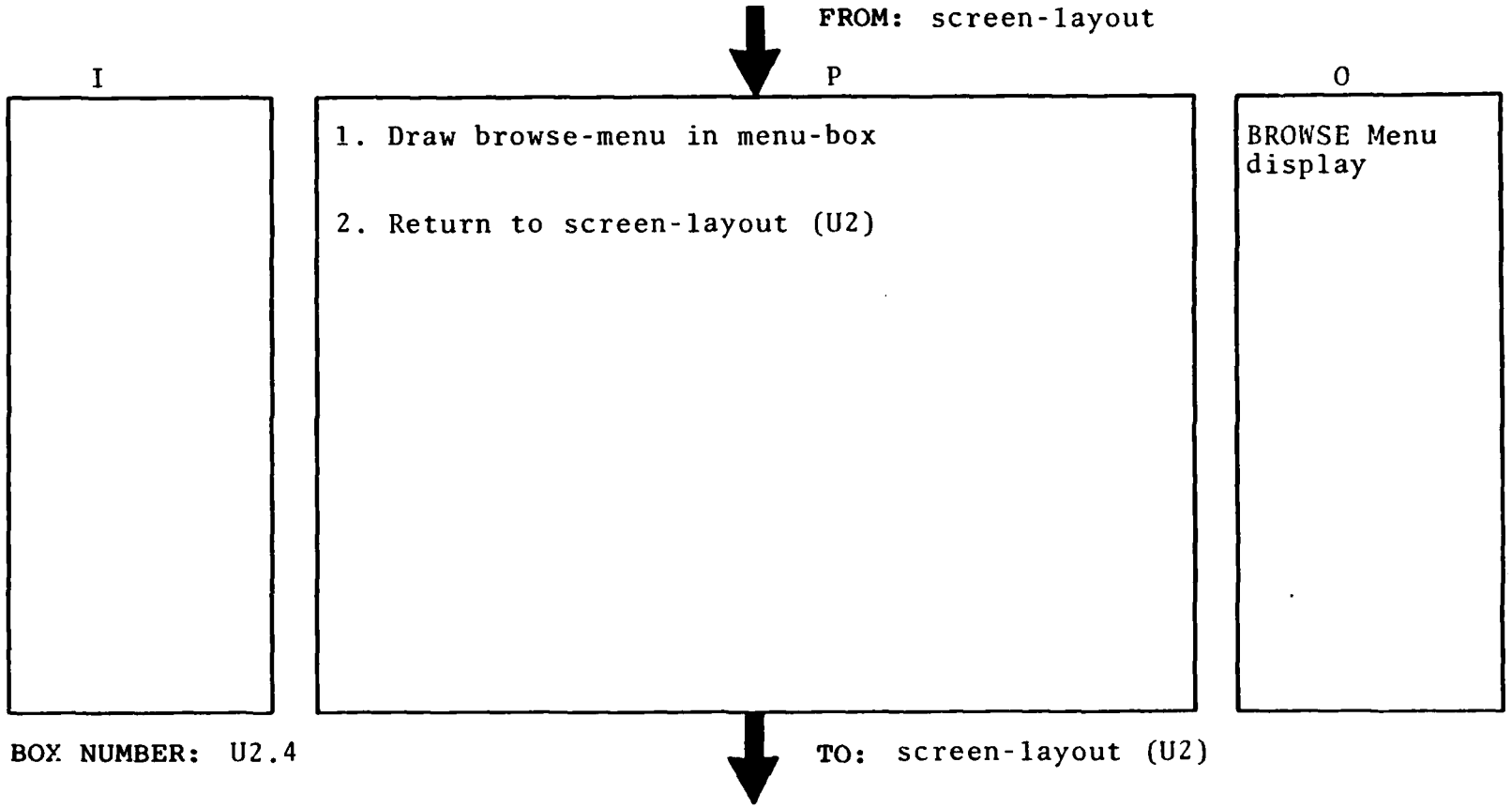


Figure 3.102 Draw-Brmenu.

HIPO DETAIL DIAGRAM

for DRAW-BRMENU (U2.4)

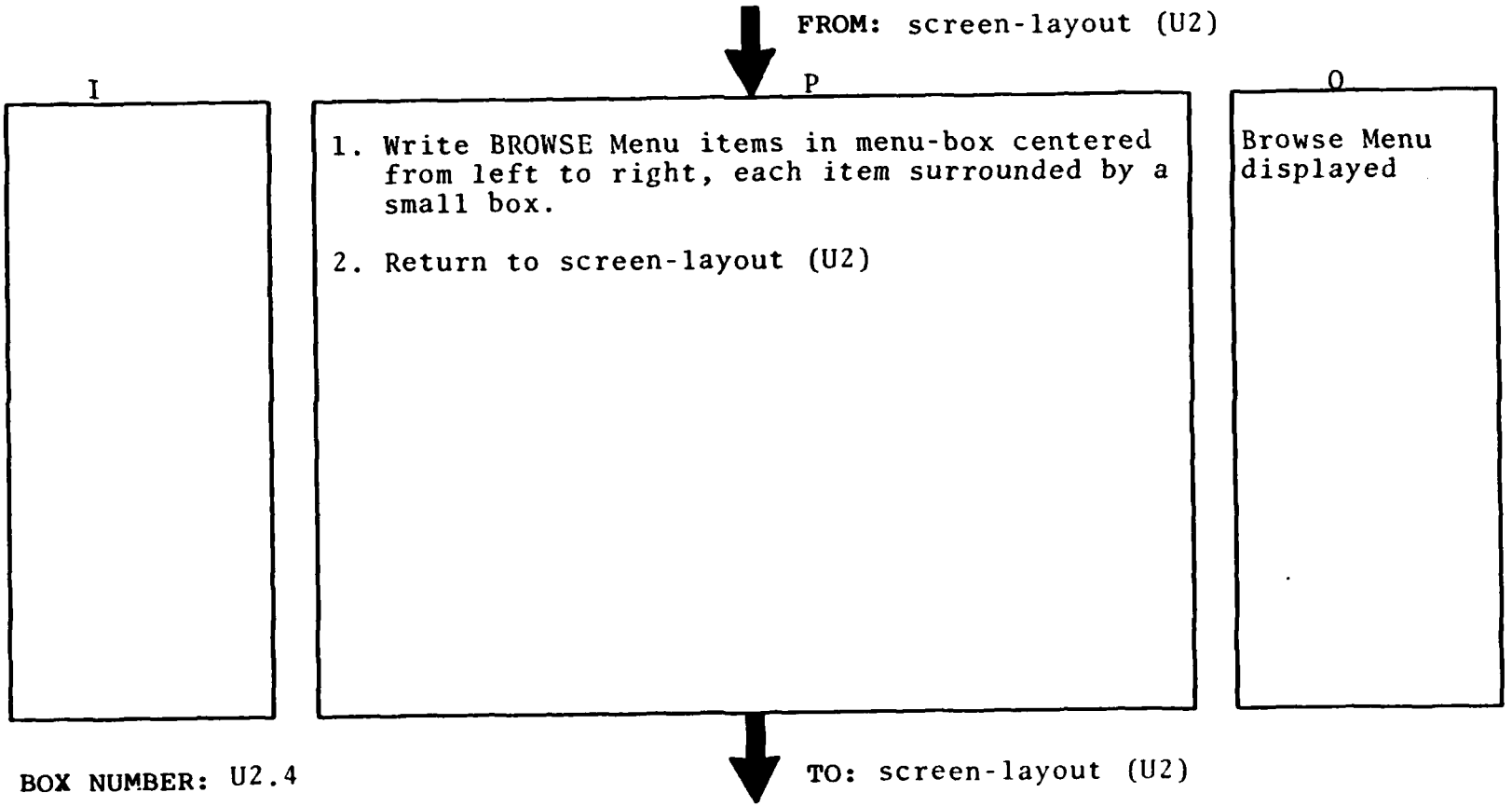


Figure 3.103 Draw-Brmenu.

HIPO OVERVIEW DIAGRAM
for DRAW-QUMENU (U2.5)

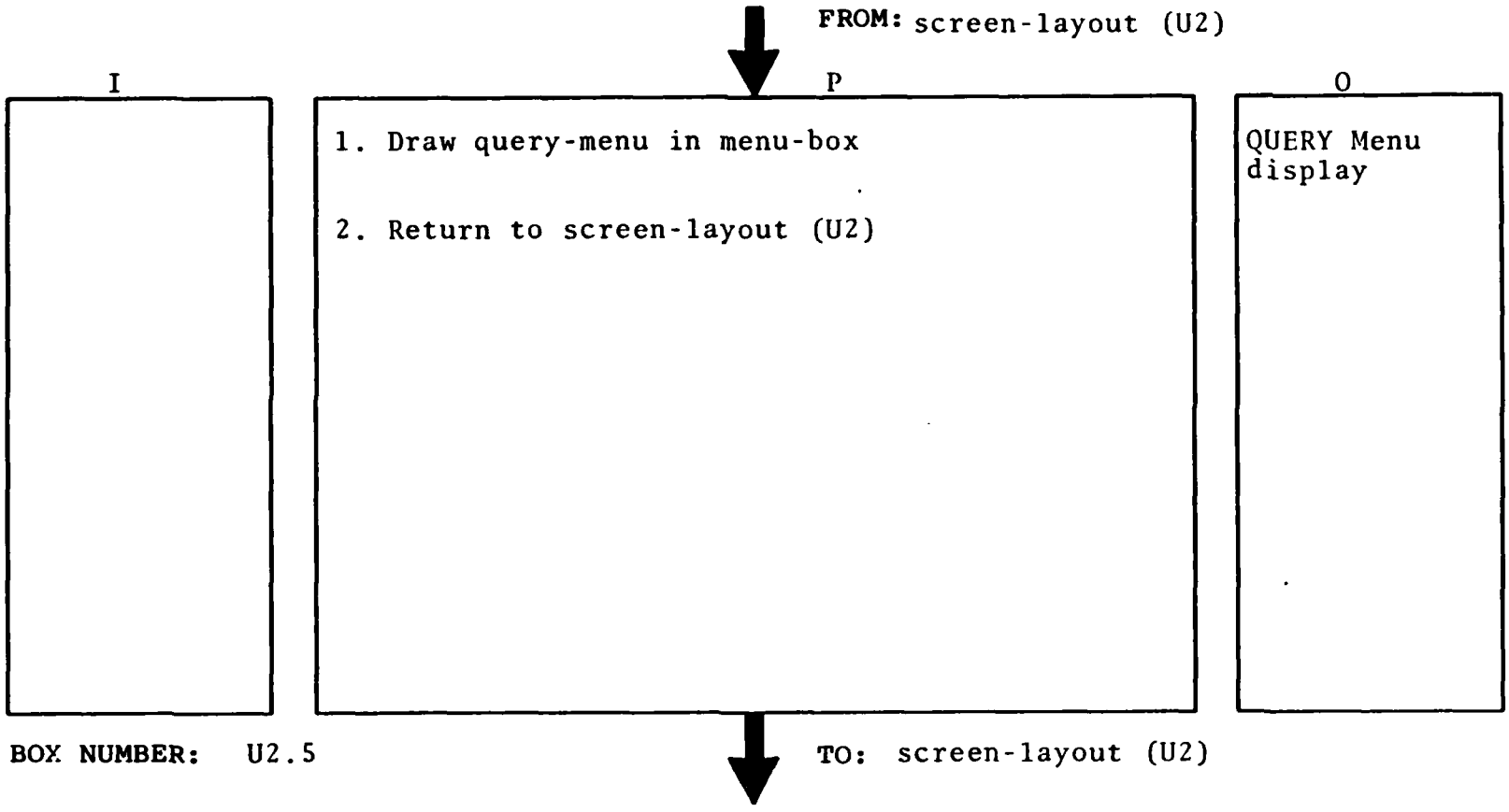


Figure 3.104 Draw-Qumenu.

HIPO DETAIL DIAGRAM
for DRAW-QUMENU (U2.5)

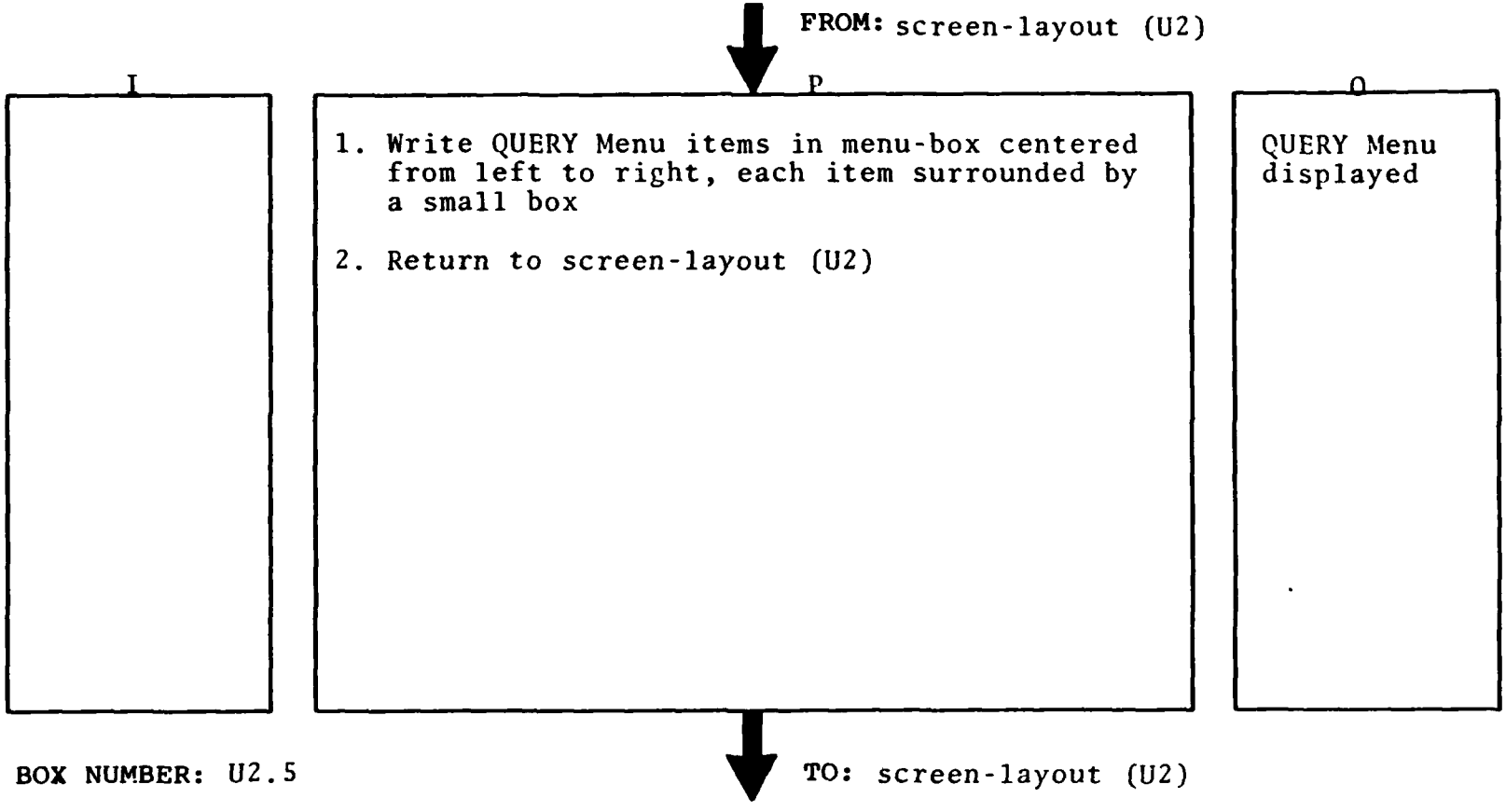


Figure 3.105 Draw-Qumenu.

HIPO OVERVIEW DIAGRAM
for ERASE-MENU (U2.6)

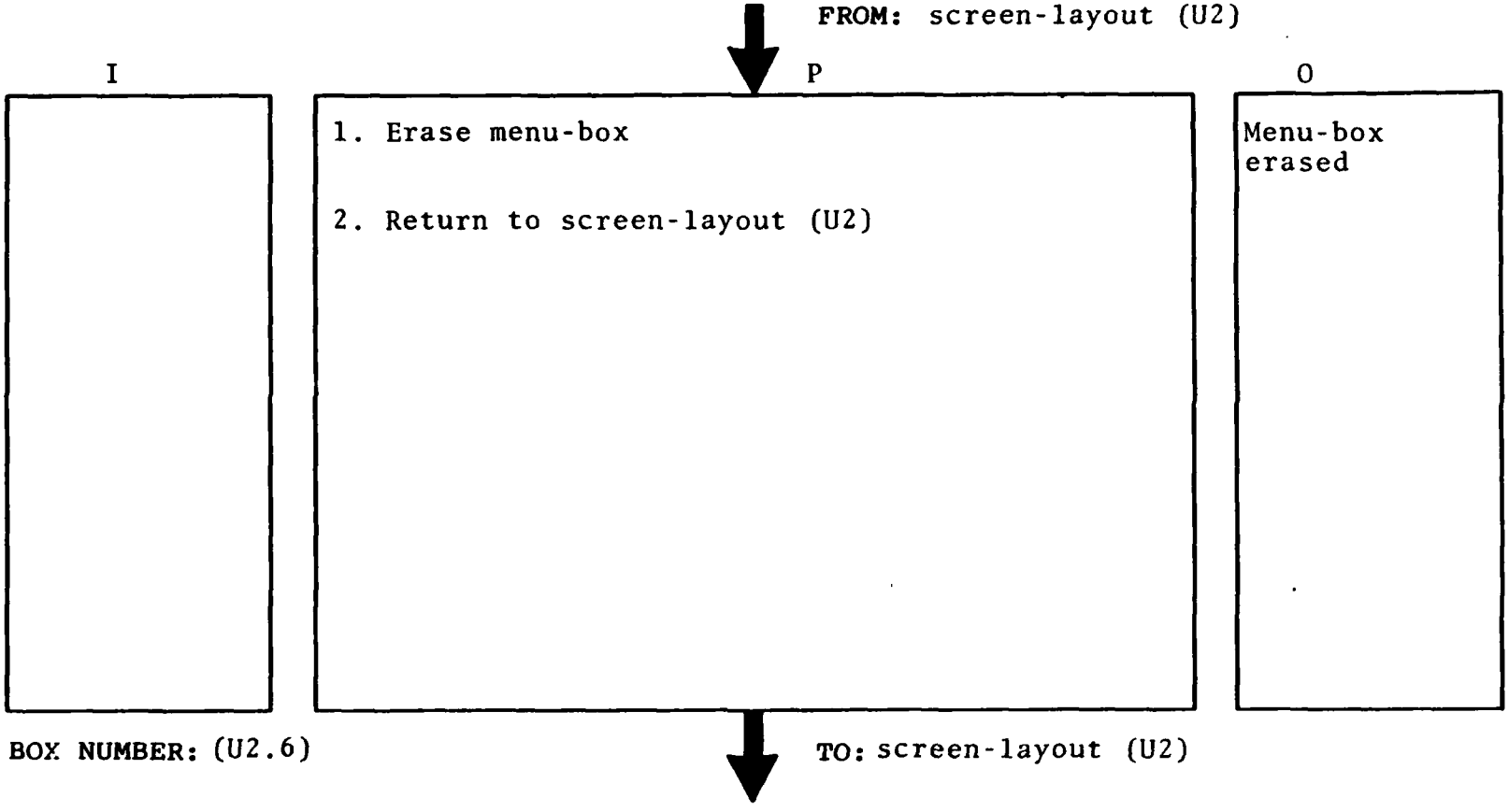


Figure 3.106 Erase-Menu.

HIPO DETAIL DIAGRAM
for ERASE-MENU (U2.6)

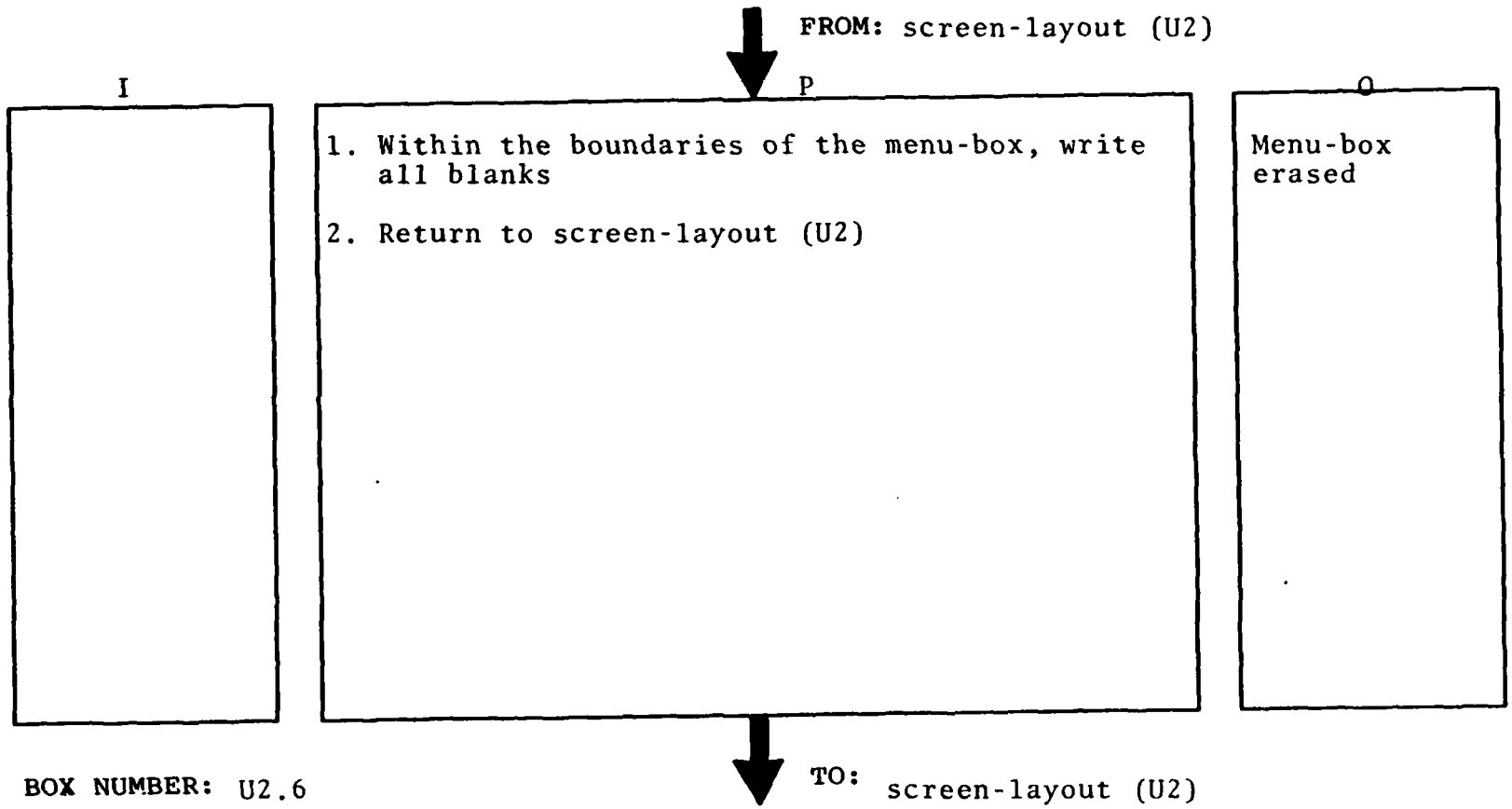


Figure 3.107 Erase-Menu.

HIPO NOTES
for ERASE-MENU (U2.6)

Step	Note
	The three "erase modules" are used to blank out the previous contents of a box/window. The reason that they all are defined by the "boundary" of the subject area rather than specific positions on the screen is that the boundaries of the areas to be erased can be altered by the use of the left "drag" mouse button, so you must make sure the entire area gets erased, not just the area initially defined by the program.

Figure 3.108 Erase-Menu.

HIPO OVERVIEW DIAGRAM
for ERASE-SCHEMA (U2.7)

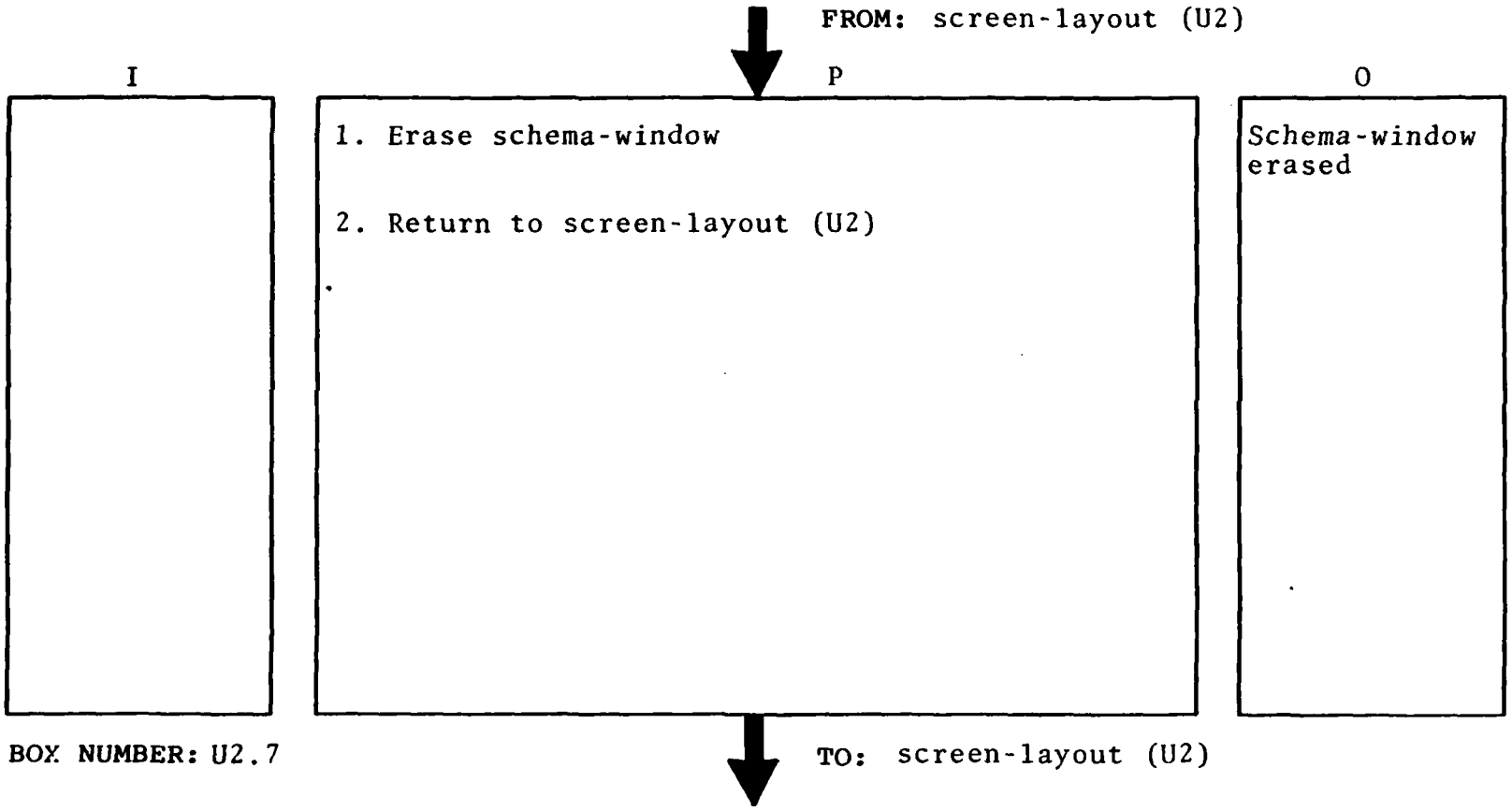


Figure 3.109 Erase-Schema.

HIPO DETAIL DIAGRAM

for ERASE-SCHEMA (U2.7)

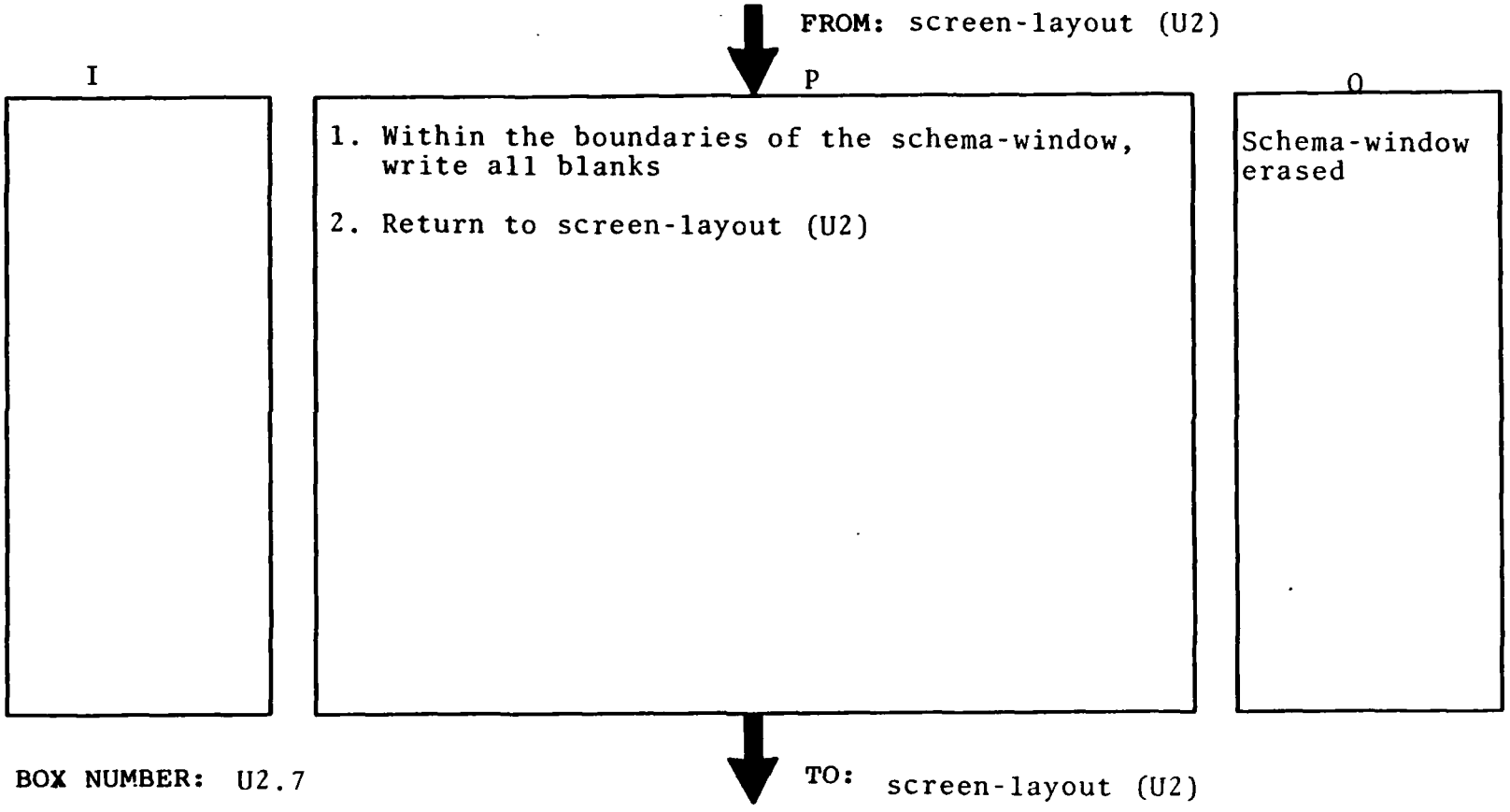


Figure 3.110 Erase-Schema.

HIPO OVERVIEW DIAGRAM
for ERASE-MISC (U2.8)

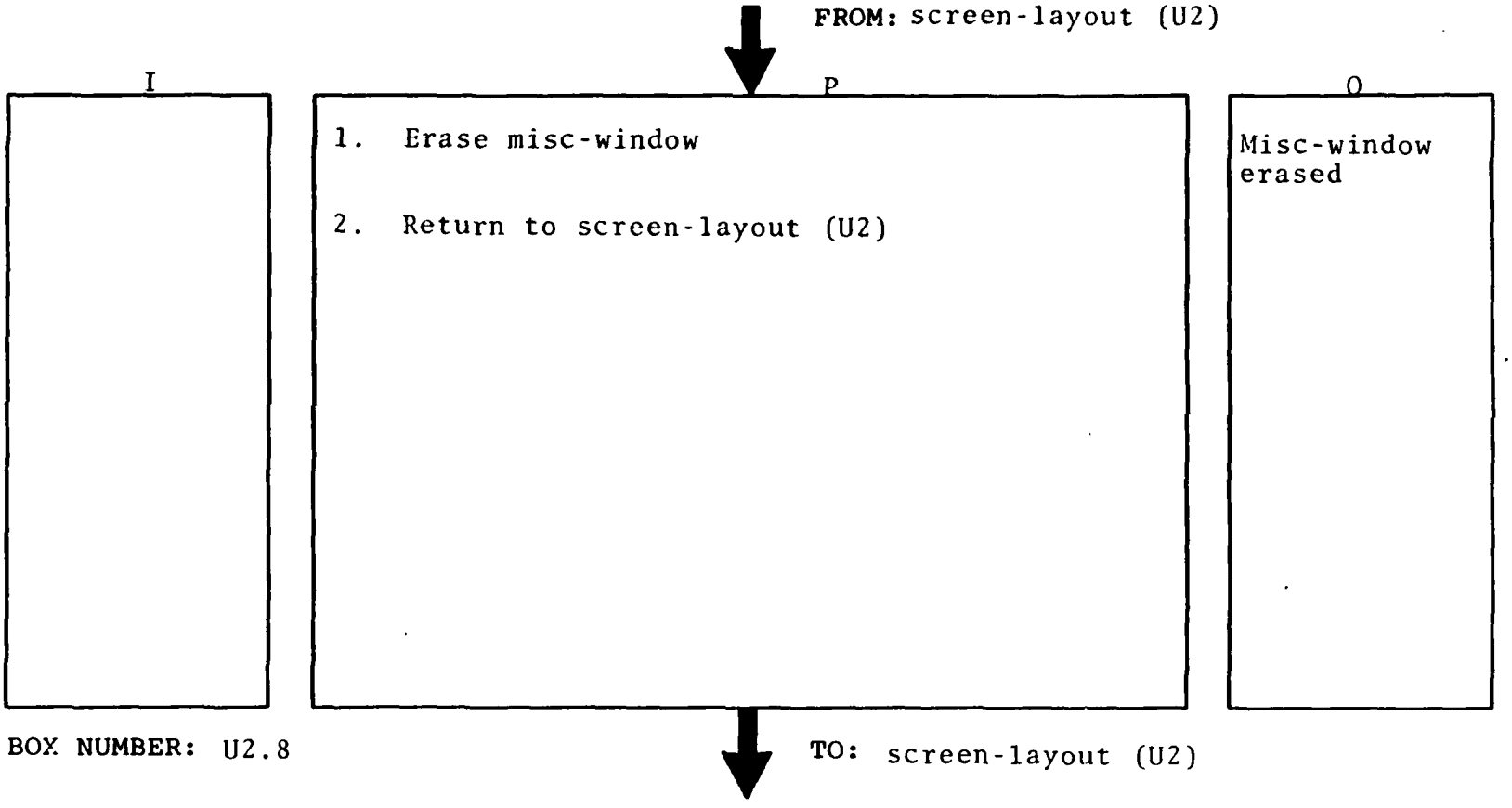


Figure 3.111 Erase-Misc.

HIPO DETAIL DIAGRAM
fpr ERASE-MISC (U2.8)

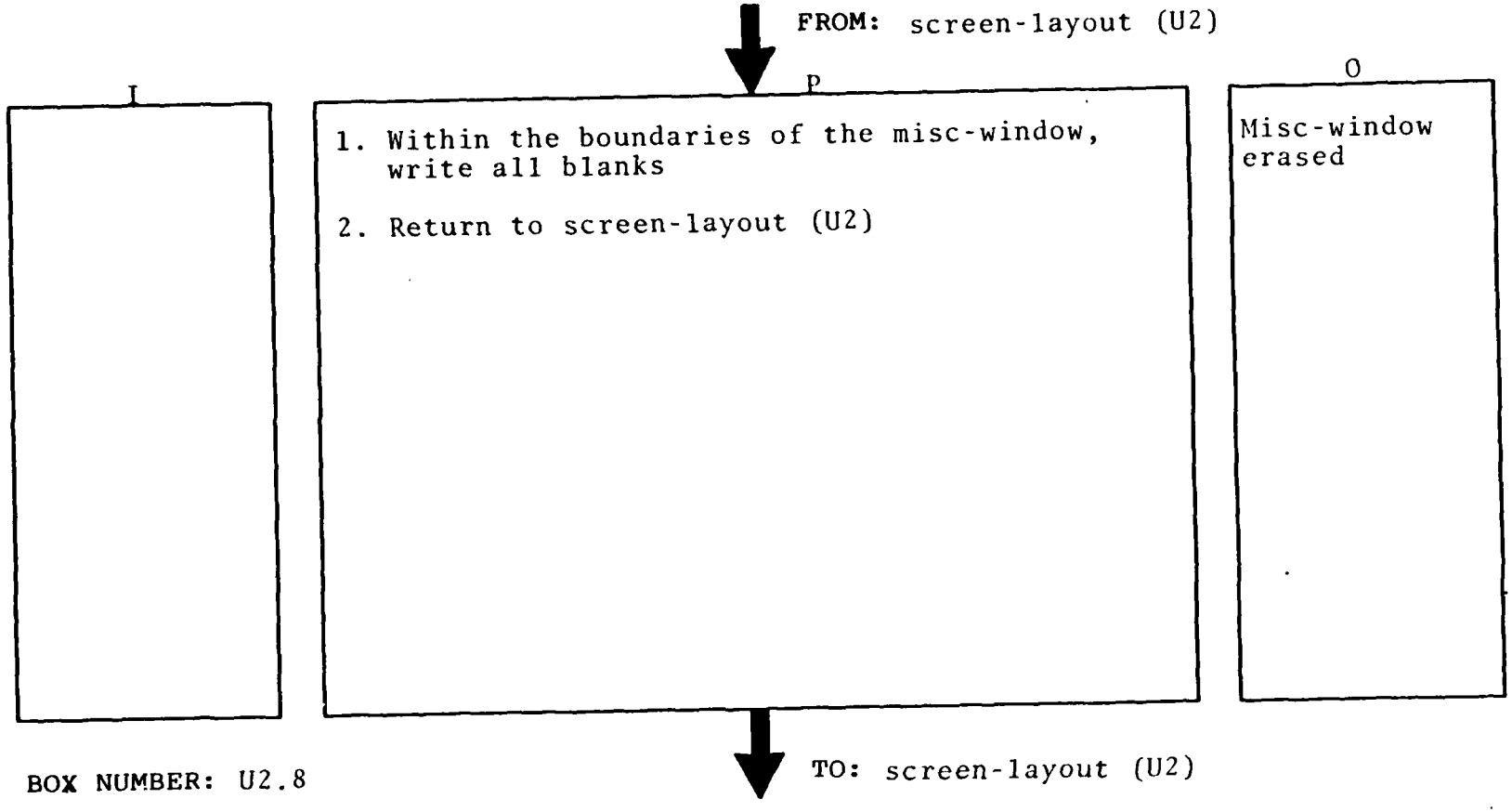


Figure 3.112 Erase-Misc.

LIST OF REFERENCES

(BOG 84) Boguraev, B. and Jones, K. "A Natural Language Front-end to Databases with Evaluative Feedback," New Applications of Databases, Gardarin, G. and Gelenbe, E., Academic Press, London, 1984, pp.159-182.

(COD 74) Codd, E., "Seven Steps to RENDEZVOUS With the Casual User," Proceedings IFIP TC-2 Working Conference on Data Base Management Systems, North-Holland Publishing Co., Amsterdam, 1974, pp. 179-200.

(DEA 82) Dean, M., "How a Computer Should Talk to People," IBM SYST Journal, Vol 21, No. 4, 1982, pp. 424-453.

(FAI 85) Fairley, R., Software Engineering Concepts, McGraw-Hill, 1985.

(HEN 77) Hendrix, G., "Human Engineering for Applied Natural Language Processing," Proceedings of Fifth International Joint Conference on Artificial Intelligence, Cambridge, 1977, pp. 183-191.

(HER 80) Herot, C., "Spatial Management of Data," ACM Transactions on Database Systems, Vol 5, No. 4, Dec 1980, pp. 493-514.

(IVE 82) Ives, B., "Graphical User Interfaces for Business Information Systems," MIS Quarterly Special Issue, 1982, pp. 15-47.

(KOR 84) Korth, H., Kuper, G., Feigenbaum, J., van Gelder, A., and Ullman, J., "System/U: A Database System Based on the Universal Relation Assumption," ACM Transactions on Database Systems, Vol 9, No. 3, Sep 1984, pp. 331-347.

(LAR 84) Larson, J., "The Forms Pattern Language," IEEE, Mar 1984, pp. 183-192.

(MAC 85) McGregor, R., "ARIEL--A Semantic Front-end to Relational DBMSs," Proceedings of VLDB, Stockholm, 1985, pp. 304-315.

(MCD 74) McDonald, N. and Stonebraker, M., "CUPID--the Friendly Query Language," Memo ERL-M487, ERL, University of California, Berkley, CA, Oct 1974.

(RAE 85a) Raeder, G., "Programming in Pictures," Technical Report No. 8/85, 22.02.85, 193 pages, Ph.D. Thesis.

(RAE 85b) Raeder, G., "A Survey of Current Graphical Programming Techniques," IEEE, Aug 1985, pp. 11-25.

(ROW 85) Rowe, L., "Fill-in-the-form Programming," Proceedings of VLDB, Stockholm, 1985, pp. 294-404.

(STO 82) Stonebraker, M. and Kalash, J., "TIMBER: A Sophisticated Relation Browser," Proceedings of VLDB, Stockholm, 1985, pp. 1-11.

(SUG 84) Sugihara, K., Miyao, J., Kikuno, T., and Yoshida, N., "A Semantic Approach to Usability in Relational Database Systems," Proceedings of IEEE International Conference on Data Engineering, Los Angeles, 1984, pp. 439-445.

(WAL 78) Waltz, D., "An English Language Question Answering System for a Large Relational Database," Communications of ACM, Vol 21, No. 7, Jul 1978, pp. 526-539.

(WON 82) Wong, H. and Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," Proceedings of VLDB, Stockholm, 1982, pp. 22-32.

(WU 85) Wu, C., "A New Graphics User Interface for Accessing a Database," submitted for publication, 1985.

(WU 86) Wu, C., "A Unified Interface Method for Interacting with a Database," submitted for publication, 1986.

(YOU 79) Yourdin, E. and Constantine, L., Structured Design: Fundamentals of a Discipline of Computer Program and System Design, Prentiss-Hall, 1979.

ZLO 77) Zloof, M., "Query-by-Example: A Database Language," IBM SYST Journal, No. 4, 1977, pp. 324-343.

BIBLIOGRAPHY

Bryce, D., and Hull, R., "SNAP: A Graphics-based Schema Manager," IEEE, June 1986, pp.151-164.

Finzer, W. and Gould, L., "Programming by Rehearsal," BYTE, June 1984, pp. 187-210.

Heiler, S., and Rosenthal, A., "G-WHIZ, A Visual Interface for the Functional Model with Recursion," Proceedings of VLDB, Stockholm, 1985, pp.209-218.

Hsiao, D., Systems Programming: Concepts of Operating and Database Systems, Addison-Wesley, 1975.

Moriconi, M. and Hare, D., "Visualizing Program Designs Through PegaSys," IEEE, Aug 1985, pp. 72-85.

Shu, N., "FORMAL: A Forms-Oriented, Visual-Directed Application Development System," IEEE, Aug 1985, pp. 38-49.

Ullman, J., Principles of Database Systems, Second Edition Computer Science Press, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor C. T. Wu Code 52Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Michael Zyda Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
5.	LT Jerry K. Adcock 15870 La Porte Court Morgan Hill, California 95037	1
6.	Computer Technology Programs Code 37 Naval Postgraduate School Monterey, California 93943-5000	1

END

DTIC

9-86