

UNCLASSIFIED

|   |
|---|
|   |
|   |
|   |
| AD NUMBER   |
| AD489666  |
| NEW LIMITATION CHANGE   |
| TO<br>Approved for public release, distribution unlimited   |
| FROM<br>Distribution authorized to U.S. Gov't. agencies and their contractors; Administrative/Operational Use; Aug 1966. Other requests shall be referred to Rome Air Development Center, Griffiss AFB, NY. |
| AUTHORITY   |
| RADC USAF ltr, 17 Sep 1971  |

THIS PAGE IS UNCLASSIFIED

489666

RADC-TR-66-474, Volume I  
Final Report



RELIABILITY CENTRAL AUTOMATIC DATA PROCESSING SUBSYSTEM

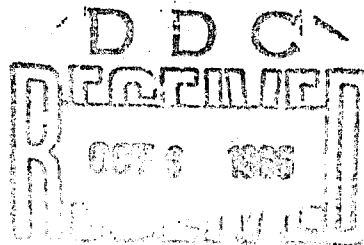
Design Specification Report

Auerbach Corporation

TECHNICAL REPORT NO. RADC-TR-66-474

August 1966

This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.



Rome Air Development Center  
Research and Technology Division  
Air Force Systems Command  
Griffiss Air Force Base, New York

Best Available Copy

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacturer, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

RELIABILITY CENTRAL AUTOMATIC DATA PROCESSING SUBSYSTEM

Design Specification Report

Auerbach Corporation

This document is subject to special  
export controls and each transmittal  
to foreign governments or foreign  
nationals may be made only with  
prior approval of RADC (EMLI),  
GAFB, N.Y. 13440.

FOREWORD

This three-volume final technical report was prepared by the Auerbach Corporation, Philadelphia 3, Pennsylvania under Contract AF 30(602)-3820, Project 5519. It is identified by the contractor as 1280-TR. The authors were Dr. J. Sable, W. Crowley, M. Rosenthal, S. Forst, and P. Harper. The Rome Air Development Center Project Engineer was Casper DeFiore, FMIID.

This technical report contains information embargoed from release to the Clearinghouse for Federal Scientific and Technical Information, Department of Commerce, by AFR 400-10.

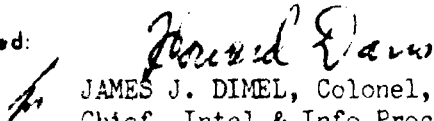
This technical report has been reviewed and is approved.

Approved:



FRANK J. TOMAINI  
Chief, Information Processing Branch

Approved:



JAMES J. DIMEL, Colonel, USAF  
Chief, Intel & Info Processing Division

FOR THE COMMANDER:

  
IRVING J. GABELMAN  
Chief, Advanced Studies Group

## ABSTRACT

This is Volume I of the three-volume final report produced for the Rome Air Development Center (RADC) under Contract AF30(602)-3620. Volumes I and II are the Design Specification Report for the Automatic Data Processing Subsystem (ADPS) of Reliability Central, known as Data Manager-1 (DM-1). Volume III is a survey of major, computer-oriented on-line information and fact retrieval systems.

The system design specification will be used for the implementation of the computer programs required to operate the RADC Reliability Central. The work reported in these volumes is an extension and detailing of the functional system design developed by AUERBACH Corporation under Contract AF30(602)-3433 and reported in RADC-TR-65-189, Design of Reliability Central Data Management Subsystem, July, 1965. The DM-1 design provides for the incorporation of the reliability data collected by the Illinois Institute of Technology Research Institute (IITRI) under Contract AF30(602)-3621 with AUERBACH Corporation as subcontractor.

This volume defines the DM-1 system, describes its use, specifies a major system components and discusses the management of the data pool and job library and the programming system services. Volume II contains a detailed description of the data pool and directories and the technical documentation and flowcharts for the system programs and jobs.

PREVIOUS PAGE WAS BLANK, THEREFORE WAS NOT FILLED

TABLE OF CONTENTS

| <u>PARAGRAPH</u>                           | <u>TITLE</u>  | <u>PAGE</u> |
|--|---|-------------|
| <u>SECTION I. INTRODUCTION</u>             |   |             |
| 1.1  | RELIABILITY CENTRAL DATA MANAGER-1 (DM-1) . . . . . | 1-1         |
| 1.2  | DESIGN SPECIFICATION REPORT . . . . .               | 1-3         |
| <u>SECTION II. DESIGN OBJECTIVES</u>       |   |             |
| 2.1  | RELIABILITY CENTRAL REQUIREMENTS . . . . .          | 2-1         |
| 2.2  | CONVENIENCE . . . . .                               | 2-1         |
| 2.3  | FUNCTIONAL RANGE . . . . .                          | 2-2         |
| 2.4  | ADAPTABILITY . . . . .                              | 2-4         |
| <u>SECTION III. SYSTEM CHARACTERISTICS</u> |   |             |
| 3.1  | USER'S VIEW . . . . .                               | 3-2         |
| 3.1.1                                      | Query Operations . . . . .                          | 3-4         |
| 3.1.2                                      | Interpretive Processing . . . . .                   | 3-5         |
| 3.1.3                                      | Sequence Execution . . . . .                        | 3-5         |
| 3.1.4                                      | Recurrent Jobs . . . . .                            | 3-5         |
| 3.2  | DATA ADMINISTRATOR'S VIEW . . . . .                 | 3-6         |
| 3.3  | PROGRAMMER'S VIEW . . . . .                         | 3-7         |
| 3.3.1                                      | Program Description . . . . .                       | 3-7         |
| 3.3.2                                      | Data Storage and Retrieval Services . . . . .       | 3-8         |
| 3.4  | INTEGRATED VIEW . . . . .                           | 3-9         |
| 3.4.1                                      | Data Base Development . . . . .                     | 3-9         |
| 3.4.2                                      | Job Library Development . . . . .                   | 3-10        |
| 3.4.3                                      | Operational Use . . . . .                           | 3-10        |
| <u>SECTION IV. DATA POOL FEATURES</u>      |   |             |
| 4.1  | DATA STRUCTURES . . . . .                           | 4-1         |
| 4.1.1                                      | Item Types . . . . .                                | 4-2         |
| 4.1.2                                      | Item Structure . . . . .                            | 4-4         |
| 4.2  | PRIMARY DIRECTORIES . . . . .                       | 4-5         |
| 4.2.1                                      | Item List . . . . .                                 | 4-8         |
| 4.2.2                                      | Term List . . . . .                                 | 4-8         |
| 4.2.3                                      | Term Encoding Table . . . . .                       | 4-8         |

TABLE OF CONTENTS (CONTD.)

| <u>PARAGRAPH</u> | <u>TITLE</u>                             | <u>PAGE</u> |
|------------------|--|-------------|
| 4.3              | DATA REPRESENTATION . . . . .            | 4-11        |
| 4.3.1            | Data Stream . . . . .                    | 4-11        |
| 4.3.2            | Segmentation . . . . .                   | 4-13        |
| 4.3.3            | Segment Index . . . . .                  | 4-15        |
| 4.4              | SAMPLE RETRIEVAL . . . . .               | 4-15        |
| 4.4.1            | Name Translation . . . . .               | 4-17        |
| 4.4.2            | Search Strategy . . . . .                | 4-18        |
| 4.4.3            | Data Segment Retrieval . . . . .         | 4-19        |
| 4.4.4            | Data Stream Interpretation . . . . .     | 4-20        |
| 4.5              | INDEXING . . . . .                       | 4-20        |
| 4.6              | LINKAGE . . . . .                        | 4-22        |
| 4.7              | DATA INTEGRITY . . . . .                 | 4-26        |
| 4.7.1            | Security Safeguard . . . . .             | 4-26        |
| 4.7.2            | Validity Safeguard . . . . .             | 4-27        |
| 4.7.3            | Item Lockout (Busybit) . . . . .         | 4-27        |
| 4.7.4            | General Procedure . . . . .              | 4-28        |
| 4.8              | THE DATA POOL . . . . .                  | 4-28        |
| 4.8.1            | Structure Development . . . . .          | 4-30        |
| 4.8.2            | Data Manipulation . . . . .              | 4-31        |
| 4.9              | CONDITIONAL RETRIEVAL . . . . .          | 4-32        |
| 4.10             | THE DIALOGUE . . . . .                   | 4-33        |
| 4.10.1           | Uses of the Dialogue Procedure . . . . . | 4-34        |
| 4.11             | RESTRUCTURING ITEMS . . . . .            | 4-35        |

SECTION V. OPERATIONAL FEATURES

|       |  |      |
|-------|--|------|
| 5.1   | SYSTEM LANGUAGE SPECIFICATION . . . . .    | 5-1  |
| 5.1.1 | The Metalanguage . . . . .                 | 5-2  |
| 5.1.2 | External Definition Languages . . . . .    | 5-4  |
| 5.1.3 | Use of INSCAN with an Item Image . . . . . | 5-7  |
| 5.1.4 | Data Language . . . . .                    | 5-10 |
| 5.1.5 | The Job Request Language . . . . .         | 5-15 |
| 5.2   | JOB-PARAMETER BINDING . . . . .            | 5-18 |
| 5.2.1 | Programs and Jobs . . . . .                | 5-18 |
| 5.2.2 | Binding Specification . . . . .            | 5-21 |



TABLE OF CONTENTS (CONT'D.)

| <u>PARAGRAPH</u>   | <u>TITLE</u>                                   | <u>PAGE</u> |
|--|--|-------------|
| 5.3  | RELATIONSHIP TO THE OPERATING SYSTEM . . . . . | 5-21        |
| 5.4  | REQUEST PROCESSOR. . . . .                     | 5-24        |
| 5.4.1  | Request Translation . . . . .                  | 5-24        |
| 5.4.2  | Job Extension . . . . .                        | 5-26        |
| 5.4.3  | The Request Record . . . . .                   | 5-27        |
| 5.5  | JOB MANAGER . . . . .                          | 5-28        |
| 5.6  | SERVICE PACKAGE. . . . .                       | 5-31        |
| <br><u>SECTION VI. DATA—POOL MANAGEMENT</u><br>                    |  |             |
| 6.1  | DATA INDEPENDENCE . . . . .                    | 6-2         |
| 6.2  | DIRECTORY MANIPULATION . . . . .               | 6-2         |
| 6.3  | DATA MANIPULATION. . . . .                     | 6-4         |
| 6.4  | PREPARATION FOR USERS . . . . .                | 6-5         |
| 6.5  | USAGE STATISTICS . . . . .                     | 6-6         |
| <br><u>SECTION VII. PROGRAM PARAMETERS AND THE JOB LIBRARY</u><br> |  |             |
| 7.1  | PARAMETER CATEGORIES . . . . .                 | 7-1         |
| 7.1.1  | Generalized Parameters . . . . .               | 7-2         |
| 7.1.2  | Specific Parameters . . . . .                  | 7-4         |
| 7.1.3  | Parameter-Binding Examples . . . . .           | 7-6         |
| 7.2  | PROGRAM ENTRY. . . . .                         | 7-9         |
| 7.3  | JOB DESCRIPTION . . . . .                      | 7-10        |
| 7.3.1  | Component Input Parameter Binding . . . . .    | 7-11        |
| 7.3.2  | Component Output Parameter Binding . . . . .   | 7-12        |
| 7.3.3  | Parameter-Binding Choices . . . . .            | 7-13        |
| 7.4  | THE JOB DESCRIPTION LIBRARY . . . . .          | 7-13        |
| 7.5  | JOB AND PROGRAM DELETION . . . . .             | 7-15        |
| 7.6  | LIBRARY DISPLAY . . . . .                      | 7-16        |

TABLE OF CONTENTS (CONTD.)

| <u>PARAGRAPH</u>                                 | <u>TITLE</u>                   | <u>PAGE</u> |
|--|--------------------------------|-------------|
| <u>SECTION VIII. PROGRAMMING SYSTEM SERVICES</u> |                                |             |
| 8.1  | DATA ACCESS SERVICES . . . . . | 8-2         |
| 8.1.1  | Open-For-Input . . . . .       | 8-2         |
| 8.1.2  | Read . . . . .                 | 8-4         |
| 8.1.3  | Seek . . . . .                 | 8-5         |
| 8.1.4  | Close-For-Input . . . . .      | 8-5         |
| 8.1.5  | Retrieve-Item . . . . .        | 8-6         |
| 8.2  | WRITING SERVICES . . . . .     | 8-6         |
| 8.2.1  | Open-For-Writing . . . . .     | 8-7         |
| 8.2.2  | Write . . . . .                | 8-7         |
| 8.2.3  | Close-For-Writing . . . . .    | 8-8         |
| 8.2.4  | Insert-Data . . . . .          | 8-8         |
| 8.3  | UPDATING SERVICES . . . . .    | 8-8         |
| 8.3.1  | Open-For-Updating . . . . .    | 8-9         |
| 8.3.2  | Read, Seek . . . . .           | 8-9         |
| 8.3.3  | Insert . . . . .               | 8-9         |
| 8.3.4  | Replace . . . . .              | 8-10        |
| 8.3.5  | Delete . . . . .               | 8-10        |
| 8.3.6  | Seek-With-Copy . . . . .       | 8-10        |
| 8.3.7  | Close-For-Update . . . . .     | 8-10        |
| 8.3.8  | Replace-Item . . . . .         | 8-11        |
| 8.3.9  | Delete-Item . . . . .          | 8-11        |

## LIST OF ILLUSTRATIONS

| <u>FIGURE</u> | <u>TITLE</u>  | <u>PAGE</u> |
|---------------|---|-------------|
| 3-1           | DM-1 System Overview. . . . .                               | 3-1         |
| 3-2           | System Operation, Block Diagram . . . . .                   | 3-2         |
| 4-1           | Tree-Structure Representations . . . . .                    | 4-1         |
| 4-2           | Purchasing Item Structure. . . . .                          | 4-2         |
| 4-3           | Sample Structure. . . . .                                   | 4-3         |
| 4-4           | Data Maze . . . . .   | 4-4         |
| 4-5           | Directories for the Purchasing Item . . . . .               | 4-5         |
| 4-6           | Tree and Network Diagrams for the Purchasing Item . . . . . | 4-6         |
| 4-7           | Purchasing Item with Links . . . . .                        | 4-7         |
| 4-8           | The Data Pool. . . . .                                      | 4-8         |
| 5-1           | Syntactic Chart . . . . .                                   | 5-1         |
| 5-2           | Structure Diagrams with Item Images . . . . .               | 5-2         |
| 5-3           | Syntactic Chart for Item Image. . . . .                     | 5-3         |
| 5-4           | EDL Syntactic Chart. . . . .                                | 5-4         |
| 5-5           | Job-Run Request Syntactic Chart . . . . .                   | 5-5         |
| 5-6           | Jobs as Black Boxes. . . . .                                | 5-6         |
| 5-7           | Request Translation . . . . .                               | 5-7         |
| 5-8           | Transitions Controlled by the Job Manager. . . . .          | 5-8         |
| 7-1           | Categories of Input-Output Parameters . . . . .             | 7-1         |
| 7-2           | Example of Parameter Binding. . . . .                       | 7-2         |
| 7-3           | Job Construction Procedure. . . . .                         | 7-3         |
| 7-4           | Intermediate Form of the Job FAILURE ANALYSIS . . . . .     | 7-4         |
| 8-1           | Interfaces of the DM-1 Services. . . . .                    | 8-1         |
| 8-2           | Sample Buffer Description List . . . . .                    | 8-2         |

LIST OF TABLES

| <u>TABLE</u> | <u>TITLE</u>  | <u>PAGE</u> |
|--------------|---|-------------|
| 4-1          | DEFINITION FOR THE PURCHASING ITEM.....             | 4-6         |
| 4-2          | TERM LIST AND ITEM LIST.....                        | 4-9         |
| 4-3          | TERM ENCODING TABLE .....                           | 4-10        |
| 5-1          | STRUCTURE OF THE REQUEST FILE .....                 | 5-27        |
| 8-1          | STRUCTURE DEFINITION FOR PURCHASING ORDERS FILE ... | 8-3         |

## SECTION I. INTRODUCTION

### 1.1 RELIABILITY CENTRAL DATA MANAGER-1 (DM-1)

DM-1, a computer-based software system, is designed for the Rome Air Development Center (RADC) to operate as the Automatic Data Processing Subsystem (ADPS) of Reliability Central. The DM-1 system will operate in a time-shared mode on the Reliability Central data processing facility in conjunction with that facility's operating system. The mission of Reliability Central, as a central activity for the acquisition, analysis, dissemination, and use of reliability information, presents an imposing requirement for data processing resources. DM-1 satisfies this requirement by providing a set of data processing elements which cohere in an integrated system. The major elements of the DM-1 system are:

- (1) A rationale for the structuring of a large, dynamic data base. The reliability data gathered by Reliability Central can be integrated into the DM-1 data pool, under structural specifications which retain the individuality of the diverse data items while manifesting the relationships among the items.
- (2) A repertoire of generalized system operations that provide for the management of the DM-1 data pool. The reliability analysts can modify the data and its descriptive parameters to accommodate new data elements, or new relationships, or to adjust to changing requirements and operational experience.

- (3) A mechanism for retrieving selected data to meet specified information needs. The Reliability Central staff can access the precise data items needed to meet on-demand requirements for information. DM-1 can display the results of an inquiry issued at a console, develop a data item with specified characteristics for further processing (reliability analyses), or deliver the specified data to a program operating with the system.
- (4) A procedure which assists an inquirer in defining his information needs. A Reliability Central user can perform a dialogue with the system. He chooses the pertinent attributes to describe the item about which he needs information from displays presented by the system. Each display identifies the classes of information available in the DM-1 data pool. The displays proceed from generic to specific identifiers in response to the inquirer's selections. After the attributes are defined, the user provides limiting conditions which define the properties of the individual units of information he needs, by selecting characteristics to define the pertinent data from another series of displays.
- (5) A library of application-oriented programs and jobs. The Reliability Central can add new programs to the DM-1 repertoire to perform special-purpose reliability processes on the data. New units of work (jobs) may be defined as combinations of existing system and application programs to meet recurring needs for data processing.
- (6) A mechanism for selecting data processing routines from the library and applying them to specified data elements in the data pool. The reliability analyst can issue commands from a console for the execution of any job in the DM-1 library. He can specify the data items in the data pool to be operated on by the job.
- (7) A method of maintaining data and programs as independent, mutually complementary resources. The reliability data base need not be oriented to any specific set of programs. The reliability application programs need not be limited to operation on specific items of data. DM-1 maintains structural information about the data in its directories and the data requirements of the programs in its library. If the requirements of the program fail to match the characteristics of the data, the system can transfer the data to the format required by the program.
- (8) A set of system routines to control the execution of DM-1 jobs and to service the data needs of programs during their operation.

- (9) A mechanism to provide for data integrity and security. Reliability Central users are protected from unauthorized access to their data.

## 1.2 DESIGN SPECIFICATION REPORT

This report specifies the Automatic Data Processing Subsystem (ADPS) of Reliability Central. The general-purpose nature of the data processing requirements of Reliability Central and the anticipated need for adjustments in its requirements predicate a need for a general-purpose solution. This report describes the Data Manager-1 (DM-1) as the initial basis for that solution.

The design specification for DM-1 is presented in two volumes. This first volume describes the system as a whole. It defines the objectives of the design and the characteristics of the system. It describes the system's features in terms of its two major aspects - as the manager of the Reliability Central data base and as an operational tool for the accomplishment of data processing objectives. This volume also presents the DM-1 components which support the Reliability Central staff in managing the data pool and the job library and which support the programmers in accessing and storing data in the system data pool.

Volume II contains the technical documentation on the system's components. It describes the detailed structure of the system data pool and directories, which provide for efficient handling of large Reliability Central files. The second volume also contains flowcharts and descriptions for the DM-1 system routines and jobs, specifically:

- (1) The routines of the Service Package which provide the interface between operating programs and the data pool.
- (2) The DM-1 Supervisor, including the Request Processor which responds to user requests from a console and the Job Manager which manages the flow of control among programs.
- (3) The system jobs which provide for the management of the job library, including the Program Entry and Job Description jobs and jobs for deleting and displaying job descriptions.
- (4) The system jobs which provide for maintenance of the data pool, including the jobs for describing and deleting data structures and the jobs for adding, deleting, modifying, and updating data in the data pool.

- (5) The system utility jobs, including Query and Conditional Search for retrieving relevant data from the data pool, conditional Reformat for subsetting and restructuring data items and Display for transforming internal data items to an external form for presentation to users.



## SECTION II. DESIGN OBJECTIVES

### 2.1 RELIABILITY CENTRAL REQUIREMENTS

The specification of the ADPS requirements is derived from the mission requirements of Reliability Central. Thus, the ADPS must have the ability not only to gather data and retrieve it selectively but also to make this data available to powerful analytic tools in the form of computer programs. The Reliability Central ADPS cannot be a system for the storage and retrieval of rigidly formatted documents. It must be capable of answering information needs by supplying facts which may depend on complex interrelationships within the data; and it must be prepared to adapt to changing requirements as Reliability Central evolves. The design objectives discussed in the following paragraphs are derived from the requirements of Reliability Central.

### 2.2 CONVENIENCE

DM-1 must be a service tool for a variety of Reliability Central users. The users will range from those who wish to use the system without learning anything about it to those who wish to be experts in the manipulation of its inner workings. Consequently, the system will be driven by a console-based user, with a spectrum of tools at his command.

The user may obtain system guidance in specifying his requirements through a multistage dialogue, in which he responds to displays generated by the system. He may issue requests to the system in a user-oriented language which shields him from the complexities of the system while giving him the power to perform useful work.

The more advanced user may specify his requirements directly to the system. He can issue complex requests that involve conditional selections and restructuring of data in the course of job execution. He may modify the degree of system control over specified data to enhance its performance in certain operations. He can define new operations for entry into the DM-1 library. Even for the expert, the interface with the system must be as simple and convenient as possible.

### 2.3 FUNCTIONAL RANGE

DM-1 will contain the full set of data processing capabilities associated with the data base of Reliability Central.

The variety and complexity of reliability data demand the ability to specify complex structures showing the interrelationships among data items. The dynamic character of the data must be accommodated by system tools which permit changes in data structures and data content. DM-1 supplies a series of system jobs that permit the Reliability Central staff to manage the data pool without the use of special programming.

The Reliability Central requirement for data reduction and analysis must be met by a library of data processing and analytic programs capable of operating on the data under DM-1 control. General-purpose programs capable of operating on any reasonable data in the reliability data base are needed. They should not be tied to narrow applications with specific data. The Reliability Central staff must be able to add new programs and jobs to the DM-1 library to meet data processing requirements as they arise. The core library of data processing and analytic programs supplied by DM-1 must be expandable by the addition of programs developed by Reliability Central.

DM-1 system jobs should provide the ability to accept new programs. The Reliability Central staff must be able to define jobs in terms of existing system and user programs and jobs in order to tailor the library to evolving requirements. With this capability, the jobs which produce the scheduled information products of Reliability Central can be defined and added to the library.

Reliability information needs will be met by DM-1. A user should be able to state an information need in a convenient language at a console and have the system retrieve and display the data which answers the need. DM-1 must supply a system language which permits the user to specify the attributes of the object, event, or process of interest and the properties (conditions) which define the individual elements about which he wants information. When his information need is not well defined, or if he is not familiar with the information available or the rules for specifying his need, the user can obtain system guidance by performing a dialogue.

Reliability Central will be faced with more complex information requirements that involve the selection of data from various parts of the data base, its analysis or reduction by programs and jobs, and the presentation of the results. DM-1 must provide the user with the ability to define the information to be processed, the job to process it, and the results to be displayed. When the steps to be taken depend on the results of analytic processes, the user may define and execute a step at a time, display or query its results, and determine the next step.

Reliability Central users must have immediate access to the services of DM-1. The system must be capable of serving competing users simultaneously. It should be designed to operate in a multiprogrammed environment where it can service many users in parallel. Each user should see the system as dedicated to him while he is using it.

Many data elements in the reliability data pool will contain proprietary information. DM-1 must protect Reliability Central users and contributors from unauthorized access to such data and provide for the protection of the data from modification by unauthorized users or collisions in usage.

## 2.4 ADAPTABILITY

All systems exist in an environment of change. DM-1 must be able to adapt to changes and continue to give useful service to Reliability Central as it evolves.

One element of adaptability relates to the DM-1 software. Its design is modular so that it can be expanded with minimum cost and effort. Its system components are general-purpose programs so that they may be applied when unforeseen uses arise. In addition, the system contains a built-in capability to adapt to changes in data and processing requirements. The library of jobs and the data pool are managed as independent resources. Data structures and data content may be changed by system jobs as the need arises, without obsoleting the programs which operate on them. New programs may be added to the library. New operations (jobs) may be defined in terms of existing programs and jobs, even when these components were written to process data in structures that differ from the ones to be operated on by the new operation. With these tools, the reliability data base and the job library can be adapted to accommodate changes in the data base or the processing requirements of Reliability Central.

The other element of adaptability relates to hardware and the operating environment. DM-1 has been designed to take as little account of equipment idiosyncrasies as possible. Its data pool is stored in a device-independent format on a variety of random-access devices. The logic of its processes is device- and computer-independent, and the processes will be coded in a procedure-oriented language wherever possible. The interfaces between DM-1 and the equipment and the operating system are designed to be as narrow as possible. For example, all system processes are performed with logical data pool items as their input and output so that only the service routines, which deal directly with the environment, take account of the features of the operating system and the equipment.

### SECTION III. SYSTEM CHARACTERISTICS

Programs and data are two prime resources of a data processing center. DM-1 contains a collection of procedures, executive functions, services, and programs that views data and programs as independent entities whose properties complement each other. The same data is useful from many viewpoints, it should be available for use in different applications and for processing by unrelated programs. The same program is a valuable component in many operations; it should be available for use in different applications to operate on unrelated data. DM-1 meets these objectives by using a system directory to separate the structure of the data from the logic of the programs.

When focusing on data, the information management aspects of DM-1 predominate. The system provides a rationale for structuring data and a set of tools for managing and querying a common data pool. The item definition language and associated maintenance jobs provide for naming data items (fields, files, or combinations), defining their structure, expressing the relationships among them, and changing these parameters to meet the needs of a large, constantly varying data pool. The data languages and related maintenance jobs provide for entering data, manipulating it, modifying it, and changing its structure to meet changing requirements. The query language and the

conditional search, query, dialog, and display jobs provide for determining the significant items, defining the significant data, retrieving it, and displaying it to meet both predictable and unscheduled needs for information.

When focusing on programs, the operational aspects of DM-1 predominate. The system provides program and job library services, symbolic binding of program input and output parameters, data storage and access services, and job execution control. A program description, which specifies the input and output requirements of the program, is maintained for every program in the system. Complex jobs may be described as sequences of programs and jobs with binding specifications to relate the inputs and outputs of the components. Any program or job may function as a task, job, or subroutine. As a job, it may be executed by a command from a user's console. As a task, it may be used as a component in a job description. As a subroutine, it may be executed by a call from within a program.

Programs and data remain as independent resources to be combined as the need arises. DM-1 maintains information about the structure of the data in the system directories; it maintains information about the input and output requirements of the programs in the libraries; it has the ability to transform the existing data to meet the requirements of the programs.

Figure 3-1 is a simplified overview of the system. It shows the conversion of external data to the system data pool through the data collection and conversion jobs and the development of the job library through the program entry and job description jobs. With these data and program resources available, the user may request the execution of an operation from the library on a data set from the data pool. The System Supervisor responds to the user's request, sets system parameters to guide the selection of data, and oversees the execution of the sequence of tasks requested by the user. The tasks store and retrieve data by calls on system service routines which interpret the requests with guidance from the system directories. The relevant results of the operation are presented to the user by the system output jobs.

### 3.1 USER'S VIEW

DM-1 is a console-directed system. It works in response to user commands issued in a job-request language from a console. The user treats the job library as a

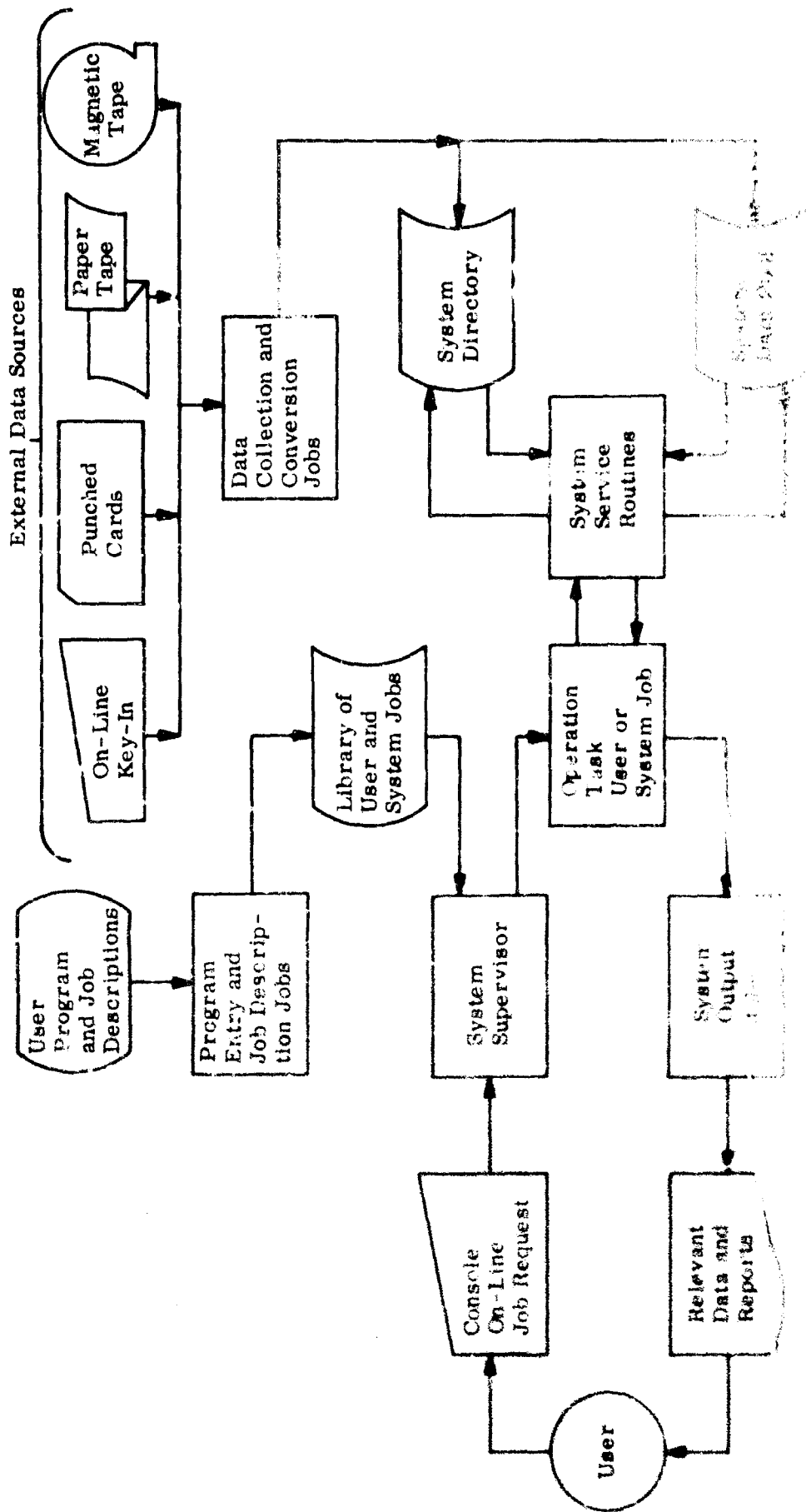


FIGURE 3-1 IBM System/360 Architecture

set of operators at his command. He treats the data pool as the operands to be bound to the operators. The jobs may be lengthy processes that consist of many tasks to be executed over large files of data or simple functions that consist of a single operation on a small unit of data. The user may execute a job by specifying its name and binding its input and output parameters. He may supply literal values for the parameters at the console, or he may give the names of the data pool items containing the values.

The jobs at the user's command fall into several categories. The input jobs accept external data and map it into prescribed items in the data pool. The processing jobs operate on items in the data pool to produce or update other items. The retrieval jobs select items on given criteria. The output jobs display items at the console or produce hard-copy reports. These are not exclusive categories. Many jobs are pre-defined sequences of other jobs that contain all of the categories.

### 3.1.1 Retrieval Operations

The user may engage in a dialogue with the system to determine the names and structural relationships of items in the data pool, to select items of interest, and to specify conditions for selecting the values of those items for display or processing. The dialogue guides the user to formulate an information retrieval statement and calls a retrieval job to create a new item for processing or to produce a report or display of selected items, based on the values of those items which meet a condition of arbitrary complexity. After a dialogue, the user may store the information retrieval statement with a name which permits him to call a retrieval job directly the next time the same need arises.

A retrieval job may be used to answer management questions about the state of any items in the data pool. A condition of arbitrary complexity may be used to select the items of interest. The condition and the structure of the data pool permit the user to focus as narrowly as possible on the significant data, thereby eliminating manual searches through bulky reports.

Another use for retrieval jobs is to develop a data set with special characteristics. An item might be developed by selecting the items which meet a condition and mapping them into the structure of the new item. The new item would then be available to other jobs for analysis or processing.



### 3.1.2 Interpretive Processing

An analyst at a console may apply a step-by-step procedure in processing data to meet a nonrecurring requirement. He begins with some initial data items and a set of operations, capable of answering his needs, in the job library. He applies a job to selected data items and its results are stored in another item. He may then use a system display job to see the results of the first step. Based on the results, he selects the next step and the data it is to operate on. This process is repeated until the desired results are achieved.

Each step of the process is initiated by a job-run request, which gives the name of the job and the values or names of the items which are to be used as the input and output parameters for the job. The outputs of earlier steps may be used as inputs to later steps. The process is feasible only if the jobs can turn around their results quickly. With a set of such operations in the library, the job-run request is analogous to a macroprogramming language which allows a user to code at a console and have his instructions interpreted and executed as he presents them.

### 3.1.3 Sequence Execution

When the user wishes to execute a sequence of jobs, the system will accept a full description of the sequence before executing any of the jobs. He specifies each job in the sequence and binds its input and output parameters, relating the outputs of one step to the inputs of later steps. The sequence will be executed as though it were a single job, with no interaction between the system and the user unless the jobs in the sequence interact with him.

### 3.1.4 Recurrent Jobs

A job may be created as a sequence of operations and stored in the job library. Some of the parameters of the job may remain unbound. In this way, useful sequences may be called and executed by a job-run request. Only the name assigned to the sequence and the binding for any unspecified parameters need be given to trigger execution of the sequence. A job created in this way may be used as a component in another job or called as a subroutine of a program, as may any other job in the library.

### 3.2 DATA ADMINISTRATOR'S VIEW

The management of a large complex data base for common use by a large number of consumers must be centralized. A data administrator is needed to maintain a balance among competing uses for the data. DM-1 provides a set of tools which support the data administrator in managing the data base. He analyzes the tradeoffs involved in competing usage and uses the system tools to maintain an appropriate balance. His primary concern is the selection of options which yield the best performance.

The data administrator manages the overall structure of the data base. Although he takes little interest in the internal structure of items which are essentially private files, he must resolve the structure conflicts that arise from common use of data. If an item's structure is oriented toward one application, other applications may require a structure transformation. If the information is maintained in several forms, any updating must be applied several times. Considerations like these influence the data administrator's decision about data structures.

The efficiency of operating on given data sets may be enhanced by several options of DM-1. The data administrator must evaluate the use of the options. Fields which are frequently used as keys in retrieving or selecting data may be indexed in one of several ways. The indexing creates an auxiliary directory table which permits rapid location of items based on values of the indexed field. Extra storage space is required, and the cost of modifying the indexed field is higher, but retrieval time is improved significantly. The data administrator must choose whether to index and which option to select. Similarly, a logical linkage may be established between separate structures in the data base. Here, too, there are advantages and disadvantages, and the data administrator is charged with the decision.

Another area of conflict relates to the integrity and security of data. The data administrator assigns all system users to a user's group, and assigns each group an access and modification level which locks them out of certain areas of the data base while permitting them to operate with other areas. Exceptions to the general rule are permitted. The data administrator may assign specific items of a forbidden class to specified users. The assignments may be made to depend on the values of a specified field.

DM-1 helps the data administrator by maintaining usage statistics on key fields and data retrievals. It supplies the tools which enable him to implement his decisions. A series of maintenance jobs permits him to define and change structures, index fields and eliminate the index, restructure large volumes of data, and enter data or modify it in a number of ways.

### 3.3 PROGRAMMER'S VIEW

DM-1 maintains a library of programs which are the basic building blocks of jobs. It controls the execution of programs and manages the sequence of the programs in a job. Guided by the program description and the data pool directories, the system relates the data pool items to the input requirements of the programs and maps the output data into other data pool items. The inputs of the individual programs are matched to data pool items and to the outputs of other programs, and any conditional selections or transformations are accomplished automatically according to the specifications of the job description or the job-run request. When a program is operating, it uses the Service Package to retrieve its input data and store its output data. These are some of the operational characteristics of the system.

#### 3.3.1 Program Description

A program cannot be made to be completely independent of the structure of the data on which it operates. During the processing steps of the program, the format of the individual units of data affected by the steps is implicit in the code of the program. However, this structure dependence exists only with respect to the units of data that share active memory with the program. The algorithms of the program are independent of the structure of data external to it, and they work properly if the data is delivered to the program's input buffer in the expected form. DM-1 manages the delivery of data to a program's input buffer and performs transformations on the data, if necessary, to put it into the form required by the program. To do this, the system must know the form of the data as it exists and the form required by the program. The former is given in the data pool directories. The form required by a program is given in the program description library.

A program written for operation in DM-1 is written to operate with formal input and output parameters. The structure of these items is oriented to the requirements of the operation performed by the program. These structures might correspond to actual structures in the data base, but this is not necessary. In fact, if a program is to be useful as a component of many jobs, it should be written to operate with the most convenient structure, and it should be independent of the precise format created by some other program. An item may be described as a formal parameter of the program, even though it is an exact replica of a data base item, to permit the program to continue to operate without change, even if the structure is changed in the data base.

A program is described to DM-1 by executing the program description job. The names and the precise structure of the formal input and output items are specified at the console. This information is translated by the program description job into an entry in the library. Once a program has been described to the system, it automatically becomes a job in the system's repertoire. It may be called and executed on specified data from a console, used as a component in another job, or called and executed as a subroutine of another program.

### 3.3.2 Data Storage and Retrieval Services

A program reads data from the data pool and writes data into it by calling on routines of the Service Package. The Service Package is analogous to an input-output control system. It contains a resident interpreter and a set of service routines. The routines are reentrant, so that they may serve more than one user at a time.

The program may retrieve an item by giving the formal name assigned by the programmer and supplying a buffer to receive the data. The system translates the formal name into the actual item assigned to it by the job description or the job-run request. An item may be written into the data pool in a similar way. The program places the data in a buffer and calls the appropriate service routine, giving the formal name of the item to be written.

When a program reads or writes parts of the same item repeatedly, it may initialize the system by opening the item for reading or writing. The translations to

internal item identifiers are accomplished when the item is opened. Later operations on the item are more direct. This service is especially valuable when reading or writing the records of a file.

The significant characteristics of the storage and retrieval services are the system's ability to transform item structures and the use of an invariant name in the program. The name remains the same in the program, no matter which items of the data pool are bound to it for a given run. It is completely independent of the location of the item or of the characteristics of the storage devices.

### 3.4 INTEGRATED VIEW

The characteristics of DM-1 may be highlighted and their relationships may be demonstrated by following the development of the two major system resources and describing the system in operation. The resources of the system are the data base and the job library.

#### 3.4.1 Data Base Development

The null data pool of the system contains a basic structure that provides logical space for items to be defined and the structure definition for a set of directory tables which describe themselves only.

In the development of a data base, items are defined to the system by using system maintenance jobs. Once the structure has been defined, data may be mapped into the data base by other maintenance jobs. These processes continue until an initial data base is developed. Since the data base is a constantly varying resource, the structures and data are repeatedly changed throughout the life of the system by system maintenance jobs and application programs. Each time a change is made in the structure of the data base, the system directories are updated. As the development progresses, the directories become richer in content.

The data administrator makes some initial decisions about indexing, linking, and the access and modification levels of various users with respect to various items. These decisions are continually modified during the life of the system to adjust to experience and to take account of new developments.

### 3.4.2 Job Library Development

The DM-1 job library contains a number of general-purpose system jobs for building, maintaining, and querying the data base and job library. This basic repertoire of operations can be expanded by adding user jobs to the library. User jobs may be composed by combining application-oriented programs and the existing system jobs to achieve the desired unit of work.

New programs are added to the library through the Program Entry job. After the program has been compiled, it is described to DM-1. The user specifies the program name, the identifier for loading the object code, and the names and structures of the program's inputs and outputs. This operation places the program in the DM-1 job library. The program can then be used as a component of a more complex job.

Additional jobs may be created through the Job Description job by combining the existing jobs in the library. The language of a job description is like a macroprogramming language. Each component job is one step in the new job. The job's description consists of a job image and a series of job requests for the steps of the job. The job image gives the name of the new job and the names of its input and output parameters, if any. The input and output parameters of the new job are those parameters of the component jobs which will not be specified until a job-run request is issued. The job requests for the steps of the new job contain the names of the component jobs and the names of their input and output parameters. The inputs are related to job inputs, the outputs of previous components, literal values, and data base items. The outputs are related to job outputs, the inputs of succeeding components, and data base items.

### 3.4.3 Operational Use

DM-1 operates in association with an operating system which controls a multi-programmed environment. The operating system recognizes DM-1 jobs as a class. When a DM-1 job is requested, the operating system ensures that the Service Package is in memory as part of the operating system's input-output control package. One set of Service Package routines can serve any number of jobs in a time-shared mode.

Figure 3-2 is a block diagram that depicts the relationships among the system components in operational use. A DM-1 job is requested by keying the job-request

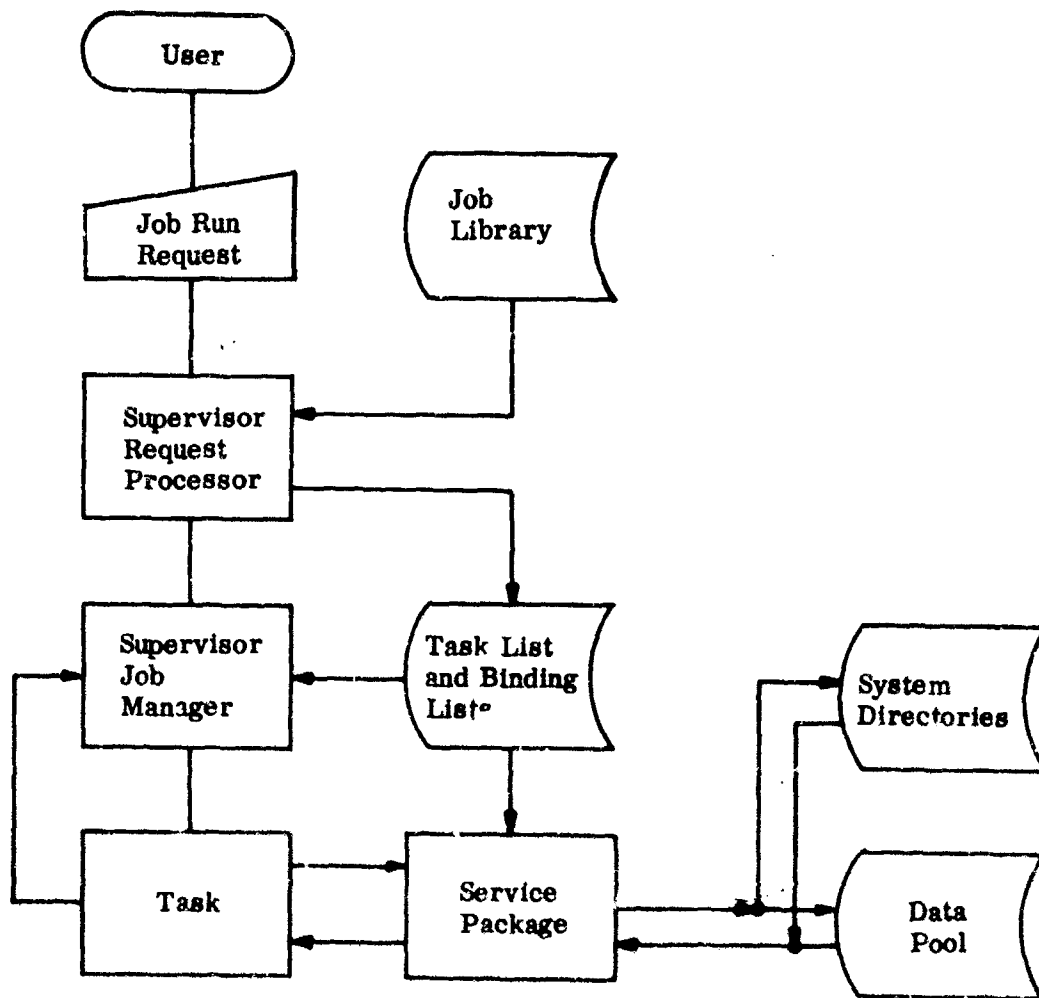


Figure 3-2. System Operation, Block Diagram

message at the console and signaling the operating system that a DM-1 job is to be executed. The operating system assigns memory to the job according to its own scheduling algorithm.

The first step in a DM-1 job is accomplished by the Request Processor. This is a part of the DM-1 Supervisor which interprets the job request, prepares the job parameters, and initiates the job. The Request Processor uses the job name to retrieve the job description from the library. It develops a task list containing the identifiers for the sequence of component tasks (jobs or programs). Using the job images from the job description and the input and output specifications from the job request, it builds the input-output binding lists to relate the parameters of each task to data pool items. Any item transformation or conditional selections required in relating actual items to formal parameters are scheduled by the Request Processor.

The Job Manager is the part of the DM-1 Supervisor that controls the execution of the tasks of the job. It reads the next task from the task list, loads it, and gives it control. The task maintains control until completion, unless the operating system's scheduling algorithm interrupts it. When the task is completed, control returns to the Job Manager to load the next task. The last task in the task list is the Request Termination task which performs final housekeeping and terminates the job.

While a task is operating, it uses the Service Package to retrieve and store its data. The Service Package translates the formal parameter names used by the task to data pool items by using the binding lists. Data retrieval and storage are accomplished by the Service Package by using the system directories to locate and interpret the data pool items.



## SECTION IV. DATA POOL FEATURES

For efficient use of a common data base, the data must be highly organized to permit rapid access and simple specification of pertinent items. The fundamental strategy of DM-1 is to retain data in as flexible and accessible a form as possible by using system directories. The data is treated as a segmented string whose structure and syntax are maintained in separate directory tables, which are themselves part of the segmented data string.

### 4.1 DATA STRUCTURES

The use of data items related in a hierarchy is a familiar artifice of data processing. The grouping of unit records behind a header card is an example of the use of a hierarchy. The entire deck corresponds to a node at the highest level in a tree structure. It is a file subsuming a record for each header group at the next level. Each record subsumes the fields of the header card and a file of trailer cards. The header fields are terminal items in the tree structure. Each trailer file subsumes a record for each card which, in turn, subsumes the fields of the trailer card.

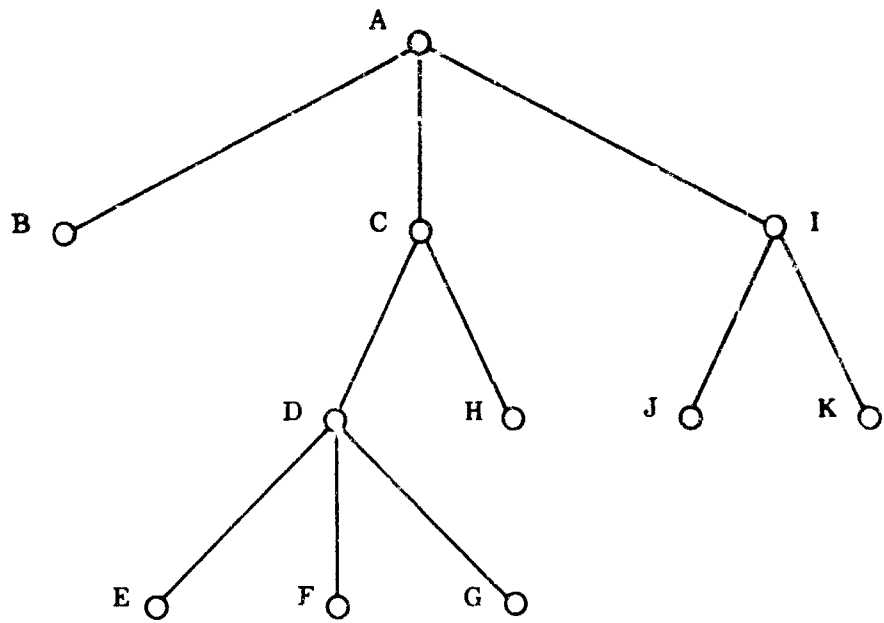
DM-1 uses several methods of representation for a tree structure. Figure 4-1 contains four different ways of showing the same hierarchical relationships among eleven items. The first form (a) represents each item as a node.

Node A contains the three subnodes B, C and I. Node C contains subnodes D and H. Node D contains subnodes E, F, and G, and node I contains subnodes J and K. The indented form (b) shows the same relationship among the items by indenting the subitems with respect to their parent items. The subitem form (c) assigns to each item a size that stands for the number of subitems contained in that item. The parenthetic form (d) uses parentheses to delimit the level of the items and slashes to separate items on the same level in the hierarchy. A is alone on the highest level and is composed of B, C, and I on the next level, where C is composed of D and H, etc.

#### 4.1.1 Item Types

An item is a single node in a tree structure. The word item is used as a generic term to include a node and all of its substructure. In this sense, the entire data pool is a single item. It is a branching structure with a unique node at the top level and branches emanating from this node to the next lower levels of nodes, and so on, until the terminal nodes are reached. In the data pool, each node corresponds to a named data item and the subtree subsumed by any node represents the definition of that item. The items recognized by DM-1 fall into several classes:

- (1) Field. A field is a terminal item; that is, it contains no substructure. A field is defined by its name, type, size, and units. A name is an alphanumeric string which is used externally to denote an item. The type specifies the coding scheme of the field. It may be alphanumeric, integer, binary, octal, decimal, or exponential. The size is the number of characters in the field (alphanumeric characters, binary bits, decimal digits, etc.). The size may be fixed or variable in length. The unit designator specifies the scale on which the value of the field is measured; e.g., volts, amperes, meters, etc.
- (2) Statement. A statement is an item which subsumes other items. Its subitems may be fields, files, or other statements. For example, a statement may contain several fields. Another statement may contain that statement and several other items. A statement is defined by its name and the definitions for its subitems. In effect, the statement is a mechanism for associating several related



(a) Node Representation

|           |       |
|-----------|-------|
| ARLINGTON | A (3) |
| BOSTON    | B (0) |
| CAMBRIDGE | C (2) |
| DENVER    | D (3) |
| ELKTON    | E (0) |
| FARGO     | F (0) |
| GENEVA    | G (0) |
| HANOVER   | H (0) |
| ISTANBUL  | I (2) |
| JAKARTA   | J (0) |
| KASHMIR   | K (0) |

(b) Indented Representation

(c) Subitem Representation

A ( B / C ( D ( E / F / G ) / H ) / I ( J / K ) )

(d) Parenthetic Representation

Figure 4-1. Tree-Structure Representations

items to show the relationship and permit them to be treated as a unit. The data value of a statement is the set of values of its subsumed fields.

- (3) File. A file is an item which subsumes an arbitrary number of subitems, each of which has an identical structure. Its subitems are records. The file is defined by naming it and defining each of the subitems of its records. The data value of a file is the group of values of the fields subsumed by all the records of the file.
- (4) Record. A record is the subitem of a file. It is exactly like a statement in that it subsumes other items. Like the statement, its subitems may be fields, statements, files, or special items in any combination. Unlike a statement, however, a record may occur an arbitrary number of times in the data. The file and record are related in such a way that one cannot be separated from the other. The record always follows the file. It is defined when the file is defined. The data value of a record is the set of values of the fields it subsumes.
- (5) Null Node. A null node is a terminal item representing a position in the logical structure. It contains no subitems and it stands for no data. It is an artifice to reserve a slot in the logical structure.
- (6) Link. A link is an item which logically subsumes a set of items on another stem in the tree structure. It permits a logical connection between two branches of the tree and gives the structure a network character.

#### 4.1.2 Item Structure

The DM-1 data description language permits the use of variable length fields, optional items, and nested structures. Files with variable numbers of records may be embedded within the records of higher level files. Any number of files, statements or fields may be subsumed by the same record or statement.

Table 4-1 shows the definition of the item PURCHASING in the indented outline form. The item is a statement containing three files. The ITEM file is a catalogue containing a record for each item which might be ordered on a purchase order. Each record in the catalogue contains three required fields. The DESCRIPTION field is marked as optional by the asterisk. The ORDER file is a list of outstanding purchase

orders. Each record contains identifying information for a purchase order and an ITEM LIST. The ITEM LIST contains a record for each item on the purchase order. This is an example of a file embedded in each record of a higher level file. The VENDOR file contains records describing vendors with a list of the active purchase orders for each vendor.

The column headed ICC in Table 4-1 contains an internal system identifier for each item in the definition. This logical name for the item is called an Item Class Code (ICC). It is derived from the relative position of the item within the data pool. In the figure, the item PURCHASING has the ICC 1, since it is the parent node of the entire structure. Its subitems are numbered on the next level: 1.1 for the ITEM file, 1.2 for the ORDER file and 1.3 for the VENDOR file. The records of the files occupy a level in the structure. The level is represented by an R in the ICC. The subitems of the records are numbered on the next level.

Another code, the Item Position Code (IPC), is used internally to identify units of data in the data pool. The ICC becomes an IPC when a record number replaces each R in the ICC. For example, the IPC 1.1.3.1 stands for the unique occurrence of the ITEM NO. field (1.1.R.1) in the third record of the ITEM file. The IPC 1.2.5.6 represents the ITEM LIST file (1.2.R.6) for the purchase order identified in the fifth record of the ORDER file.

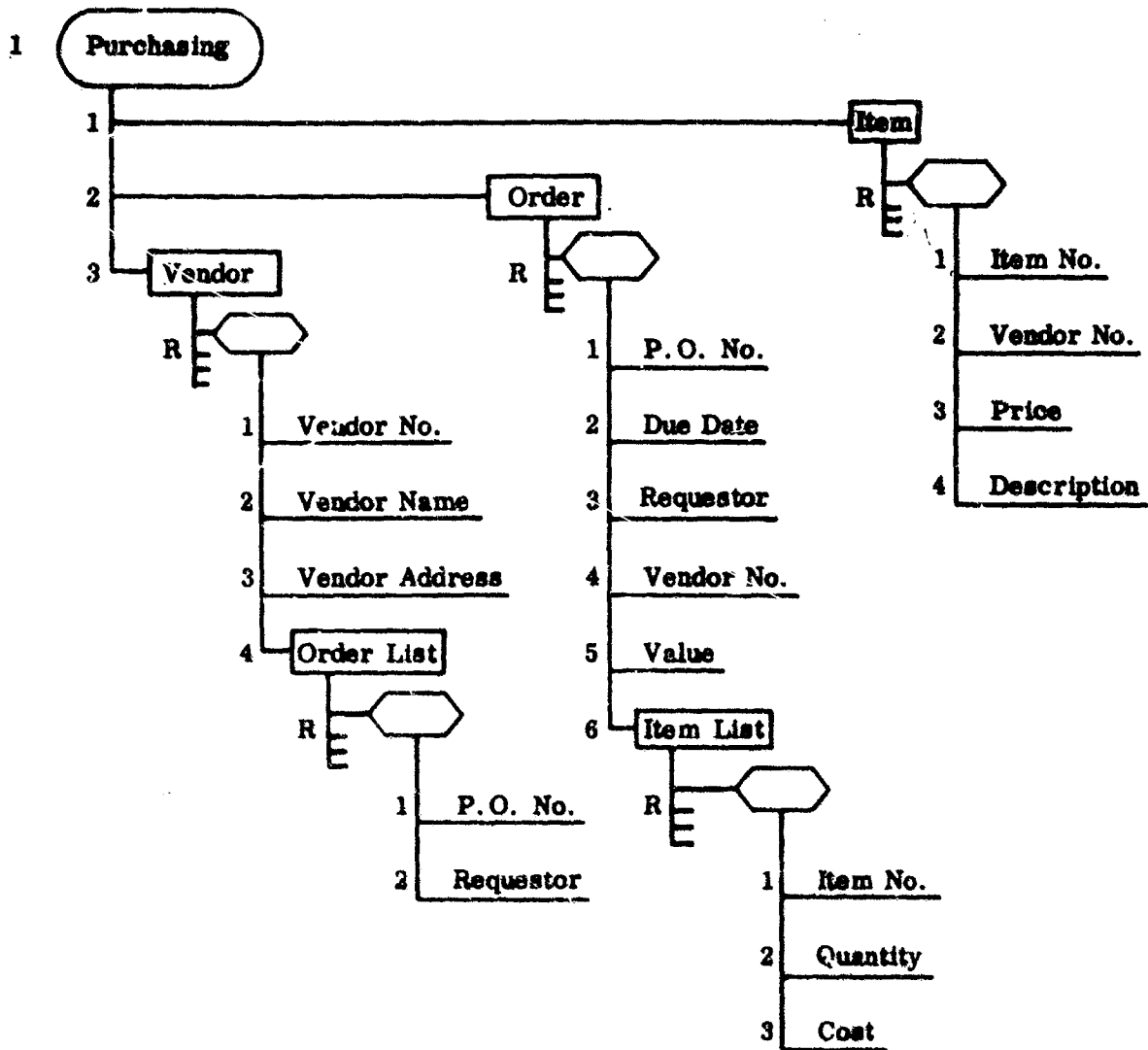
The structure diagram shown in Figure 4-2 is another form of item definition. The structure of the PURCHASING statement is shown in a form analogous to the node form of structure representation. The shape of the node specifies the type of the item at that node: an oval for a statement, a rectangle for a file, a hexagon for a record, and a line for a field. Figure 4-2 clarifies the meaning of the Item Class Code as the logical position of the item in the structure. ICC's are assigned to items by counting the items on each level and moving to a deeper level each time a statement or file is encountered.

#### 4.2 PRIMARY DIRECTORIES

The structural description of the data is maintained by D.M-1 in the system directories. The primary directories contain the information from the item definition. The primary directories are the Item List, the Term List, and the Term Encoding Table. These are used to focus in on the data and to describe its structure. They function as a guide in interpreting the data so that it may be delivered to a program in a suitable form for processing.

TABLE 4-1. DEFINITION FOR THE PURCHASING ITEM

| ICC         | ITEM DEFINITION      |
|-------------|----------------------|
| 1           | PURCHASING, S        |
| 1.1         | ITEM, F              |
| 1.1.R.1     | ITEM NO., I, V       |
| 1.1.R.2     | VENDOR NO., I, 4     |
| 1.1.R.3     | PRICE, E, V          |
| 1.1.R.4     | *DESCRIPTION, A, V   |
| 1.2         | ORDER, F             |
| 1.2.R.1     | P. O. NO., I, 6      |
| 1.2.R.2     | DUE DATE, D, 6       |
| 1.2.R.3     | REQUESTOR, A, V      |
| 1.2.R.4     | VENDOR NO., I, 5     |
| 1.2.R.5     | VALUE, E, V          |
| 1.2.R.6     | ITEM LIST, F         |
| 1.2.R.6.R.1 | ITEM NO., I, V       |
| 1.2.R.6.R.2 | QUANTITY, I, 5       |
| 1.2.R.6.R.3 | COST, E, V           |
| 1.3         | VENDOR, F            |
| 1.3.R.1     | VENDOR NO., I, 4     |
| 1.3.R.2     | VENDOR NAME, A, V    |
| 1.3.R.3     | VENDOR ADDRESS, A, V |
| 1.3.R.4     | ORDER LIST, F        |
| 1.3.R.4.R.1 | P. O. NO., I, 6      |
| 1.3.R.4.R.2 | REQUESTOR, A, V      |



Note: The letter "R" indicates that the item is a record which is repeated in the data as many times as the item is recorded.

Figure 4-2. Purchasing Item Structure

#### 4.2.1 Item List

The Item List is at the center of the directory system. It is a file with a record for each item (node) in the data base structure. The records are in order by the Item Class Code of the item. Each Item List entry contains the item type and the size of the item. The size of records are determined by the number of subitems they subsume directly.

The Item List also contains other information about the item. However, the primary structural information is the item type and size. Other parts of the Item List entry will be discussed in other sections of this report. The detailed structure of the Item List is presented in Volume II, Paragraph 2.3.

Table 4-2 shows the Item List for the PURCHASING item in the third column. The item type and the size are shown in each entry. The optional DESCRIPTION field is flagged with an asterisk. The structure of an item is implied by the sizes given for nonterminal items.

#### 4.2.2 Term List

The item names and units are maintained in a Term List file which is parallel to the Item List file. These elements are maintained separately because they are not needed to interpret the data structure and it is desirable to store the structural information as compactly as possible in the Item List. For each record in the Item List there is a record in the Term List, and the corresponding record numbers contain information about the same item.

#### 4.2.3 Term Encoding Table

The item name is used as the identifier of the item by DM-1 users. The ICC is used as the identifier by the system. To enable the system to translate from an item's name to its ICC, the Term Encoding Table (TET) is maintained. The TET is a table containing a record for each unique item name in alphabetical order. The ordering permits rapid translation from an item's name to its ICC. Since there may be more than one item with a given name, each record of the TET contains a file of ICC's corresponding to a single name. Table 4-3 shows the Term Encoding Table for the purchasing statement.



TABLE 4-2. TERM LIST AND ITEM LIST

| <u>TERM LIST</u> | <u>ICC</u>  | <u>ITEM LIST</u> |
|------------------|-------------|------------------|
| PURCHASING       | 1           | S, 3             |
| ITEM             | 1.1         | F, V             |
| -----            | 1.1.R       | R, 4             |
| ITEM NO.         | 1.1.R.1     | I, V             |
| VENDOR NO.       | 1.1.R.2     | I, 4             |
| PRICE            | 1.1.R.3     | E, 6             |
| DESCRIPTION      | 1.1.R.4     | A, V, *          |
| ORDER            | 1.2         | F, V             |
| -----            | 1.2.R       | R, 6             |
| P. O. NO.        | 1.2.R.1     | I, 6             |
| DUE DATE         | 1.2.R.2     | D, 6             |
| REQUESTOR        | 1.2.R.3     | A, V             |
| VENDOR NO.       | 1.2.R.4     | I, 5             |
| VALUE            | 1.2.R.5     | E, V             |
| ITEM LIST        | 1.2.R.6     | F, V             |
| -----            | 1.2.R.6.R   | R, 3             |
| ITEM NO.         | 1.2.R.6.R.1 | I, V             |
| QUANTITY         | 1.2.R.6.R.2 | I, 5             |
| COST             | 1.2.R.6.R.3 | E, 7             |
| VENDOR           | 1.3         | F, V             |
| -----            | 1.3.R       | R, 4             |
| VENDOR NO.       | 1.3.R.1     | I, 4             |
| VENDOR NAME      | 1.3.R.2     | A, V             |
| VENDOR ADDRESS   | 1.3.R.3     | A, V             |
| ORDER LIST       | 1.3.R.4     | F, V             |
| -----            | 1.3.R.4.R   | R, 2             |
| P. O. NO.        | 1.3.R.4.R.1 | I, 6             |
| REQUESTOR        | 1.3.R.4.R.2 | A, V             |

TABLE 4-3. TERM ENCODING TABLE

| <u>NAME</u>    | <u>ICC FILE</u>               |
|----------------|-------------------------------|
| COST           | 1.2.R.6.R.3                   |
| DESCRIPTION    | 1.1.R.4                       |
| DUE DATE       | 1.2.R.2                       |
| ITEM           | 1.1                           |
| ITEM LIST      | 1.2.R.6                       |
| ITEM NO.       | 1.1.R.1<br>1.2.R.6.R.1        |
| ORDER          | 1.2                           |
| ORDER LIST     | 1.3.R.4                       |
| P. O. NO.      | 1.2.R.1<br>1.3.R.4.R.1        |
| PRICE          | 1.1.R.3                       |
| FURCHASING     | 1                             |
| QUANTITY       | 1.2.R.6.R.2                   |
| REQUESTOR      | 1.2.R.3<br>1.3.R.4.R.2        |
| VALUE          | 1.2.R.5                       |
| VENDOR         | 1.3                           |
| VENDOR ADDRESS | 1.3.R.3                       |
| VENDOR NAME    | 1.3.R.2                       |
| VENDOR NO.     | 1.1.R.2<br>1.2.R.4<br>1.3.R.1 |

In practice, the names of other items in the data pool would be merged with the names of the items in the purchasing statement in a single TET.

### 4.3 DATA REPRESENTATION

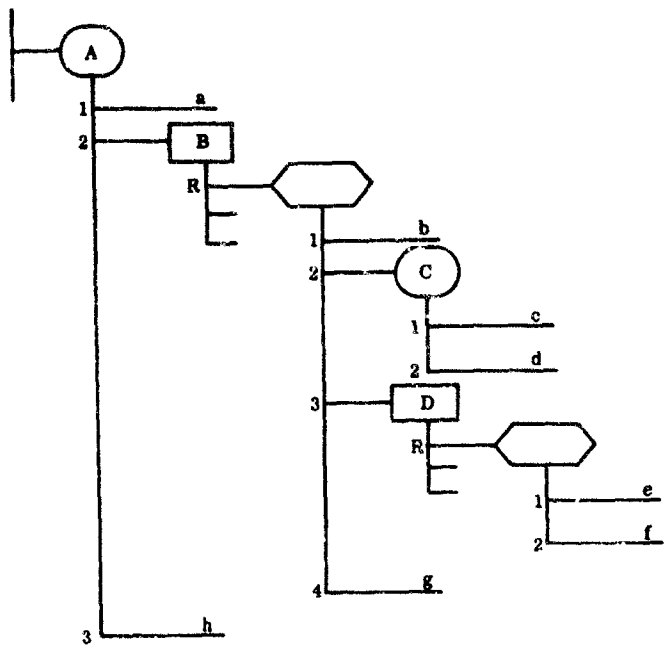
DM-1 treats the entire data pool as an unformatted stream of binary bits. This stream is segmented arbitrarily at any item boundary. Since no account is taken of word boundaries or the coding mechanisms of the devices, the data pool segments are independent of the characteristics of the computer and the storage devices. The service routines that interpret the data stream with the aid of the system directories are computer dependent. This approach focuses the computer dependence of the system in a small set of routines.

#### 4.3.1 Data Stream

A hypothetical example will be used to explain the system's mechanism for representing data. Figure 4-3 (a) contains a structure diagram of a statement named A. It consists of the field a, the file B and the field h. The file B contains the field b, the statement C, the file D and the field g. The statement C contains two fields and each record of D contains two fields. The subitem representation of the structure is given next to the structure diagram. This is the form in which structure information appears in the Item List.

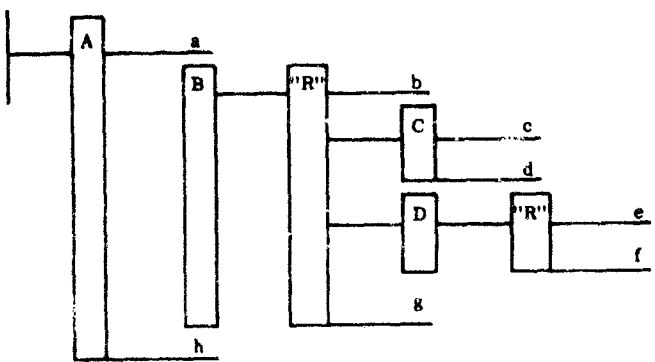
The files, records and statements in a structure relate their subnodes as a single entity, but only the fields take on values in the data. The diagram in Figure 4-3 (b) emphasizes this character of a structure. It shows the string of fields, a through h, emanating from vertical bars and representing the nonterminal items. The diagram in Figure 4-3 (c) is derived from the buss network. It is a maze which defines the logical order of items in the structure. The path through the maze begins at the top left. The entrance is at item A. Since this is a nonterminal item, the maze must be followed to the right until a field is encountered. At the field a, the maze contains a gap which opens to the next subitem of the statement A. The file B is a nonterminal item and its subitems, the records of the file, are also nonterminal. The path moves to the right until the field b occurs with an opening to the next level. This pattern continues through the maze, passing the items of the structure in their data base order:

A, a, B, "R", b, C, c, d, D, "R", e, f, g, and h.



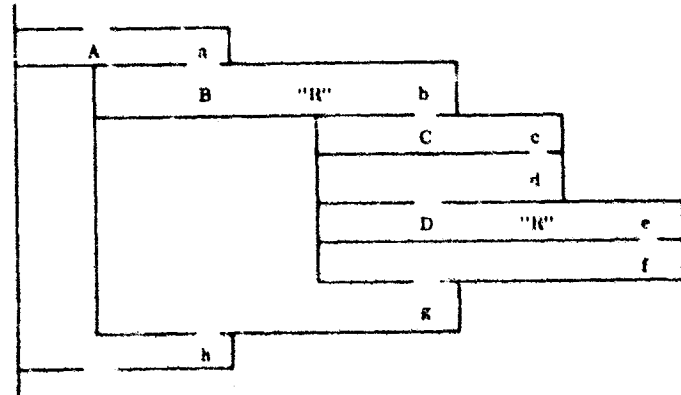
- A (3)
- a (Ø)
- B (R)
- "R"(4)
- b (Ø)
- C (2)
- c (Ø)
- d (Ø)
- D (R)
- "R"(2)
- e (Ø)
- f (Ø)
- g (Ø)
- h (Ø)

(a) Structure Diagram



- a
- b
- c
- d
- e
- f
- g
- h

(b) Bus Network Diagram



- a
- b
- c
- d
- e
- f
- g
- h

(c) Structure Maze

Figure 4-3. Sample Structure

The Item List is like a template in the form of the maze and is to be used to interpret the unstructured string of data values. Only field values occur in the data stream. An example of a data maze is given in Figure 4-4. The data stream, represented in the figure by the column of subscripted letters at the right, consists of values for the fields. The interpretation of this data stream, with the Item List as a template, is symbolized by the path through the data maze. The example assumes that there are three records in file B, with three records of file D in the first record of file B, four in the second, and two in the third.

#### 4.3.2 Segmentation

The fields in the data stream must follow each other in the strict logical sequence dictated by the data pool structure. This does not mean that the data must be stored in a strict, physical sequence. DM-1 segments the data stream and incorporates the ability to store the segments anywhere on the available devices. The logic of the system permits the segments to be of arbitrary size, but a size of 9216\* bits has been selected for convenience.

In writing data, the fields of the data stream are composed in memory blocks of segment size under the direction of the Item List. When a segment is full, it may be stored in any available location of any storage device. The segment is identified by the Item Position Code (IPC) of the first item it contains. This IPC is used as the key whenever the segment is retrieved.

Each segment is assigned a segment name which is used by the operating system to retrieve the segment. DM-1 maintains a Segment Name List (SNL) so that it may translate an IPC into the segment name of the segment containing the data identified by the IPC. The Segment Name List is a file whose records contain the IPC of the first item in a segment and the segment's name. It is segmented like any other data, and the existence of SNL entries for SNL segments permits the system to focus rapidly on the desired segment through a multilevel, variable depth, indirect addressing mechanism.

---

\* The number 9216 is divisible by 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 36, 48, etc. It is a convenient number for devices with many word lengths.

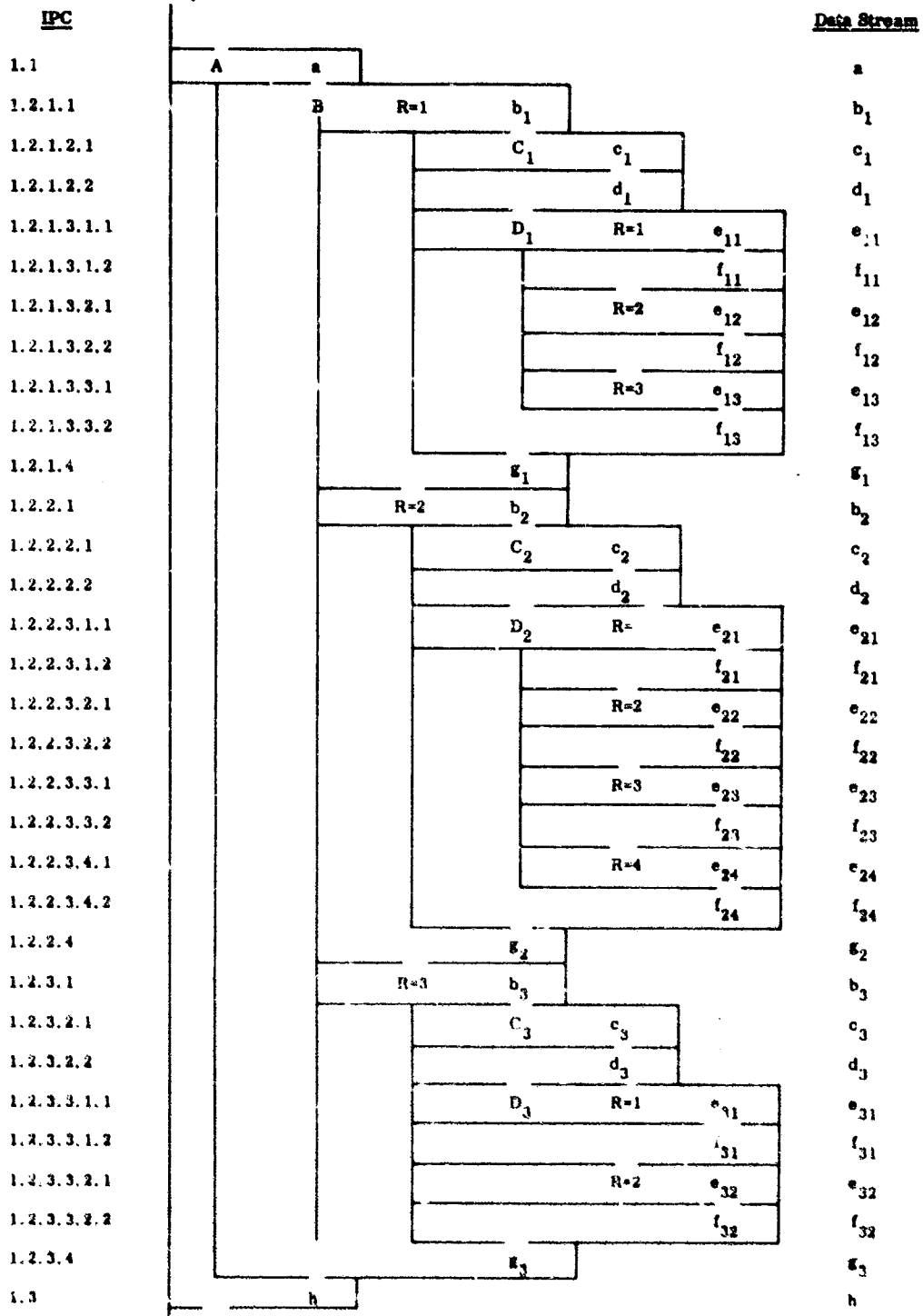


Figure 4-4. Data Maze

### 4.3.3 Segment Index

The Item List does not contain enough information to allow DM-1 to interpret the data stream. It gives the fixed size (number of subnodes) of records and statement and the size for each fixed length field. However, there are two levels of variability which must be taken into account. The sizes of variable length fields may differ from one value to the next. The Item List contains only one entry for a field which may take many values. Similarly, the size of a file, i.e., the number of records it contains, varies from one occurrence of the file to the next. A file embedded in a higher file occurs in each record of the higher file, but it has only one entry in the Item List.

The segment index, a string of bytes in the data segment, gives the size for variable length fields and the number of records for files in the order of the occurrence of the variable items in the segment. The segment index for the data stream of Figure 4-4 might be:

|          |                       |                       |                       |                       |                       |                       |                       |                       |                       |
|----------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 3        | 18                    | 3                     | 7                     | 5                     | 4                     | 20                    | 12                    | 2                     | 8                     |
| <u>B</u> | <u>c</u> <sub>1</sub> | <u>D</u> <sub>1</sub> | <u>g</u> <sub>1</sub> | <u>c</u> <sub>2</sub> | <u>D</u> <sub>2</sub> | <u>g</u> <sub>2</sub> | <u>c</u> <sub>3</sub> | <u>D</u> <sub>3</sub> | <u>g</u> <sub>3</sub> |

if the fields c and g are variable length. Since the first variable item encountered in the stream is the file B, its size is given first in the segment index. The file has three records. The next variable length item in the stream is the field c. Its value in the first record of file B is 18 units (bits or characters) long. The field g has a value whose length is seven units. The remaining numbers of the segment index give the sizes of the field C, the file D and the field g for their occurrences in records 2 and 3 of the file B.

### 4.4 SAMPLE RETRIEVAL

An example will be used to demonstrate the use of the system directories in the retrieval of an item of data. The example is somewhat artificial because it is taken out of context, and it must be simplified to highlight the relationships without burdening the reader with undue detail.

The PURCHASING item, introduced in Table 4-1, Figure 4-2, and Tables 4-2 and 4-3 will be used as a basis for the example. The directories for the item are shown in Figure 4-5.

| TERM ENCODING TABLE |        | ICC    | ITEM LIST |      | SEGMENT NAME LIST |              |
|---------------------|--------|--------|-----------|------|-------------------|--------------|
| Name                | ICC    |        | Type      | Size | Identifier        | Segment Name |
| Cost                | 12R6R3 | 1      | S         | 3    | (I) 1             | 58469        |
| Description         | 12R4   | 11     | F         | V    | (I) 1.2.R.2       | 87466        |
| Due Date            | 12R2   | 11R    | R         | 4    | → (I) 1.3         | 42879        |
| Item                | 11     | 11R1   | I         | V    | (N) Cost          | 74346        |
| Item List           | 12R6   | 11R2   | I         | 4    | → (N) Order       | 49632        |
| Item No.            | 11R1   | 11R3   | E         | 6    | (N) Quantity      | 89248        |
|                     | 12R6R1 | 11R4   | A         | V    | (S) 1             | 73296        |
| Order               | 12     | 12     | F         | V    | (S) 1.1.68.4      | 25321        |
| Order List          | 13R4   | 12R    | R         | 6    | (S) 1.1.132.2     | 64843        |
| → P.O. No.          | 12R1   | 12R1   | I         | 6    | (S) 1.2.5.3       | 46932        |
|                     | 13R4R1 | 12R2   | D         | 6    | (S) 1.2.23.5      | 65167        |
| Price               | 11R3   | 12R3   | A         | V    | .                 | .            |
| Purchasing          | 1      | 12R4   | I         | 5    | .                 | .            |
| Quantity            | 12R6R2 | 12R5   | E         | V    | .                 | .            |
| → Requestor         | 12R3   | 12R6   | F         | V    | (S) 1.3.10.2      | 87933        |
|                     | 13R4R2 | 12R6R  | R         | 3    | → (S) 1.3.48.3    | 34658        |
| Value               | 12R5   | 12R6R1 | I         | V    | (S) 1.3.64.1      | 24863        |
| Vendor              | 13     | 12R6R2 | I         | 5    | (S) 1.3.82.1      | 52178        |
| Vendor Addr.        | 13R3   | 12R6R3 | F         | 7    |                   |              |
| Vendor Name         | 13R2   | 13     | → F       | V    |                   |              |
| → Vendor No.        | 11R2   | 13R    | R         | 4    |                   |              |
|                     | 12R4   | 13R1   | I         | 4    |                   |              |
|                     | 12R1   | 13R2   | A         | V    |                   |              |
|                     |        | 13R3   | → A       | V    |                   |              |
|                     |        | 13R4   | F         | V    |                   |              |
|                     |        | 13R4R  | R         | 2    |                   |              |
|                     |        | 13R4R1 | → I       | 6    |                   |              |
|                     |        | 13R4R2 | A         | V    |                   |              |

Figure 4-5. Directories for Line Purchasing Item



Suppose that a user wants to retrieve the purchase order numbers for purchase orders issued by J. Jones against the vendor whose number is 3204. The request might look like:

```
RETRIEVE:      P. O. NO
               IF VENDOR NO. = 3204 AND
               REQUESTOR = J. JONES
```

This request would be handled by the query job; however, the steps explained in this section are common to many retrieval situations. The details are greatly simplified and the condition is selected so that an orderly retrieval results. The general conditional search capability of the system is more comprehensive than this example implies.

#### 4.4.1 Name Translation

The item names in the request must be translated to the system identifiers, the ICC's. This is done through the Term Encoding Table (TET). The TET is a data file like any other data in the system and it is segmented. The names are translated to ICC's by retrieving the appropriate segment of the TET and searching its entries until a match is found on the names. This is handled by a system service routine.

The routine first uses the Segment Name List (SNL) to discover the segment of the TET to retrieve. The SNL entries for the TET are prefixed with a special identifier. The SNL of Figure 4-5 shows some entries prefixed with (N). These are TET entries. To translate the name P. O. NO., the routine takes these steps:

- (1) Match the name against the identifiers in the SNL until an identifier less than or equal to P. O. NO. is found, where the next entry has an identifier greater than P. O. NO. Since the entry sought falls between the identifiers, it is in the segment identified by the first of the two identifiers. In the SNL shown in Figure 4-5, the name P. O. NO. is found to fall between ORDER and QUANTITY. The TET segment containing the entry for P. O. NO. is, therefore, the one which begins with the entry for the name ORDER; the segment's name is 49632.
- (2) Retrieve the segment of the TET containing the desired entry, and search the TET entries until a match is found. In the TET in Figure 4-5, the entry for P. O. NO.

is found in the second segment. There are two items with that name: 12R1 and 13R4R1. Only one of these is needed. The choice must be made by the use of qualifiers or by the context of the problem. For example, the qualifier **VENDOR** could have been used in the problem statement to indicate that all pertinent items are subsumed by the vendor file. This would dictate the selection of the ICC 13R4R1 for P. O. NO. , since **VENDOR** has the ICC 13 and is a parent of the pertinent item.

- (3) Follow similar steps for the names **REQUESTOR** and **VENDOR NO.** The consistent set of ICC's discovered for the three names is:

|            |   |        |
|------------|---|--------|
| P. O. NO.  | = | 13R4R1 |
| VENDOR NO. | = | 13R1   |
| REQUESTOR  | = | 13R4R2 |

#### 4.4.2 Search Strategy

The structural relationships among the items in the sample retrieval request dictate the strategy of searching for the pertinent purchase order numbers. The ICC of the P. O. NO. field, 13R4R1, contains two record numbers. The condition on **VENDOR NO.** and **REQUESTOR** is used to set these record numbers to the values which meet the conditions. The search strategy used in an actual query job is more comprehensive than the one to be discussed here.

The best strategy is to establish the record numbers which meet the conditions at the higher level first. This narrows the number of files which must be searched at the lower level. The condition on **VENDOR NO.** is the key to establishing the record numbers at the higher level. Only those records which contain the value 3204 for **VENDOR NO.** need be considered.

There are several ways of determining which records contain the key value. If the field is indexed, the system maintains a subsidiary directory table that relates each value the field assumes to the set of record numbers containing the value. Indexing will be discussed in Paragraph 4.5. If the field is not indexed, the file with record numbers are to be established must be searched to determine the records which contain the key value. For the purposes of this example, assume that record number 51 is found to contain the value 3204 for the field **VENDOR NO.**

The second record number need be established only within record number 51 at the higher level. If the higher file contains 100 records, there are 100 files at the lower level. Establishing the record number at the higher level first eliminates 99 files from consideration. Effectively, the ICC of the pertinent purchase order numbers is translated from 1.3.R.4.R.1 to 1.3.51.4.R.1, with only the record number of one file to be established by further operations. The condition on the field REQUESTOR establishes the record numbers in the lower level file. Again, the subsidiary directory is used if the field is indexed, or the file is searched if it is not. In this example, assume that record number 12 is discovered to contain the key value J. JONES for REQUESTOR.

On each level, more than one record number might meet the condition. In general a multidimensional array of record numbers is developed from a condition. The array provides the appropriate record numbers for any set of desired items to be retrieved under the condition. The retrieval steps discussed in the following paragraphs are performed for each set of related items and for each of the record number groups which meet the condition.

The retrieval steps will be discussed for the retrieval of the purchase order number in the 12<sup>th</sup> record of the order list file which is in the 51<sup>st</sup> record of the VENDOR file. The condition established these record numbers which convert the logical identifier of the desired field, P. O. NO., from the ICC 1.3.R.4.R.1 to the IPC 1.3.51.4.12.1.

#### 4.4.3 Data Segment Retrieval

When the IPC of the pertinent item is known, the data segment containing that item can be retrieved through the SNL. The steps discussed are used for random retrieval of any item in the data pool or for initializing an item for serial processing or random processing within the bounds of the item.

The first step is to obtain the segment name and the range of its data contents from the SNL. The desired item has the IPC 1.3.51.4.12.1. As shown in Figure 4-5, this IPC falls between the segment identifiers 1.3.48.3 and 1.3.64.1. The desired item is within the segment named 34658 with other data ranging between the bounding identifiers.

The next step is to prepare an Item List Table to act as the structure template of the part of the data stream contained in the segment. Since both boundaries of the segment are within the file whose ICC is 1.3 (the VENDOR file), the definition for that file is sufficient to interpret the entire data stream within the data segment. The segment of the Item List which contains the definition for the VENDOR file is retrieved through the SNL. The segment needed is 42879 as shown in the SNL of Figure 4-5. This Item List segment is retrieved, and the definition for the VENDOR file is mapped into the Item List Table.

The data segment (34658) is retrieved and the Item List Table is initialized so that the system can step through the data stream segment to the desired item. This is accomplished by stepping down the Item List Table to the entry corresponding to the first item in the data segment (1.3.R.3 corresponding to 1.3.48.3), thereby setting parameters which direct the system in further stepping.

#### 4.4.4 Data Stream Interpretation

At this point in the example, a segment of the data stream is available with the part of the Item List needed to interpret it. The system steps from the item 1.3.48.3 to the item 1.3.51.4.12..., which is the desired purchase order number. The stepping is accomplished by summing the sizes of each item preceding the desired item to develop a pointer to the precise location of the desired item within the segment. The sizes of fixed length fields, records, and statements are obtained from the Item List Table. The sizes of variable length fields and files are obtained from the segment index. Sizes are accumulated and the IPC of each item is developed until the IPC of the desired item is reached. The value for this item may be extracted for display or processing.

#### 4.5 INDEXING

An optional feature offered by DM-1 is the ability to index selected fields. When a field is indexed, the system maintains a subsidiary directory table relating the values assumed by the field to the numbers of the records in which those values occur.

The indexing feature provides a tradeoff between the speed of retrieval and the size of storage required. When a field used in a condition is indexed, the system can focus very rapidly on the pertinent items without searching the data stream. This is

accomplished by maintaining a Field Value Table (FVT) and an R-Value Index Table (RVIT) which occupy additional storage space and must be updated each time a change is made to the set of values for the field.

The payoff for indexing a field is readily apparent when the alternatives available to the system in interpreting a condition are investigated. In the preceding section, the retrieval example included the condition:

VENDOR NO. = 3204

The attribute VENDOR NO. occurs in each record of the VENDOR file. There may be hundreds of such records. Since each record contains a subsumed file, the occurrences of the VENDOR NO. field are widely dispersed through the data stream. If the data stream must be searched to determine the records which meet the condition, there is a high likelihood that a different segment retrieval will be required for each record to be checked. This amounts to several hundred segment retrievals.

If the VENDOR NO. field is indexed, the values it assumes are stored in a compact file with a link to the list of record numbers which contain that value. There is a high likelihood that the first segment of the FVT retrieved will contain the desired value. If there is only one record meeting the condition, its value is stored in the FVT and the search is finished with one segment retrieval. If there are a number of records with the key value, the list of record numbers is maintained in the RVIT. A link in the FVT entry for the key value points directly to the record number list. In this case, the search is accomplished with two segment retrievals. Either way, the time saving is great. The payoff is greatest for records of files at a high level which contain embedded files, and it improves as the number of records in the file increases.

An indexed field is tagged in the Item List with a code specifying the type of indexing: all values, ranges, or selected values. The Item List entry for an indexed field contains a record number which identifies the specific Field Value Table for that field. When the system needs to determine the record numbers for the occurrences of an indexed field which contains a given value, the record number in the Item List entry converts the ICC of the FVT, 1.2.5.R.4, to an IPC. The correct FVT can be retrieved directly, and the entry containing the given value can be found. This record contains the record number sought, if there is only one occurrence of the field with the given

value. Otherwise, the FVT record contains a record number identifying a specific RVIT file which contains the list of record numbers for the occurrences of the field which contains the given value.

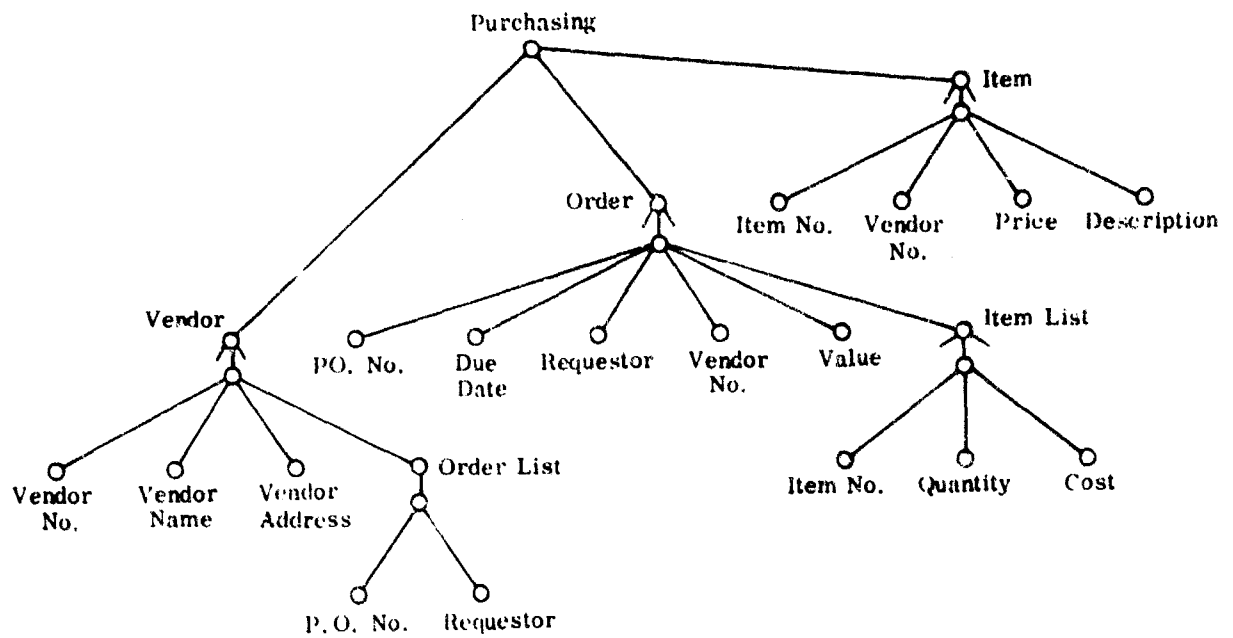
#### 4.6 LINKAGE

The data pool is basically a tree structure. Each node has a single parent node and may subsume a number of subnodes. In order to relate separate items in a tree structure, they must branch off from a common stem of the tree at the point they have in common.

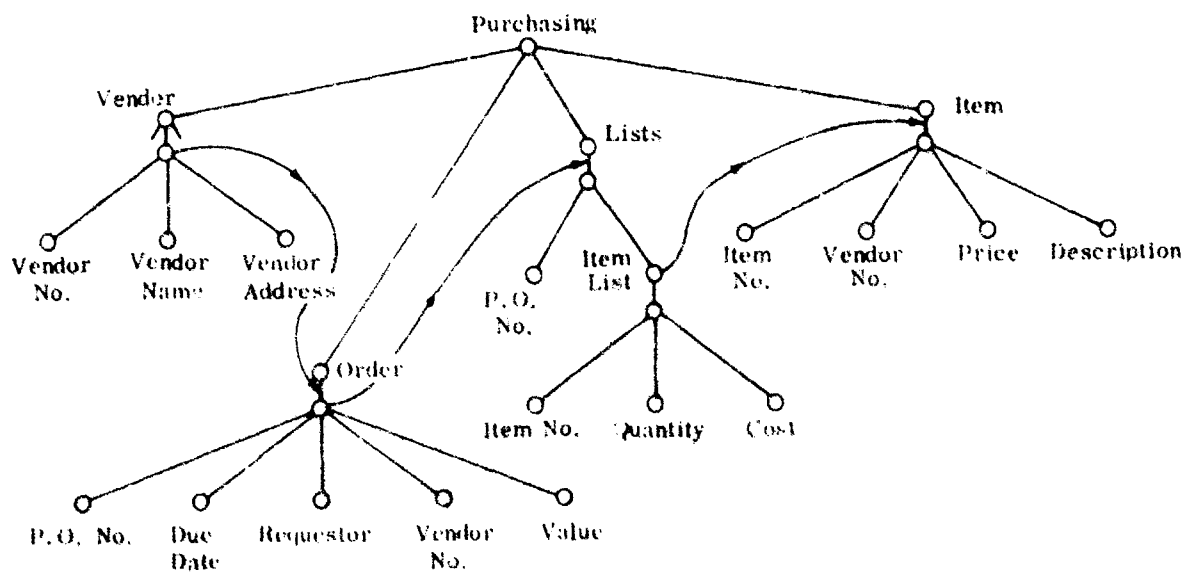
Figure 4-6 (a) is a node diagram of the pure tree structure for the PURCHASING item (Figure 4-2). The fields in the VENDOR file, VENDOR NO., VENDOR NAME, and VENDOR ADDRESS, are related because they are all attributes of a vendor. The list of outstanding purchase orders against the vendor, ORDER LIST, is also an attribute of the vendor. The relationship among these four items is shown in the tree structure by placing them as direct subitems of the record of the VENDOR file. The fields of the purchase order file, ORDER, and the list of items ordered have a similar relationship. They are all attributes that describe a purchase order, so they are defined as direct subitems of the record of the ORDER file.

The VENDOR file and the ORDER file are two elements of purchasing information. This relationship is shown in the tree structure by subsuming both files directly under the statement PURCHASING. However, there is another relationship between the two files which is not shown in the structure. All the attributes of a purchase order are pertinent descriptors for the purchase orders in the list for a given vendor. This could be shown by placing the entire set of purchase order attributes in each record of the file ORDER LIST. This results in a gross redundancy if the existing purchase order file is retained. If that file is eliminated, the purchase order information is available in the VENDOR file, but it must be grouped by vendor in that part of the tree structure.

It can be assumed that the high activity use for purchase order information is accomplished more conveniently if the ORDER file is retained in purchase order number sequence as a direct subitem of the PURCHASING statement. Also, the detailed attributes of a purchase order are needed only rarely when processing the VENDOR file,



(a) Pure Tree Structure



(b) Linked Network

Figure 4-6. Tree and Network Diagrams for the Purchasing Item

and such information is superfluous most of the time. In such cases, a logical link can be established to cut across separate branches of the tree structure. The link shows the relationship between its source and target items and allows it to be exploited while eliminating the redundancy of duplicate items and permitting the high activity data to be stored compactly. The high activity data can be processed much more efficiently when superfluous data is removed to a logically independent node, yet rarely used data can be associated with the source item when it is needed.

Figure 4-6 (b) shows the same purchasing information in a structure which makes liberal use of links. More of the relationships among the items are shown with less redundancy. The same item is shown in the structure diagram in Figure 4-7. The VENDOR file contains only the fields which describe a vendor. The list of purchase orders against the vendor has been replaced by a link to the records of the ORDER file. Each record of the VENDOR file is linked to the set of records in the ORDER file which have a VENDOR NO. field equal to the VENDOR NO. field of the VENDOR record. Logically, each VENDOR record subsumes a purchase order file for one vendor, with the full set of descriptors for each purchase order. Yet the VENDOR file remains a compact list of vendors' attributes and no duplication of data is required. The ORDER file may be maintained without reference to the VENDOR file. The set of records associated with a given vendor is automatically redefined by changes in the VENDOR NO. field (the link criterion) in the ORDER file.

Similar links have been established in Figure 4-7 to relate other items. The ORDER file contains a link to a record of the LISTS file where the list of items on that purchase order is maintained. The link criterion is the purchase order number. Also, each item in the ITEM LIST file is linked, by item number, to a record in the ITEM file. This makes the full item description logically available with each item on the purchase order without cluttering the ITEM LIST file for the majority of uses.

A link connects two items, i. e., a source and a target. The source link is an item with some characteristics of a statement. It subsumes the link criterion, a field whose value is the key to closing the link. The target link subsumes the criterion field in the target branch of the tree. When the values of the criterion field are equal in the source and target links, the target link's parent item is logically subsumed by the source link. The source link operates like a file whose records contain the items



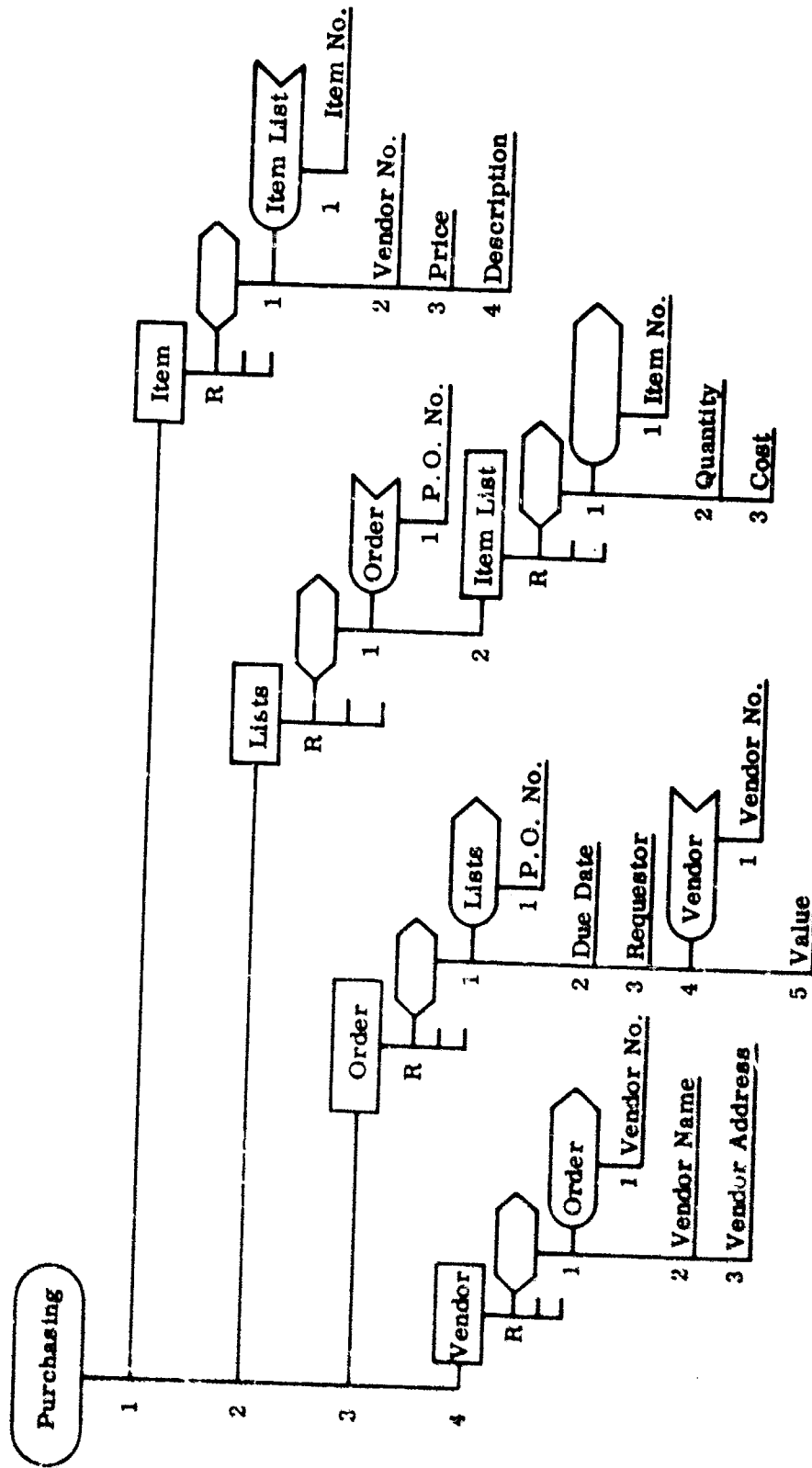


Figure 4-7. Purchasing Item with Links

of the target structure for each occurrence of a match between the source and target criterion values.

The system follows a link through a subsidiary directory, the Linkage Table. A link item contains a record number in its Item List entry. This leads to an entry in the Linkage Table, which contains the ICC of the matching link. Both source and target links have entries in the Linkage Table, permitting the system to follow a link in either direction. The target's substructure is subsumed by the source link, and the source's superstructure subsumes the target link.

#### 4.7 DATA INTEGRITY

A data protection system must provide for safeguards against accidental or malicious actions of authorized and non-authorized users and data destruction caused by hardware or software bugs (including failure of the data protection system itself). This objective can be stated in terms of the following sub-goals.

- (1) Data Security Checks must be provided to protect the user against invasion of privacy by protecting his data against unauthorized read and write operations.
- (2) Data Validity Checks must be provided to protect authorized users against collisions of data usage in a time-shared common data base and to protect the data base against illegitimate modification by authorized users.

The data protection mechanism is built into the resident reentrant Service Package which responds to all user data access and storage requests. Since these routines perform all data access and storage operations for all users, the constraints with regard to data usage for these routines must be inherited from their parent jobs and not be inherent in the routines themselves.

##### 4.7.1 Security Safeguard

The system of data protection employs two separate but interacting mechanisms: security level, and access/modification rights. Each data item class is assigned a security level for access and another level for modification from one of eight classifications.\* Likewise, each user receives a clearance level which gives him access and

---

\* In practice, security levels will range from 0 (unrestricted) to 6 (highest restriction), and clearance levels will range from 1 (lowest clearance) to 7 (no constraints).

modification rights to all items below his level. His rights to items classified at his level or above depend upon whether or not the item requested is on his access-rights or modification-rights list. A table of such rights, negotiated with the Data Administrator, is maintained for each user. A message to the Data Administrator is prepared for each unauthorized access or modification attempt.

Two degrees of access/modification rights will be recognized. The simpler is a right to a class of data given by item name, such as a file. A more discriminating right is to a particular subset of records in a file, where the subset is made conditional on a data check. An example of this is the right to specific raw data such as test results only if the data satisfies a condition, such as a given value in an identity field.

#### 4.7.2 Validity Safeguard

The approach to access or modification rights solves the problem of data integrity with regard to controlling access against unauthorized users. However, by itself, it does not protect the data base against destruction of data by system failure or by authorized users acting on the basis of invalid information. This aspect of the integrity-ensurance problem is solved by a combination of devices and procedures which hinge on the ability to identify and recognize the edition (or "generation") of data, both on an absolute basis and on a relative basis, i. e., relative to a specified "current" edition.

Edition control provides the ability to detect and control collisions in data usage arising from noncoordinated, overlapping read and write operations by independent users of a common data base. A graphic example of collision of data usage arises in an on-line reservation system in which two agents attempt to reserve the same space. When the agents query the system for space availability, the reply contains an edition number. When a reservation is attempted, the transaction must contain that edition number, which is checked by the system and incremented when the space status is updated. The reservation is then confirmed to the agent. An independent attempt to reserve space on the basis of the original status message will be rejected as the edition number check will fail.

#### 4.7.3 Item Lockout (Busybit)

Comprehensive data maintenance operations which perform structural modifications over multisegment data sets present additional conflict and protection requirements.

In such situations, multi-users read-write safeguards may not be sufficient to ensure valid operations if the data set being modified is used independently during the maintenance operation. To provide for such protection, temporary data lockout is provided by a "busybit" in the SNL entries for the data being modified. This bit, set and reset at the request of a maintenance job, effectively locks out use of any data in a class during the time the privileged maintenance job is running, thus ensuring that all data delivered to users is consistent with the Item List.

#### 4.7.4 General Procedure

There are two three-bit fields in each entry of the Item List assigning security restriction levels to the item, one for access and one for modification. Level 0 will designate unrestricted data and level 6 will designate the most highly restricted data. Restriction levels are assigned so as to be nondescending when moving from an item to its parent item. Each user is assigned a clearance level which gives him unconditional access to all data whose restriction level is below his clearance level. Access or modification to data classified at or above the user's clearance level is conditional upon whether the data belongs to the class of data for which the user has specific rights (Open Class) or meets a conditional check for unique items (Field Condition).

A user can be assigned a class of data items or specific data items expressed as a field-value condition, for which he has explicit rights. A rights check will be made only if a data request fails the clearance check. The usual rules of inclusion of items in an item class hold for the rights check. In the case of a field-value condition, the user is permitted access only to those records containing a field whose value is specified in the Field Condition list in his entry.

#### 4.8 THE DATA POOL

The data pool contains all items of data under the control of DM-1. This includes the data base, the system directories and job library, a set of application-oriented work items, and a set of transitory scratch items for task-to-task communication of intermediate data.

Figure 4-8 shows the entire data pool as a statement subsuming four items.

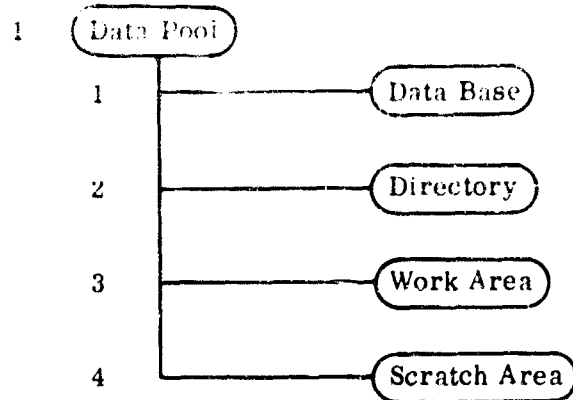


Figure 4-8. The Data Pool

The data base is a statement whose substructure is defined as the system evolves for a given application. It is under the control of the Data Administrator, and it represents the primary data resource of the system. The data base is the consolidated repository for data to be used in common in serving the information needs of the user group. Its structure might be relatively static after the initial transition period, but the data values change constantly to reflect the changes in the operations, events, and objects about which information is retained.

The directory is a statement whose substructure is defined the same way for any application of DM-1. It contains the system information describing the data pool and the programs under the system's control. The Item List, Term List, Term Encoding Table, Segment Name List, index files, link tables, data integrity tables, and job library files are all subitems of the directory statement. The structure of these items is constant. Section II of Volume II describes the structure of the directory in detail. The data items in the directory are changed by system jobs and routines to reflect changes in the structure of the data pool, segmentation of the data stream, composition of the job library, etc.

The work area is a statement which subsumes application-oriented items. They may be private files developed from some operations on the data base or items used by a class of users. The work area is the repository of data to be communicated from job to

job. Data derived by querying the data pool or developed by a user or system job may be stored in the work area, so that it can be used as input to later jobs.

The structure of the work area evolves under the direction of the system's users. Some items are defined by users and are used to contain relatively permanent data. Other items are defined by users and are used repeatedly to accept the output of a frequently called job. Still other items in the work area are defined by programs which create items in performing their function. For example, the query job defines the structure of the item it produces, based on the request. Its output might be stored as a work item with a user-assigned name and a program-assigned structure. On the other hand, a frequently requested query might be bound to a prestructure item in the work area at the user's request. The data in the work area changes primarily as a result of the execution of jobs whose outputs are bound to work-area items by the user.

The scratch area is a statement which is used to contain transitory items. These are items which are intermediate results in the execution of a particular job. Such items are created by the system in preparing for the job or during its execution and are deleted by the system at the end of the job. The substructure of the scratch item's statement is constantly changing to reflect the temporary storage requirements of the set of jobs being executed. The data values in scratch items are written by tasks of a job, read back by the same or other tasks, and destroyed with the corresponding structure definitions at the job's termination.

#### 4.8.1 Structure Development

The data base must be designed as a structure independent of its data content. The components of the structure are defined to the system with an item-definition request. An item-definition request is a call for the execution of the system job Define-Item. The parameters of the job are a node specification and an item image. The node specification identifies the node in the already existing structure of the data pool at which the new structure is to be placed. It gives the name of a reference item and the new item's relation to it. At first, only the statements DATA BASE and WORK AREA can be affected by an item-definition request. Any definition to be added must be the definition for a subitem of one of these statements; that is, the new item must be a file, statement, or field which is to be subsumed directly by the statement DATA BASE or the statement

WORK AREA. Once some subitems of these statements exist in the structure, a new item may be added as a subitem of any statement or record at any level in the data base or work area.

The item image in an item-definition request contains the name, item type, and other parameters for the item and each of its component items. The item image may be an indented outline, as shown for the PURCHASING item in Table 4-1, or a parenthetical string.

An item may be deleted from the structure by a call for the execution of the system job Delete-Definition. The only parameter of the job is the name of the node at which the structure is to be deleted. The job eliminates all subitems of the referenced node and converts the node itself to a null node. A new item may be placed at the null node by a later item-definition request. If the node itself is to be deleted, the Delete-Node job is called instead of the Delete-Definition job. The only way the structure of the data base can be modified is through the use of the system jobs for item-definition maintenance. The structure of the work area may be modified by these jobs also; however, work-area structure may also be defined indirectly. When a user calls any system or user job which produces an output item, he may request that a work-area item be automatically generated to serve as the job's output. He does this by supplying a name for the item in the job request and signalling that the structure is to be generated as a work-area item. For example, if a user calls the Query job to retrieve selected items under a given condition, he may have the results stored as a work-area item. This can be used later as input to some other job.

#### 4.8.2 Data Manipulation

A series of system maintenance jobs gives the user the ability to affect the data values in items of the data base or work area by commands issued at a console. The user may add or delete specified data values, modify values according to an arithmetic expression, or update a master file with values from a transaction file.

The item to be affected by the data manipulation job is specified by naming the item and providing a condition. The condition specifies the pertinent occurrences of the item within its parent files. For example, a condition might be used to identify the purchase orders of a given vendor, so that a data change would be applied only to them.

The Replace Data job replaces the values in the specified data items with the value of a source item. It can replace a field, statement, record, or file with new values for the item and all its subitems. If the condition specifies more than one occurrence of the items, all occurrences are replaced with the same source value. The Add-Data job functions the same way, except that the specified items must be empty (null value).

The Delete-Data job deletes the values which meet a given condition. The job converts all such values to null. If no condition is given the entire set of values for the item is deleted.

The Modify-Data job develops a new value for a specified field. The new value is the result of an arithmetic expression relating existing values and constants. This value replaces the value in all occurrences of the field which meet the condition.

The Update-Data job updates a master file with values from a transaction file. Each record of the transaction file contains a key which identifies a record of the master file. Other items of the master file are replaced with the values from the transaction file.

#### 4.9 CONDITIONAL RETRIEVAL

The fundamental strategy of DM-1 is to store data so that it is accessible to meet the information needs of users and programs. These needs are to be met by supplying to the consumer precisely that information which is pertinent and unencumbered by a context of irrelevant data. The ability to achieve this goal is provided by the conditional retrieval facility in concert with the system directories.

In a conventional system, information embedded within a file can be extracted by a program that contains within itself two kinds of implicit information: the implicit description of the file structure and the implicit procedure for discriminating between relevant and irrelevant data. In DM-1, an explicit description of the entire data pool is maintained in the system directories; it can be used repeatedly in response to the full range of information needs. The abstraction of relevant information from a broad context in the data pool is the function of the system programs for conditional retrieval.



#### 4.10 THE DIALOGUE

A user approaching the DM-1 data pool with an information requirement need not know the complete specification for the information. The system can help him to define the relevant items and the selection criterion through a dialogue procedure. The user is presented with a succession of multiple choices. Each time he makes a selection, he is presented with a more specific set of choices until he has defined his information requirement. This dialogue procedure encourages the user to probe the data pool to discover information relevant to his problem. In this mode of operation, he is in a position to apply his judgment continuously as he pursues a line of inquiry. The feedback at each stage in the process affects both the direction and degree of additional effort he devotes to the inquiry process.

The dialogue takes place in two phases which correspond to the two levels of definition of pertinence for information. In the first phase, the user is asked to choose the items which define the object, event, or operation about which he needs information. In the second phase, he is asked to provide criteria which pinpoint the individuals of the class of items selected in the first phase.

The tree structure of the data pool is the key to the dialogue procedure. When the inquirer indicates that he wants a dialogue, the system presents him with the names of the items on the highest level in the structure and asks him to select one for further probing. These are generic names which segregate the data base into logical groups. When the user selects an item, he is presented with the names of the items subsumed by that item, and so on, until he steps to the area of the data base containing the attributes of the object in which he is interested. He defines the relevant attributes and proceeds to the second phase. The process permits the user to backtrack, correct, and proceed at any point.

In the second phase, the user is asked to select the property of the object of interest which determine the relevant individuals. Each time he selects an attribute, he is asked to choose a value and a relation which defines the property the relevant item should have. If an attribute is indexed, the inquirer is presented possible values in digestible groups and he chooses the pertinent values. Otherwise, he is given an example of a value and asked to key in the pertinent values. Each term of the condition is developed by this procedure.

Based on the user's response to the displays, the logical operators relating the terms of the condition are inserted by the system. The user is guided to narrow the search by adding terms until a sufficient condition is developed as a logical product. He is then given the opportunity to broaden the condition by supplying alternative properties.

#### 4.10.1 Uses of the Dialogue Procedure

The dialogue procedure may be used to fulfill a number of information needs with respect to DM-1. It may be used to support other elements of DM-1 which function with a condition. The identification of the individual data items to be affected by a maintenance operation is accomplished with a condition. The selection of data as input to a job in a job request may also be conditional. The dialogue can be used to probe the data pool to define the precise elements for these conditional operations by developing a condition with system guidance.

An inquirer who approaches the dialogue with the need for an answer from the data pool elects to go on to a retrieval job after the dialogue. He specifies that the results are to be displayed to him or printed on hard copy.

A user who wants to perform further analysis on the information he has selected also selects a retrieval job as the last stage of the dialogue. He provides a name for the information and requests that the results be stored in the work area for further processing. He may specify a structure for the results, or he may accept the structure derived by the retrieval job from the relationships among the desired items in the data pool. When the retrieval is finished, the newly created item is available in the work area. It may be bound to any job for further processing by specifying the user assigned name.

Another user might have some new data to be added to the data pool, or he might have some other maintenance operation to perform. If he is uncertain of the precise nodes in the structure which should be affected by the operation, he may perform a dialogue. While determining the structure in the pertinent area of the data pool, the user may develop a condition which defines the precise data he wishes to change. He may request that the condition be stored with an assigned name. After the dialogue, the user may call the appropriate maintenance job and refer to the condition by name.

These examples show the range of uses for the dialogue procedure. Its primary value is to help the user of the system to formulate a specification of his information needs. But it may be used for purposes that range from a review of the structure to the development of a private file for analysis.

#### 4.11 RESTRUCTURING ITEMS

Restructuring is the process of mapping data from existing structures into an object structure, under the directions of a specification relating items of the object structure to items of the source structure. Restructures are performed in DM-1 for a number of reasons. Among them are:

- (1) To permit data to be collected conveniently in a form that is compatible with the source documents. Once the data is in the data pool, it can be transformed to a format that is oriented to its consumers.
- (2) To adjust to changing usage patterns. The history of usage for certain items might dictate that they be organized differently to improve the efficiency of operations.
- (3) To prepare data for analysis. A subset of the data pool may be selected conditionally and formatted into a structure that is convenient for processing. The resulting item may be retained in the work area and bound as input to any job in the library.
- (4) To transform data to meet the structural requirements of programs. Programs which were not designed to interface with each other may be combined in the same job. An output of one program may be bound to an input of another if a restructure program is capable of transforming the item into the format required by the second program.
- (5) To bind existing data to a job in a job request. This is similar to the task-to-task communication situation presented previously. When a user wishes to run a job, the data may not be in the appropriate form for the job. His job request may specify that the data be reconstructed to put it into the appropriate form.

## SECTION V. OPERATIONAL FEATURES

DM-1 is concerned with the management of a large, integrated data pool and its use in meeting the information needs of consumers. The system features associated with data pool management cannot be divorced from the operational features which provide the framework for interaction between the system and the user. The operational aspects of the system define the means for the specification of system languages, the development of the job library, the binding of data to programs, the management of job execution, and the servicing of the data needs of programs.

### 5.1 SYSTEM LANGUAGE SPECIFICATION

The means of communication between DM-1 and the system's users is through system languages. Descriptive languages specify item structures, report formats, program parameters, etc. Declarative languages contain the data of the system in external or internal format. Command languages evoke actions by specifying jobs to be executed and supplying parameters which affect the execution.

Each language type must be recognized at the appropriate level in the system by the appropriate processor. The processor must be constructed to respond to the

meaning of the language (semantics) by performing the appropriate actions (pragmatics). The semantics and pragmatics of the language are closely interwoven with the logic of the processor. However, just as the DM-1 approach separates the format of the data from the logic of the program, the system separates the language construction (syntax) from the logic of its processor.

The syntax of a system language may be described in a metalanguage similar to the metalanguage of ALGOL. The language description is maintained by DM-1 in a table called an action-graph. External languages are processed through the Input Scan Routine (INSCAN), which uses the action graph to direct the flow of logic. The external language may be converted to an internal form or interpreted directly by the execution of routines at action points in the syntax-directed scan.

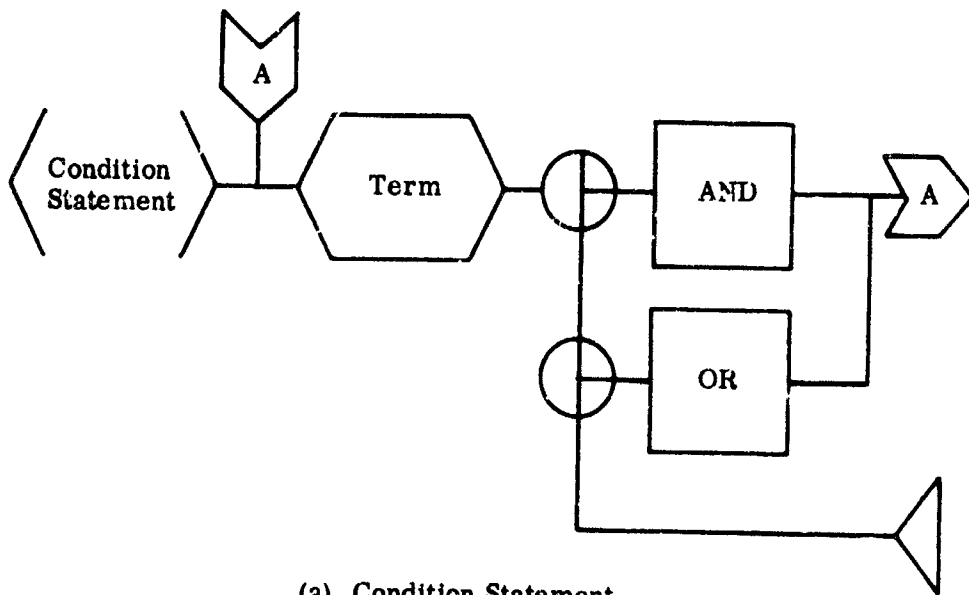
#### 5.1.1 The Metalanguage

The metalanguage is a language for describing the syntax of languages. DM-1 uses a charting technique to describe the construction rules for the components of a

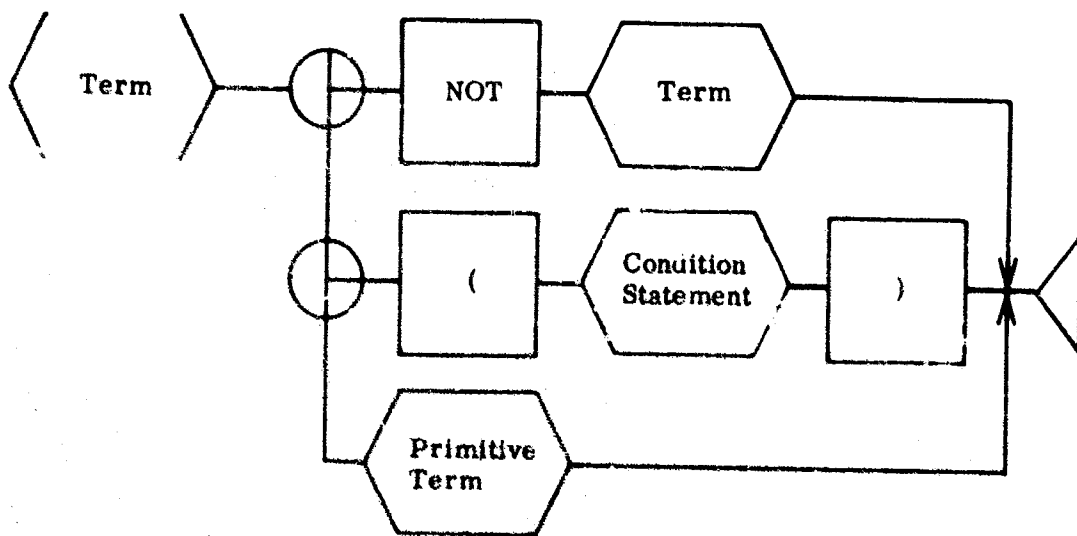
The syntactic chart can be converted directly to an action-graph. This separates the language processor from the language construction rules. The syntax may be changed without affecting the logic of the processor. A change is made by changing the action-graph.

The syntactic chart describes an element of a language by showing the symbols used and the order in which they occur. For example, a condition statement consists of terms connected by AND and OR. A term may be a primitive term, giving an item name, a relation and a value, or a smaller condition statement in parentheses. Figure 5-1 (a) is the syntactic chart for a condition statement. The chart says that a condition statement is a term which may be followed by AND or OR. If the term is followed by AND or OR, the next part of the condition statement must be another term. When a term is not followed by AND or OR, the condition statement is finished.

Figure 5-1 (b) is the syntactic chart for a term. There are three choices for a term. It may be a NOT followed by a term; an open parenthesis, followed by a condition statement, followed by a closed parenthesis; or a primitive term. The action graph for a primitive term (not shown) would specify an item name, followed by a relational operator, followed by a value.



(a) Condition Statement



(b) Term

Figure 5-1. Syntactic Chart

There are seven shapes used in a syntactic chart. They can be converted directly to codes in the action-graph table. The chart may include action points which permit the execution of routines of the language processor in the midst of the scan. The Input Scan Routine and the action-graph tables are discussed in detail in Volume II, Section VIII.

### 5.1.2 External Definition Languages

The structure of an item may be presented to DM-1 in an indented outline language or in a parenthetic string. The indented outline language is illustrated in Table 4-1. Its syntax is quite simple, and its syntax chart consists of a simple element with several action points to count the level of indentation.

The parenthetic string language specifies the structure of an item in an item image. The item image uses parentheses to delimit a statement and double parentheses to delimit a file. The slash is used to separate items on the same level in the structure. Figure 5-2 shows two structure diagrams and their corresponding item images. The first is a statement subsuming two fields, a statement, and another field. Its item image begins by giving the statement's name and its types. The rest of the item image, from the left parenthesis following the S to the last right parenthesis, describes the subitems of the statement. The field a is described by giving its name, type (U), and size. This is followed by a slash to signal the existence of another item on the same level. The field b is described and is followed by a slash. The statement B is described by its name, type (S), and the subitem description within the parentheses. The specification ends with the description of field e and the right parenthesis that closes the statement A. The other example is a file whose subitems are described in the same notation.

Figure 5-3 is a syntactic chart for an item image. It describes an item image as a name, followed by a comma, followed by a choice for the item type. If the type is a file, the rest of the item image consists of a left parenthesis, followed by a statement specification, followed by a right parenthesis. If the item is a statement, the rest of the item image is a statement specification. If the item is a field, a field specification follows. In practice, the syntactic chart would contain references to actions to be taken at various points in the scan. These references call on routines in the language processor to be executed when conditions in the input string warrant it. These action points are not required in Figure 5-3 to specify the syntax of an item image.

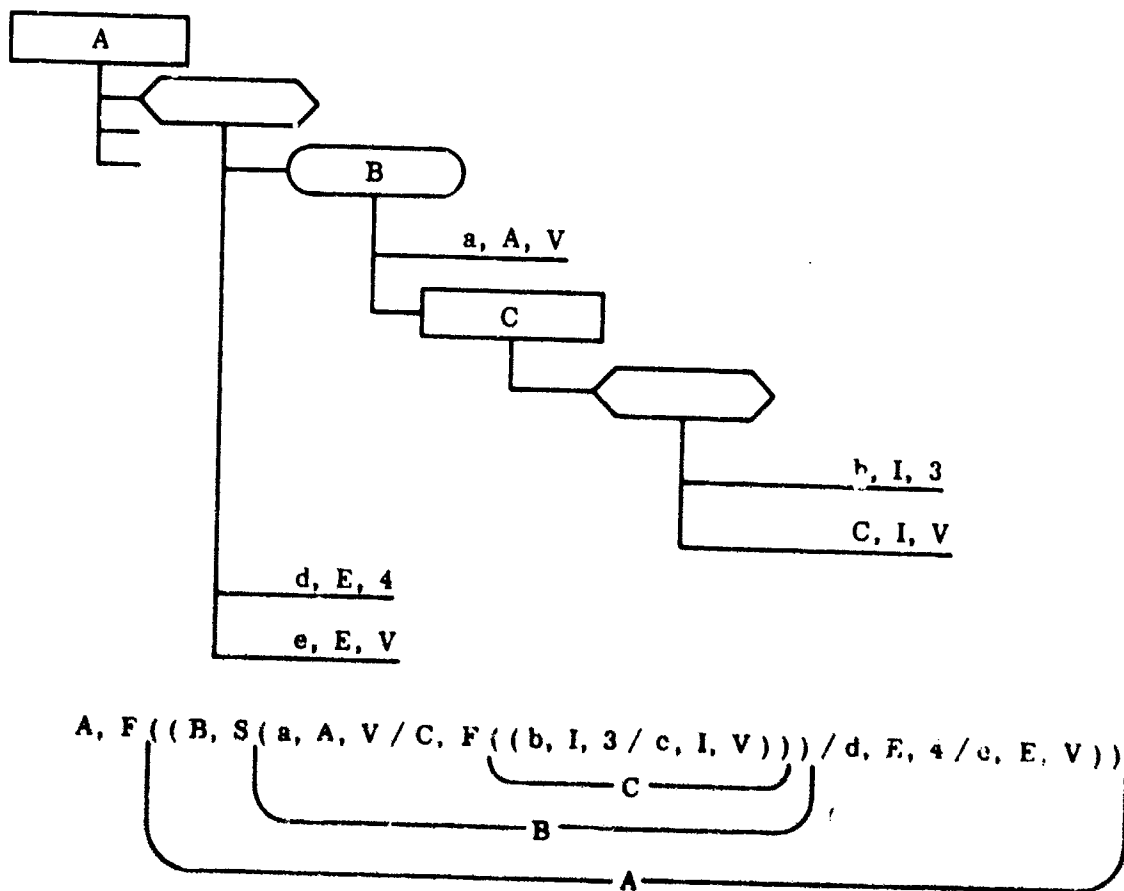
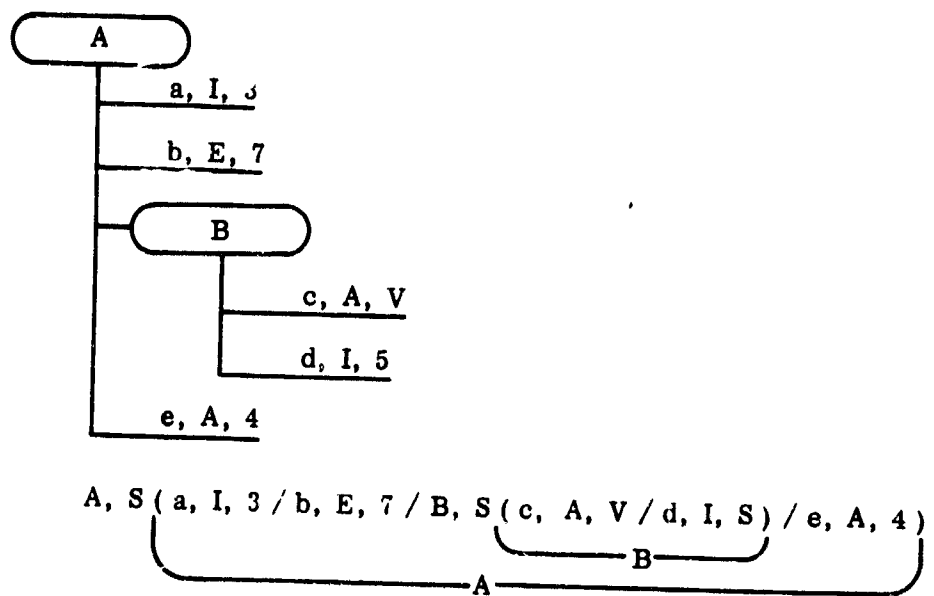
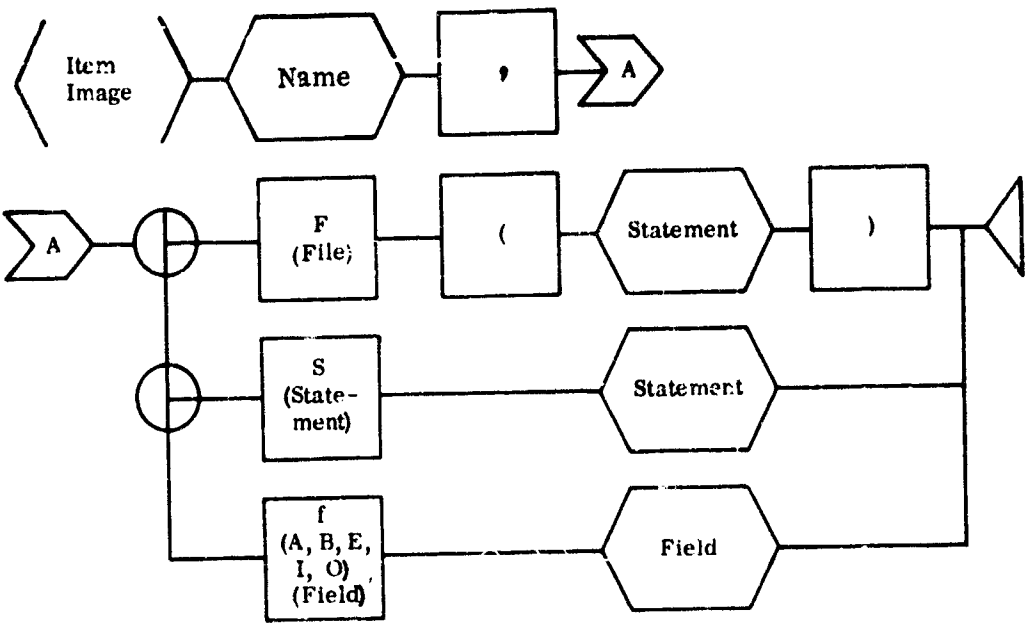
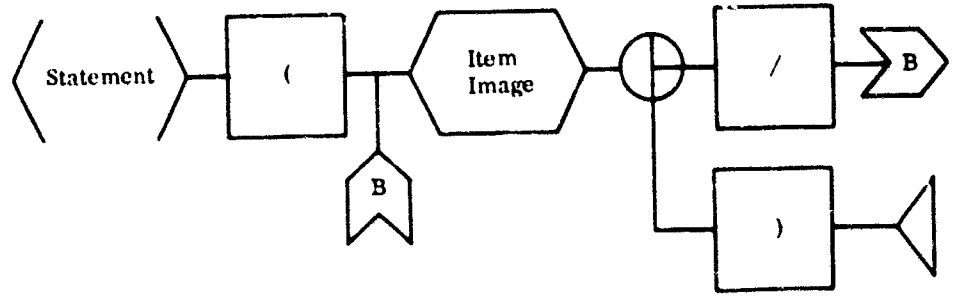


Figure 5-2. Structure Diagrams with Item Images

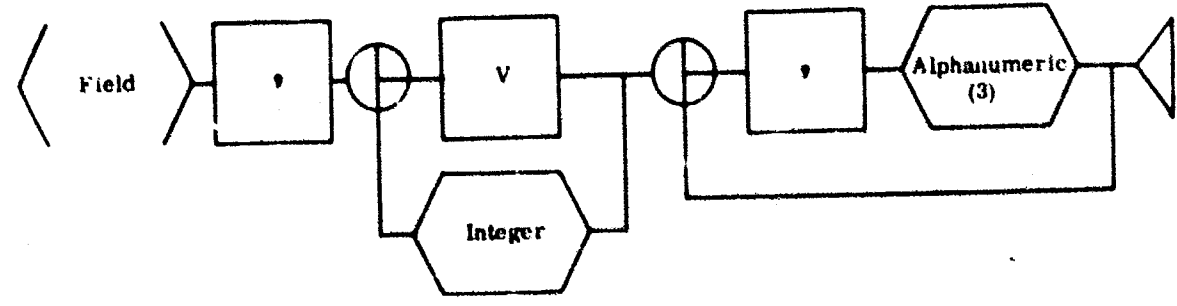




(a) Item Image



(b) Statement



(c) Field

Figure 5-3. Syntactic Chart for Item Image

### 5.1.3 Use of INSCAN with an Item Image

An item image is scanned by INSCAN when the subroutine is called by the language processor. The calling sequence identifies the text to be scanned and the action-graph to drive INSCAN. Suppose that INSCAN were called to scan the item image for the second structure of Figure 5-2 with the action-graph derived from the syntactic chart of Figure 5-3. INSCAN steps over the item image, a character at a time, responding to the tests specified in the action-graph. The steps of the scan are described below:

- (1) A. The first element of the input string is the item name for the file A. This passes the first test in the syntactic chart ITEM IMAGE.
- (2) ,. The next test specifies that the character after the item name must be a comma.
- (3) F. Following the comma, the syntactic chart offers a choice. The character may be an F, or S or one of the field types. In this case, it is an F, which indicates that the item is a file. The other choices are not tried, and the scan continues with the box following the test for F.
- (4) (. When the item type is F (file), the next character must be a left parenthesis. The structure of the records of the file will be specified by the statement image contained between this parenthesis and the closing parenthesis of the entire item image. This structure is specified by the characters:

(B, S(a, A, V/C, F((b, I, 3/c, I, V)))/d, E, 4/e, E, V)

This is a statement and the ITEM IMAGE chart specifies that the STATEMENT chart should be used to scan it. INSCAN saves a return point to the ITEM IMAGE chart in a return address list and control passes to the STATEMENT chart.

- (5) (. This is the left parenthesis of a statement (or record). It is detected in the first test in the STATEMENT chart. The statement (or record) consists of a series of subitems separated by slashes. Each subitem is itself an item image. The STATEMENT chart uses the ITEM IMAGE chart to scan each item. When the ITEM IMAGE chart is called, INSCAN pushes down the return address list, saves the return point to the STATEMENT chart and control passes to the ITEM IMAGE chart to scan the first item in the statement:

B, S(a, A, V/C, F((b, I, 3/c, I, V))

Each time INSCAN reverts to another chart (action-graph), the return address list is pushed down. The original chart,

ITEM IMAGE, was scanned on level 0. The STATEMENT chart was scanned on level 1. This call to the ITEM IMAGE chart places the scan on level 2. The levels continue to increase until a chart exits. Then the level is decreased by one and control passes to the return point on the chart which called the exiting chart. When the level is 0 and a chart exits, the scan is completed.

(6) B,S. These characters are scanned by the ITEM IMAGE chart on level 2. Since a choice exists for the type (F or S or f), the routine finds a match with S after failing the test for F. Since the item is a statement, the ITEM IMAGE chart calls for the STATEMENT chart on level 3.

(7) (. The statement begins with a left parenthesis. The STATEMENT chart then calls the ITEM IMAGE chart on level 4 to scan the first subitem of the statement:

a, A, V

(8) a,A. These characters are scanned with the ITEM IMAGE chart on level 4. When the first two tests for type fail, the test for a field is applied. A is one of the valid field types so the test succeeds. Control passes to the FIELD chart on level 5.

(9) ,V. The comma is the first character after the type on a field specification. This is followed by a choice of V or an integer specifying the size of the field. V stands for variable length.

The next element of a field specification is a comma if a unit designator is given. Since the next character in the input string is a slash, no unit specification is given. The FIELD chart exits bringing the level up from 5 to 4. The return point is to the ITEM IMAGE chart at the point after the call for the FIELD chart. Since the ITEM IMAGE chart has now scanned the complete item image for the field:

a, A, V

This chart also terminates. It was on level 4 so the level is decreased to 3 and control passes to the STATEMENT chart at the point after its call for the ITEM IMAGE chart.

(10) /. The completion of the ITEM IMAGE scan for the first subitem of the statement B places the STATEMENT scan on level 3 where it began in step (7). After any subitem, there may be another subitem or the end of the statement. This is the reason for the choice between / and ) as the

next character. Since the . is another subitem in the statement B, the slash is detected and the chart transfers on the same level to the call on the ITEM IMAGE chart to scan the next subitem on level 4.

- (11) C, F(. These characters are scanned with the ITEM IMAGE chart on level 4. Because the item is a file, the first parenthesis is checked and the STATEMENT chart is called on level 5 to scan the subitems of the records of the file.
- (12) (. The left parenthesis begins the record specification. The ITEM IMAGE chart is called on level 6 to scan the first subitem of the record.
- (13) b, I, 3. These characters are scanned with the ITEM IMAGE chart on level 6 and the FIELD chart on level 7 in the same way as the previous field was scanned. When the field specification is passed, control returns to the STATEMENT chart to complete the scan of the record's subitems begun in step (12).
- (14) /. The statement chart detects the slash and calls the ITEM IMAGE chart to scan the next subitem on level 6.
- (15) c, I, V. These characters are scanned with the ITEM IMAGE chart and the FIELD chart. When the scan is completed, control returns to the STATEMENT chart on level 5.
- (16) ). The right parenthesis marks the end of the record structure for file C. The STATEMENT chart exits to the ITEM IMAGE chart on level 4. It returns to the point behind the call on the STATEMENT chart to scan the record of file B. This continues the ITEM IMAGE scan begun in step (11).
- (17) ). This parenthesis marks the end of the item image for file B. It is detected with the ITEM IMAGE chart which exits to the STATEMENT chart on level 3. This continues the STATEMENT scan begun in step (7).
- (18) ). This parenthesis marks the end of the statement B. The STATEMENT chart returns to the ITEM IMAGE chart on level 2. Since this is the end of the item image for the statement B, the ITEM IMAGE chart exits to the STATEMENT chart on level 1. This continues the scan of the subitems of the record of file A begun in step (5).

- (19) /. The slash means that another subitem follows, so the STATEMENT chart calls the ITEM IMAGE chart on level 2 to scan the next subitem.
- (20) d, E, 4. These characters are scanned by the ITEM IMAGE chart on level 2 and the FIELD chart on level 3. When the scan is completed, control returns to the STATEMENT chart on level 1.
- (21) /e, E, V. The slash is detected by the STATEMENT chart. It calls the ITEM IMAGE chart to scan the field specification. Control passes back to the STATEMENT chart on level 1 after the field is scanned.
- (22) ). The right parenthesis (next to last in the string) marks the end of the record of file A. The STATEMENT chart has completed the scan begun in step (5). It exits to the ITEM IMAGE scan which continues the scan of the file A begun in Step (1).
- (23) ). The final character of the item image for file A is the right parenthesis. It is detected with the ITEM IMAGE chart on level 0. When this chart exits, the scan is completed because it was the language processor which requested the scan of the item image for file A rather than another chart. The routine exits to the processor.

#### 5.1.4 Data Language

The External Data Language (EDL) is the form in which data values may be presented on-line to DM-1 to be mapped into the data pool, under the control of the structure specification in the system directories. An off-line data entry mechanism is also provided to preprocess data into the Internal Data Language (IDL).

The off-line mechanism processes punched cards containing item definitions and data values in an external language. It converts the definitions into a directory and the data into the augmented stream of bits used internally by DM-1. These elements can be merged into the data pool through an IDL Data Entry job.

Data may be entered directly at the console or on punched cards or paper tape in the External Data Language. This is a punctuated string of field values in an alphanumeric code. The values are presented to the system in the order of the occurrence of the fields and files in the item definition. The presence of a statement in the item

definition has no effect on the order of presentation of items or the punctuation of the data string. The data string contains values for fields and files, with a slash between each value.

A field value is an alphanumeric string of characters consistent with the definition for the field. For example, an integer is presented as a string of decimal digits; an exponential field is presented as decimal digits with a decimal point and a scale factor, if required; an alphanumeric field is presented as a string of alphanumeric characters; etc. The length of the value string must be consistent with the definition. A value for a variable length field may be arbitrarily long. A value for a fixed length field may not exceed the length specified in the field's definition.

A file value consists of a series of record values which, in turn, consist of a string of field values and embedded file values. A file value is bounded by parentheses and the data string for each record is delimited by parentheses.

At any point in a data string, the name of the following item may be inserted. A name is signalled by an asterisk. The name forces the EDL processor to position itself to the specified item. There are three reasons for inserting the name of an item in the data string. First, it supplies redundancy so that the entire string is not lost if the data coder skips a value or gives too many values in a part of the string. Second, the name permits the coder to skip a series of items for which he has no values. All items between the one corresponding to the last value and the named item are set to null by the EDL processor. Third, it is used to announce a value for an optional item. Any optional item is presumed missing unless its name appears ahead of the value for the item.

The EDL data string contains a value for a statement, file, or field that is consistent with the syntax shown in the chart in Figure 5-4. A statement value is a string of field values and file values for the subitems of the statement, with no delimiters surrounding the statement. A file value begins with a double left parenthesis, ((, one to open the file and one to open the first record. The boundary between records is marked by a right parenthesis and a left parenthesis, ) (, one to close the previous record and the other to open the next record. The end of the file is marked by a double right parenthesis, )) , which closes the last record and the file, respectively. If the data string contains only a field value, no delimiters are used.

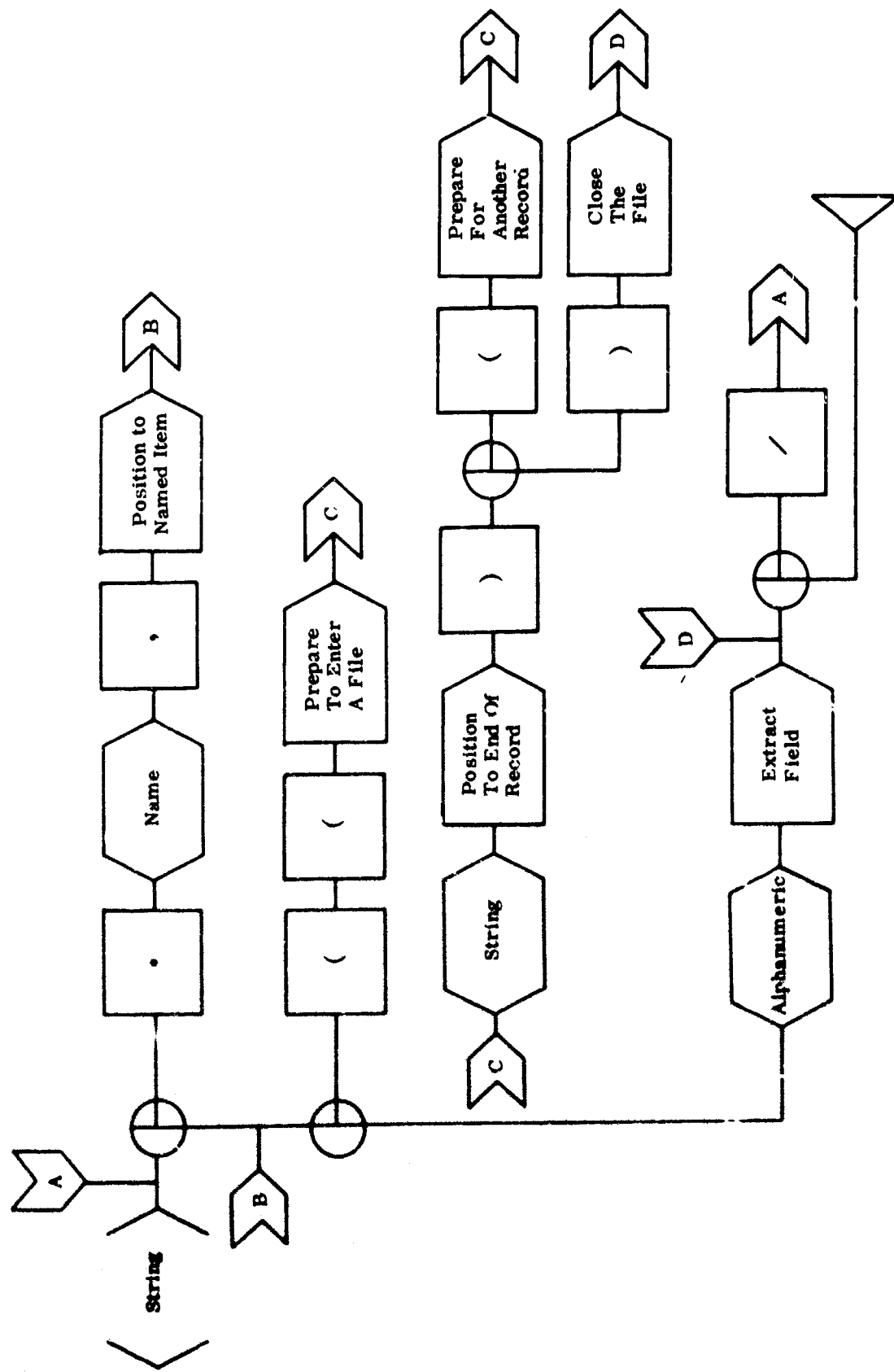


Figure 5-4. EDL Syntactic Chart

An item image for file A of Figure 5-2 is repeated below with a possible data string as an example. The steps taken by the Input Scan Routine in interpreting the data string with the syntactic chart of Figure 5-4 are then presented. The notation Va is used to represent a value for the field named a.

**EXAMPLE —**

Item Image:

A, F((B, S(a, A, V/C, F((b, I, 3/c, I, V)))/d, E, 4/e, E, V))

File C

Statement B

File A

Data String:

|        |          |   |
|--------|----------|---|
| File A | Record 1 | ((Va/((Vb/Vc) (Vb/Vc) (Vb/Vc))/Vd/Ve)   |
|        | Record 2 | (Va/((Vb/Vc) (Vb/Vc))*d, Vd/Ve)         |
|        | Record 3 | (*C, ((Vb/Vc) (Vb/Vc) (Vb/Vc) (Vb/Vc))) |
|        | Record 4 | (Va/*e, Ve)                             |

- (1) ((. The syntactic chart in Figure 5-4 is supplied to the INSCAN Routine (level 0) by the EDL processor. A data string may begin with an asterisk, a double left parenthesis, or a field value. The double left parenthesis is encountered in this example, and INSCAN executes the processor's routine which prepares for a file. This is an example of an action in the midst of a syntactic chart (or action-graph). The STRING chart is called on level 1 to interpret the string for the first record of file A.
- (2) Va/. The first thing encountered with the STRING chart on level 1 is a value for the field A. A processor action is executed to extract the value. Since the value is followed by a slash, the scan transfers to the start of the chart, which is still on level 1.
- (3) ((. An embedded file (file C) is entered in the first record of file A. The STRING chart is called on level 2 to scan the string for the first record of file C.
- (4) Vb/Vc. The value for field b is scanned and extracted. Since a slash follows, there is another item in the string and the scan transfers to the start of the chart, which is still on level 2. The value for field c is extracted. Since



the next character is not a slash, there are no more items in the string on level 2 (the first record of file c). The STRING chart exits to level 1, and control is picked up at the point after the call to the STRING chart. An action is performed in the processor to ensure that all subitems of the record have been received. If any are missing, the processor fills out the record with null values.

- (5) ) (. The right parenthesis marks the end of the first record of the embedded file c within the first record of file A. Following a record end, a choice exists. The next character might signal the beginning of another record or the end of file. In this case, the left parenthesis signals the beginning of another record. The STRING chart is called on level 2 to scan the second record of file c.
- (6) Vb/Vc) (Vb/Vc. The second and third records of file C are scanned like the first record.
- (7) )). The double right parenthesis marks the end of the embedded file C in the first record of file A. The processor performs the actions required to close the file and transfers to point D on the chart to check for a slash.
- (8) /Vd/Ve. These field values are scanned on level 1 and extracted.
- (9) ) (. The right parenthesis marks the end of the first record of file A and the left parenthesis signals the beginning of another record. These are detected on level ø and the STRING chart is called on level 1 to scan the string for the second record of file A.
- (10) Va/((Vb/Vc) (Vb/Vc))/. The values for the field a and the file C are scanned and acted upon as before. The slash signals that another item exists on level 1.
- (11) \*d, Vd/Ve). The asterisk is detected by the first choice at point A on the STRING chart. It signifies that an item name follows. The processor steps to the named item. In this case, it is already at the named item, so the name serves only as a check. The value for fields d and e are scanned and extracted. The chart exits to level ø when no slash is found after the value for field e. The right parenthesis is detected by the chart on level ø, marking the end of the second record of file A.

- (12) (\*C(Va/Vb)(Va/Vb)(Va/Vb)(Va/Vb)). The left parenthesis marks the beginning of the third record of file A. The level  $\emptyset$  chart calls the STRING chart on level 1 to scan the record value. The name C is the first element in the data string. In positioning to the item C in the definition, the processor inserts a null value for the field A, since that item is skipped in the data string. The value of file C is scanned and extracted as before. The first two right parentheses in the triple mark the end of the embedded file C. Since no slash follows, the chart exits to level  $\emptyset$  again. Values for fields d and e were skipped in this third record of file A. This is shown by the third parenthesis following the close of file C. The processor action fills out the record with null values.
- (13) (Va/\*e,Vc). The fourth record of file A does not contain a value for file C. In positioning to the named item e, the processor action will mark the file null (no records).
- (14) ). The final right parenthesis marks the end of file A. The chart exits on level  $\emptyset$ . Control is returned to the EDL processor, where a check is made to ensure that the entire item was filled out with values.

#### 5.1.5 The Job Request Language

A user controls the work performed by DM-1 by job-run requests in the Job Request Language. He may request the execution of any system or user job in the library by typing a request at the console. The Job Request Language provides a broad spectrum of flexibility by permitting simple requests consisting of a job name, or complex requests containing binding specifications for many job parameters with qualifications directing conditional selection and restructuring of data items, to be used as job requests.

A job-run request is patterned after the job-description image. This consists of the job name followed by a list of the names of the job's formal input parameters and formal output parameters. For example, the job-description image of a job which determines the equation of a line from a set of coordinate points might be:

REGRESSION points; line

where the job's name is REGRESSION, its input parameter is a file called points and its output parameter is a statement called line which give the coefficients of the line. When a user wishes to execute the job, he binds the input and output parameters in a job-run request. He replaces the formal name points in the job-description image with

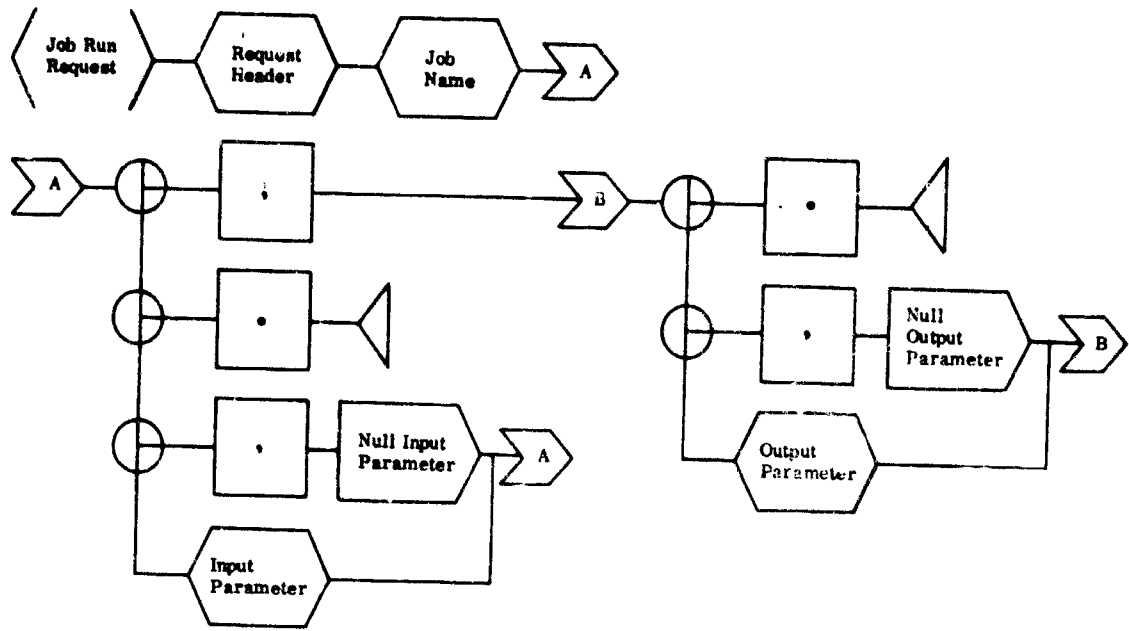
the name of a data-pool file containing the coordinate points or with a literal file typed at the console. He also binds the output parameters by replacing the formal name line with the name of a statement in the data pool or with a name to be assigned to a work-area item to be created by the system to accept the job's output. For example, a job-run request might be:

REGRESSION test data; \* failure line.

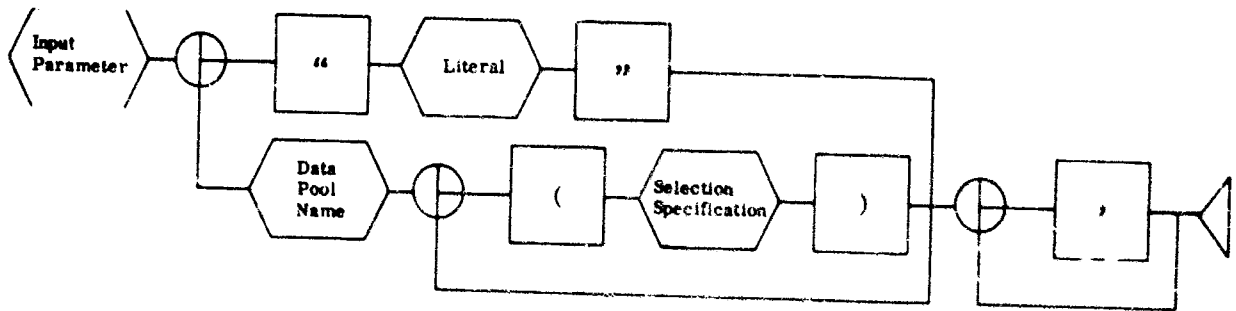
where test data is the name of a file and failure line is a name the user wishes to assign to the job's output, which is to be written into the work area of the data pool.

Figure 5-5 is the syntactic chart for the Job Request Language. The chart for a job-run request is given in Figure 5-5 (a). It specifies that a job-run request begins with a request header, which identifies the user and is in a common format for all requests. The rest of the request is the bound job-description image. It consists of the job name followed by a choice of four components. A semicolon occurs if there are no input parameters to be bound, or after the last input-parameter binding specification. It announces the beginning of a set of output-parameter binding specifications. A period occurs after the job name and terminates the request if there are no parameters to be bound. If there are parameters, the period follows the last parameter binding specification. A comma normally separates binding specifications. The appearance of a comma immediately after the job name implies that the job has several input parameters and that the first parameter will not be specified. In other words, if the comma is used, but is not preceded by a parameter specification, the formal parameter in the corresponding position in the job-description image is not to be bound for this run. If none of these punctuation characters follows the job name, an input-parameter specification was given. Following the parameter specification or a null input parameter, the chart of Figure 5-5 transfers to point A to repeat the scan for the next element of the request. After the last input-parameter specification, a semicolon signifies the beginning of the output parameters, or a period specifies that no output parameters are to be bound.

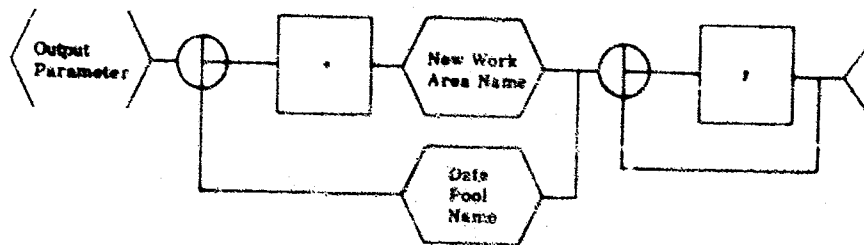
If the semicolon is encountered, the scan passes to point B in the syntactic chart. A period signifies that no more output parameters are to be bound. A comma specifies a null output parameter. Otherwise, an output-parameter specification occurs at this point in the request. After the output parameter on a null specification, the chart transfers to point B to repeat the scan for the next parameter. After the last parameter specification, a period marks the end of the request.



(a) Job Run Request



(b) Input Parameter



(c) Output Parameter

Figure 5-5. Job-Run Request Syntactic Chart

Figure 5-5 (b) is the syntactic chart for an input-parameter binding specification. It is referenced by the chart for a job-run request. A quotation mark signifies that a literal value for the input parameter is given in the request. The literal is terminated by a closing quotation mark. If the specification is followed by a comma, the character is scanned over before the chart exits. If the input parameter specification is not a literal, it must be the name of a data-pool item. This may be followed by a selection specification enclosed in parentheses. A selection specification may contain a condition which identifies a subset of the named item as the data to be fed to the job. It may also contain a reformat specification which specifies that the data should be fed to the program under a structure definition that is different from its current definition. Again, the comma is scanned over if it appears after the input-parameter specification.

The syntactic chart for an output parameter is given in Figure 5-5 (c). An output parameter is either a data-pool name or an asterisk followed by a user-supplied name. The latter is a request to the system to define an item in the work area to receive the output parameter and assign the given name to it. The item can be used later as input to another job for further processing or display.

## 5.2 JOB-PARAMETER BINDING

Another of the operational features of DM-1 is its control over the binding of data items to programs. In a job-run request, a user may specify the data to be operated on by its symbolic name and the system associates the data with the program's input and output requests. The system also coordinates the communication of data from one program to another within the framework of the job.

### 5.2.1 Programs and Jobs

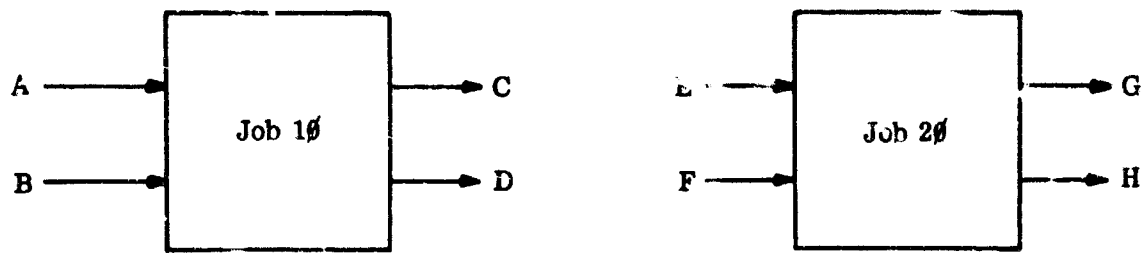
A job in the DM-1 library may be viewed as a black box with certain inputs and certain outputs. Each input and output is a data item with a certain structure. These are formal parameters of the job. Each parameter has a name. Parameter names may be assigned by the programmer when he writes a program and used by him to reference the parameter in calls to the system. Or, parameter names may be assigned by a user when he creates a job from existing components.

When a program is written for DM-1, the programmer assigns formal names to each of his input and output data items. These formal parameters may have a fixed

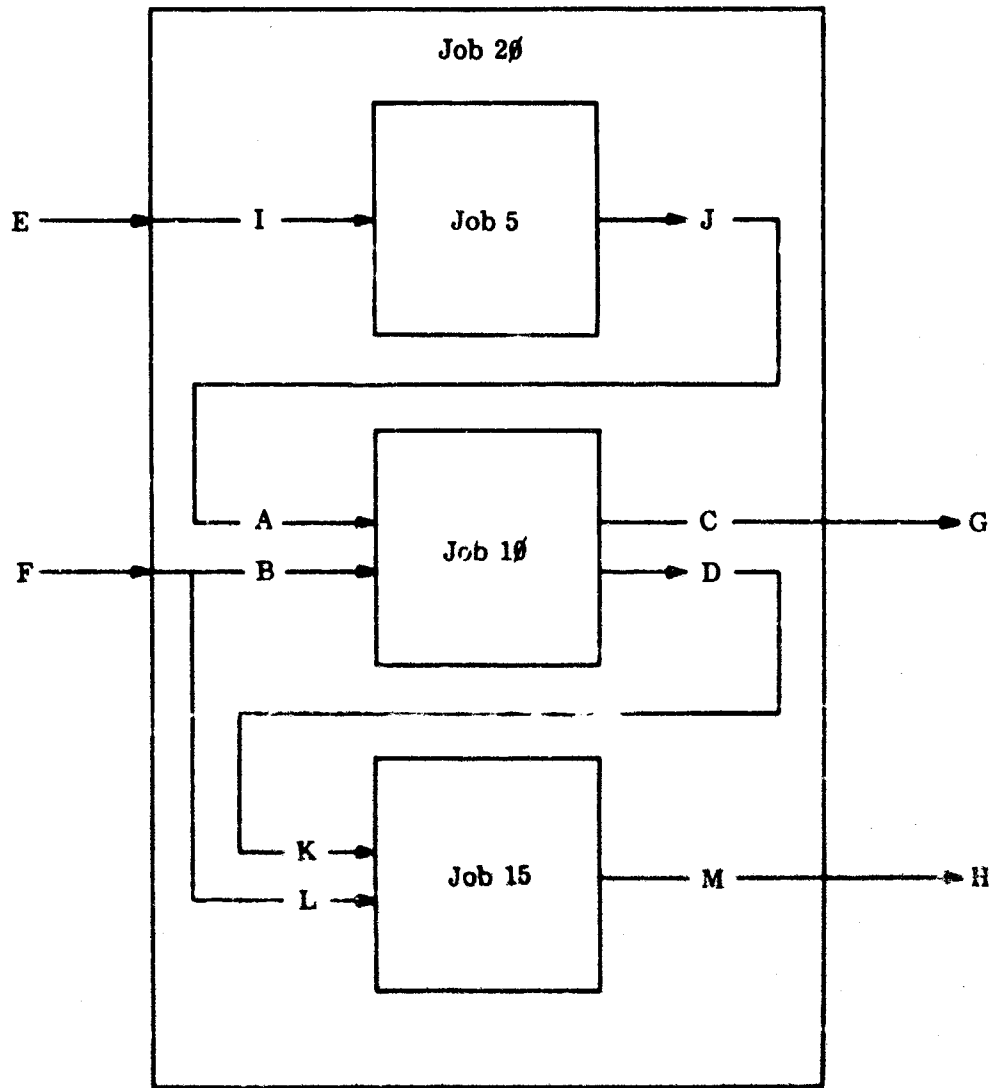
structure, in a the sense of a specified item definition, or they may have a variable structure. If the structure is fixed, the program is written assuming that the fixed structure exists in the data pool and all calls to the system associated with that data item relate implicitly to the fixed structure. If the structure of a parameter is variable, the program is written to operate with a system-supplied Item List Table for input parameters to determine the structure with which it is operating; or, the program must supply such a table to the system to specify the structure of output parameters.

After a program is written, it is entered into the DM-1 library by the execution of the Program Entry job. Each of its formal parameters is described to the system by an item definition. The system places the program's name and its parameter specifications into the library. This step makes the program a job. Its job-description image is the program's name (job name) with the list of formal input and output parameters. Once the job is in the library it may be used, like any other job, as a component in more complex jobs, as a subroutine called through a job extension from a program, or as a unit of work callable by a user with a job-run request. Effectively, it becomes a black box with specified formal inputs and formal outputs which can be bound to data-pool items, to the inputs and outputs of other jobs, or to data supplied by a program in a job extension.

Figure 5-6 (a) shows two jobs as black boxes. Each has two inputs and two outputs. As black boxes, they are very similar. However, JOB1# is a single-task job; i.e., it is a program which was entered into the system through the Program Entry job. JOB2# is a multitask job; i.e., it was described to the system with the Job Description job as consisting of a sequence of jobs which already existed in the library. Figure 5-6 (b) shows JOB2# as a sequence of three other jobs; JOB5, JOB1#, and JOB15. The inputs to JOB2# are actually inputs to its components, and the outputs are actually component outputs. The inputs and outputs of the individual components are related in a fixed way by the job description which created JOB2#. Once the job is created, it is eligible to be called and executed by a user at a console or by a program through a job extension. It is also eligible to be a component in another job. (In the sense that a job may be defined in terms of jobs, the definition of a job is recursive.) In any of these uses, the user may ignore the internal structure of the job description. Only the formal input and output parameters of the job need be considered. The system takes care of



(a) Two Jobs as Black Boxes



(b) Components of Job 2

Figure 5-6. Jobs as Black Boxes

relating the data items bound to the formal parameters of the job, through an arbitrary number of levels of indirect references, to the terminal program which actually processes the data.

### 5.2.2 Binding Specification

In describing a job as a sequence of existing jobs, the user may assign formal names to job-input and job-output parameters to be bound when a run request is issued for the job. He uses these formal job-input names, data-pool names, and formal component-output names to define the formal inputs of the component jobs. Similarly, he defines the formal outputs of the component jobs by assigning them to data-pool names, formal inputs of other components, or formal outputs of the job.

When a sequence of independent jobs is defined as a new job, the output of one component of the sequence may be semantically compatible with the input of another component but the format requirements may differ. To equate such parameters, the binding specification contains a reformat clause. The reformat clause specifies a structural transformation to take place in interpreting the source item as an input to the second component. When the input requires a subset of the source item, the binding specification contains a condition clause. The condition is used to select a subset of the source item which is to be used as the input item. For example, one component of a job may produce a file containing a record for each employee. Another component processes similar records for employees in the stock-option plan. A condition may be used in binding the employee file to the second component, so that only the records for employees who meet the requirements of the stock-option plan will be fed to the second component. The reformat clause and the condition clause may be combined in a binding specification to produce a subset and a structure transformation.

The same requirements for parameter binding occur in job-run requests. Items in the data pool may be bound to formal job inputs with a binding specification that contains a reformat clause and a condition clause.

### 5.3 RELATIONSHIP TO THE OPERATING SYSTEM

DM-1 is designed to function in a multiprogrammed environment in conjunction with an operating system which controls the environment. Wherever possible, the system uses the features of the operating system to enhance its performance. However,



there are operational features of DM-1 which replace similar features of the operating system, so that the full range of flexibility and power of DM-1 may be exercised. The characteristics of the job-run request, the features for the binding of parameters, and the mechanisms for accessing and storing data symbolically under directory control are not accounted for in the features of a general-purpose operating system. The following summary indicates the features that must be provided by parts of the DM-1 system itself:

- (1) To provide the ability to respond to job-run requests, the DM-1 Request Processor is required. A user must be able to call for the execution of jobs from the DM-1 library, using the data binding features of the system. Since the maintenance of the structural requirements for the parameters of programs and jobs is a DM-1 feature, the library is not under the control of the general operating system.
- (2) To provide the flexibility of DM-1 during the execution of the job, the Job Manager is required. It must operate with the output of the Request Processor and monitor the loading and execution of each of the programs in the job. It must respond to program calls to execute other jobs in the library as subroutines (job extensions).
- (3) To provide data access and storage services to programs during their operation, the Service Package is required. It uses the directories and the data binding information produced by the Request Processor and acts as an intermediary between programs and the data pools.

Several approaches could be taken to accommodate the DM-1 requirements within the operating system. The two extreme positions help to define the choices open to the system. At one extreme, the DM-1 system could be organized to function as a job in an existing operating system. It would appear no different from any other job executed under the operating system's control. At the other extreme, the scope of DM-1 could be expanded to incorporate all the features of the operating system, namely an integrated system to control such elements as multiprogrammed scheduling, device control, and resource allocation, as well as management of the data pool and job library. Programming language processors could be modified to incorporate language elements of the DM-1 system.

If the DM-1 system is to function as a job under the operating system, all system facilities must be provided for within the job. The response to a job-run request could be handled by the first stage of the job, and the appropriate coding to execute the request could be loaded as overlays or subroutines. However, space would be required in the memory assigned to the job for the system service routines as well as the code of the task programs. These routines would be duplicated in every DM-1 job. This would severely limit the number of DM-1 jobs which could function together in the time-shared mode.

If an operating system is developed to include the DM-1 system, the coordination between the systems would enhance performance. The programming language processors would contain facilities for calling DM-1 services and operating on structured items. The scheduling algorithms could take advantage of the degree of control exercised by DM-1 over the data. Efficiencies could be introduced by combining levels of storage control. However, an operating system and its associated language processors represent a large investment in design and development. Its valuable features cannot be discarded lightly.

DM-1 takes a compromise position that is based on an evaluation of the planned implementation under the RADC Mobile Wing ECP-1 for the M-1218 computer. The details of the planned relationship between DM-1 and the operating system are given in an appendix in Volume II. DM-1 relates to the operating system in such a way that it appears as an embedded part of the system to a program, while it appears as a job to the operating system. A request for a DM-1 job is issued at a console as data from the viewpoint of the operating system. This data is for the DM-1 system which is recognized by the operating system as a single job. The DM-1 Request Processor decodes the request and the DM-1 Job Manager supervises its execution within the confines of a single operating system job.

The DM-1 job has a special status within the operating system. Whenever it is requested, the operating system ensures that the DM-1 Service Package is resident in memory before initiating the DM-1 system as a job. Since the Service Package is reentrant, only one resident copy is needed to serve many DM-1 programs initiated under separate requests. This artifice permits the system to take advantage of the multi-programmed environment, without duplicating the DM-1 service routines in every program.

The operating system permits programs to call for DM-1 service routines through its normal executive call with a code signifying the Service Package. It passes control to the resident Service Package which decodes the request and responds to it.

The interface between the operating system and DM-1 is a narrow one, which permits easy transfer of the DM-1 system to another environment with a different operating system. Improvements or changes in either DM-1 or the operating system should have little effect on the other system.

#### 5.4 REQUEST PROCESSOR

The DM-1 Request Processor is initiated by the Job Manager for each user request for the DM-1 system. It reads the console message which contains a job-run request in the Job Request Language. The job-run request contains the name of a job in the DM-1 library and the binding specifications for the job's input and output parameters. The Request Processor uses the Input Scan Routine with the action-graph for a job-run request to decode the message. It uses the job name to locate the job's description in the library. With the description as a guide, the request processor constructs a request record containing the information needed to execute the sequence of programs (tasks) that make up the job. This record is written into the request file to be used by the Job Manager and Service Package during the execution of the job.

The request record contains the Task List which identifies the sequence of programs to be executed for the job. The Task List is derived from the job description and the job-run request. It contains identifiers for the component programs of the job and for any implied tasks needed to achieve the conditional reformatting of parameters specified in the job-run request.

The Task List is a file embedded in the request record. Each record of the Task List contains two parameter binding files, one for the task's input parameters and one for its output parameters. The records of the parameter binding files are equations relating the formal parameter names used by the program in service calls to the data-pool identifiers of the actual data bound to the program for the run.

##### 5.4.1 Request Translation

Figure 5-7 is a schematic diagram of the steps taken by the Request Processor in translating the job-run request into the internal request record. The user begins the

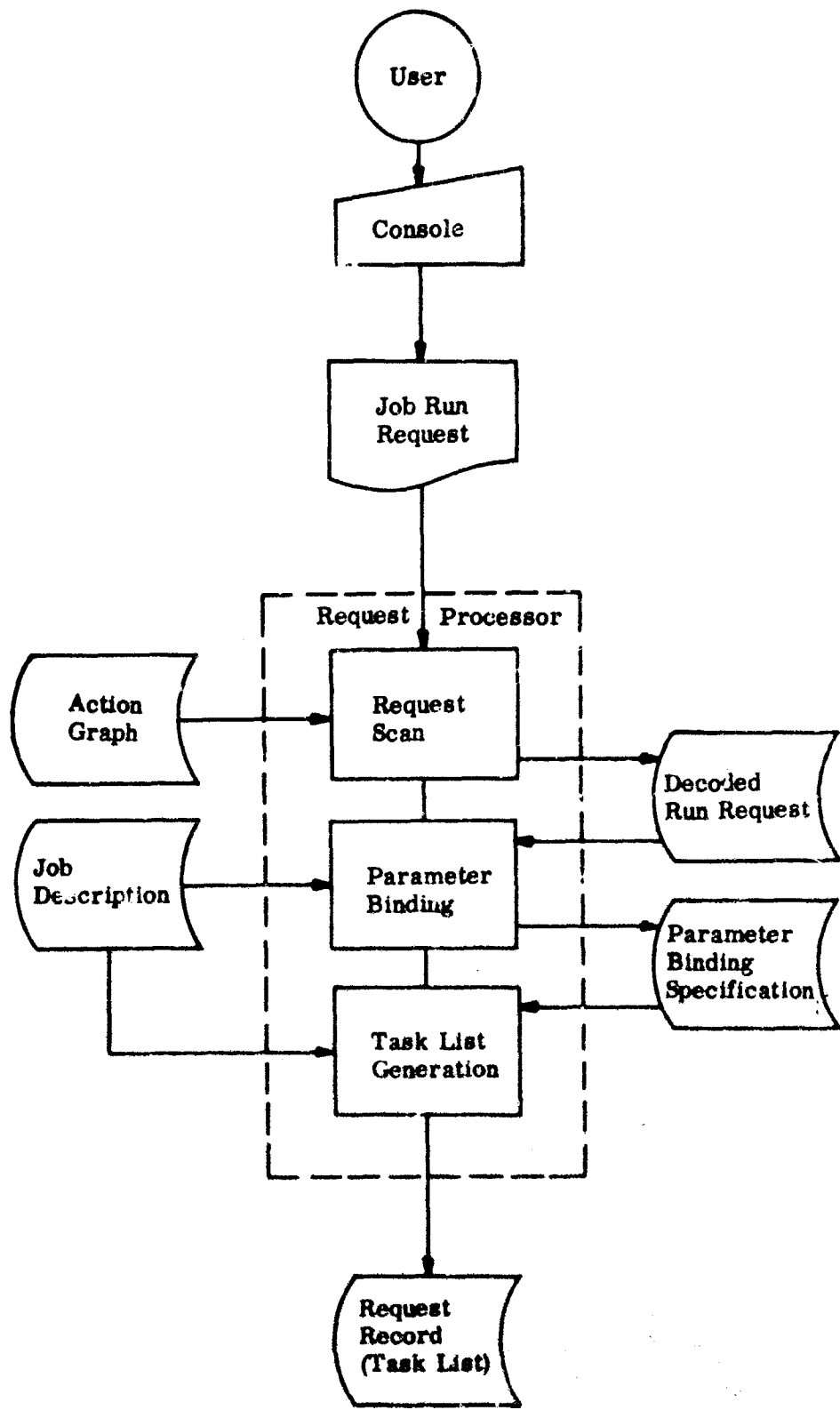


Figure 5-7. Request Translation

process by developing a job-run request at the console and signaling the operating system that a DM-1 job is to be executed. The operating system schedules the Job Manager, assigns memory to it, and loads it. A standard set of tasks for the Request Processor is executed by the Job Manager. The Request Processor reads the job-run request through an operating system service. It retrieves the action-graph for the Job Request Language and uses the Input Scan Routine to scan the request, check its syntax, and decode it into an internal form. The decoded run request and the job description from the library are used in binding each of the job's input, output, and intermediate data parameters. Each parameter is assigned an IPC in the structure. Any implied tasks needed to reformat items or select data subsets conditionally are scheduled for inclusion in the Task List. These implied task requirements and the IPC's assigned to the job's parameters are stored in the parameter binding specification. This is an input to the last step of the process, the generation of the Task List for the requested job. The Task List, which contains one record for each task, is written into a record of the request file. A binding record for each of the task's input and output parameters is written into the input or output binding file, which is embedded in the Task List record for the task. The binding records equate the formal parameter name to the IPC of the data item bound to it.

The information derived by the Request Processor about the job is written as a record of the request file to segregate one request from another. In a multiprogrammed environment, many independent requests may be active at once. Each active request is assigned a different record number in the request file. This number is used internally to identify the request for its duration.

#### 5.4.2 Job Extension

A job extension is a request by an operating program for the execution of a job from the library. The program may execute the job as a subroutine, with control being returned to the calling program at the completion of the extended job, or it may request the job extension at its termination.

The same parameter binding powers may be exercised by a program in a job-extension request as by a user in a job-run request issued at a console. The calling program binds data-pool items to the job's input and output parameters by using any

convenient item identifiers: data-pool names, formal parameters names, temporary item names, or IPC's. The job-extension request may specify structural changes in the items and conditional selections.

The Request Processor responds to job-extension requests just as it responds to job-run requests that are initiated at the console. After the initial response by a resident portion of the Job Manager, a scan of the job-extension request is performed by a routine of the Request Processor to produce a decoded run request that is identical to that produced for requests issued from the console. The parameter binding and Task List generation steps of the Request Processor operate on the decoded run request and produce a request record containing the Task List for the extension job. This job is then executed normally. When it is finished, the program that called it receives control at the point after the call, unless the job extension was called at the program's termination.

#### 5.4.3 The Request Record

For each job-run request and job-extension request, the Request Processor develops a request record containing the information needed by the system during the job's execution. The key components of a request record are shown in Table 5-1.

TABLE 5-1. STRUCTURE OF THE REQUEST FILE

```
REQUEST FILE, F
  PARENT TASK, H, V
  DUMP DATA, *, S
    DUMP ID, O, 4
    RETURN ADDRESS, I, 12
  TASK LIST, F
    TYPE, B, 3
    TYPE ID, O, 4
    FLOATS, I, 3
    INPUTS, F
      FORMAL NAME, A, V
      TYPE, B, 3
      IPC, H, V
    OUTPUTS, F
      FORMAL NAME, A, V
      TYPE, B, 3
      IPC, H, V
```

The request record for an extension job contains the identification of the task which called it in the field PARENT TASK. The value of this field is two record numbers, the number of the request record for the job and the number of the Task List record for the task which issued the job extension. The DUMP DATA statement specifies the location in auxiliary storage and the return address of the shelved task if it was rolled-out to accommodate the job extension. For console-initiated requests, the request record contains a null DUMP DATA statement.

The Task List contains one record for each task in the requested job. The fields in the Task List record identify the object code for a program and specify its location in auxiliary storage. They are used by the Job Manager to load and initiate the task. The files for inputs and outputs contain equations relating the formal names of the program's parameters to the IPC's of the data-pool items bound to those parameters. They are used by the Service Package routines to interpret input-output requests issued by the program.

#### 5.5 JOB MANAGER

The Job Manager is a set of system operations which coordinate the flow of control among programs. It consists of four elements which manage the transition of control among tasks, the operating system, and the Request Processor. The set of transitions accomplished through elements of the Job Manager is shown in Figure 5-8.

- (1) When the user requests a DM-1 job, the operating system loads the Request Bootstrap of the Job Manager. This element accomplishes the first transition -- between the operating system and the Request Processor. The Request Bootstrap program assigns a request record number for the request, prepares some parts of the record, and initiates the Request Processor.
- (2) After the Request Processor digests the request and builds the request record, it calls the Task Terminate routine of the Job Manager. This element accomplishes the second transition -- between the Request Processor and the first task of the requested job. It reads the first record of the Task List, determines the location of the object code, and loads the task and executes it.
- (3) Each time a task of the job completes its operation, it calls the Task Terminate routine. In this case, the Task Terminate routine accomplishes the third transition -- between a task and the next task in the job. It reads the next record of the Task List to locate and load the next task.

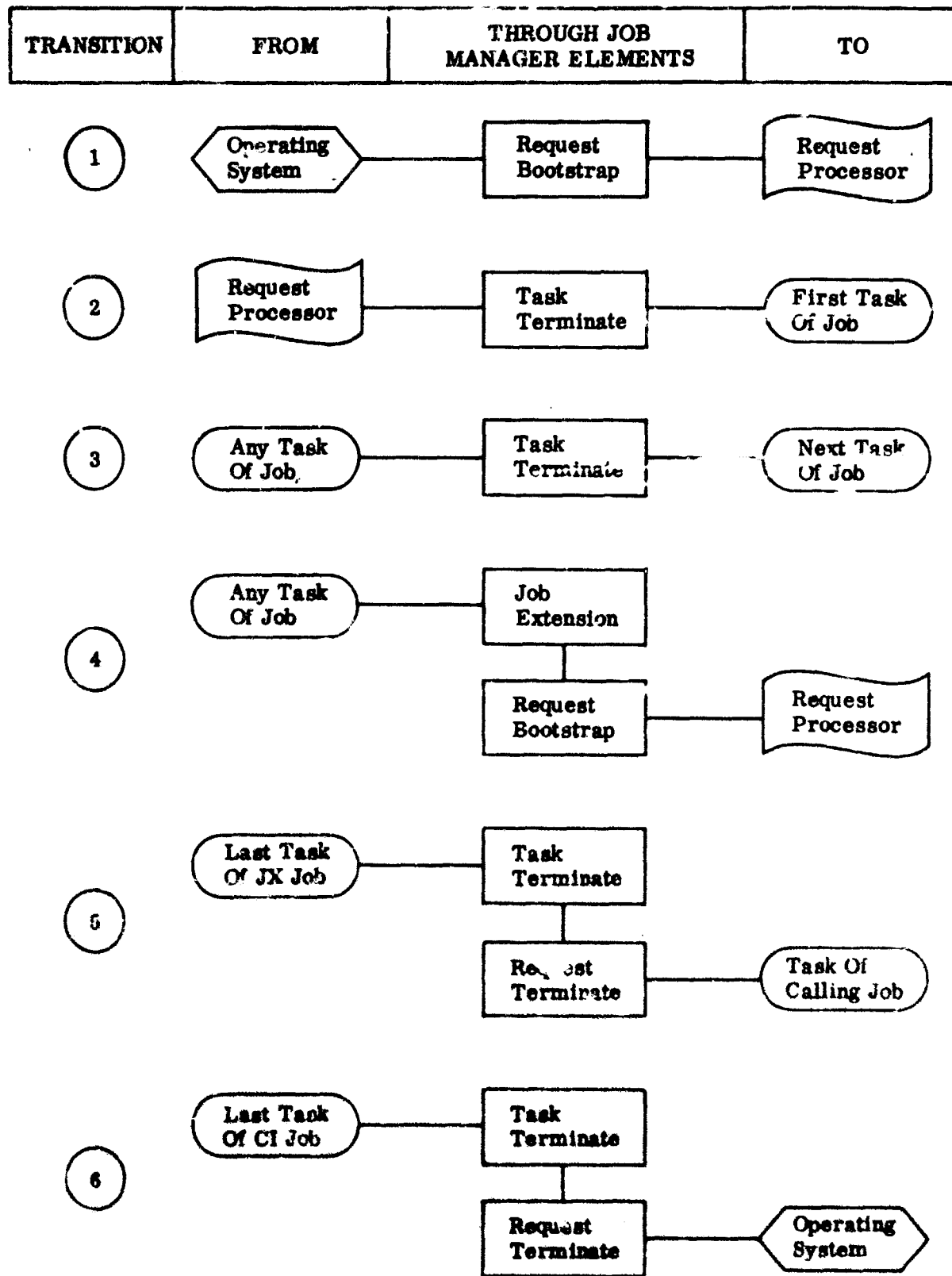


Figure 5-8. Transitions Controlled by the Job Manager



- (4) Any task in a job may execute a job in the DM-1 library as a subroutine or at its termination. The task calls the Job Extension routine of the Job Manager. This element housekeeps for a job extension, rolls out the requesting task, if necessary, and passes control to the Request Bootstrap, which prepares a request record. It assigns a request record number, records the identity of the calling task in the request record for the extension job, and initiates the Request Processor. The Job Extension Routine and the Request Bootstrap function together to accomplish the fourth transition — between any task which issues a job extension and the Request Processor which prepares the extension job for execution. Transition from the Request Processor to the first task of the extension job and the task-to-task transitions within the extension job are accomplished by the Task Terminate routine as shown for the second and third transitions in Figure 5-8.
- (5) When the last task of an extension job terminates, the Job Manager returns control to the appropriate task of the job which requested the job extension. The Task Terminate routine functions as always, by reading the next record of the Task List and executing the task identified by that record. The last record of every Task List produced by the Request Processor calls for the execution of the Request Terminate task of the Job Manager. This element accomplishes the fifth transition of Figure 5-8 — between a completed job extension and the appropriate task of the job which requested the job extension. The Request Terminate task reads the request record for the terminating job and deletes the request record and any scratch data developed by the job. If the task which requested the job extension was rolled out, its location is given in the DUMP DATA statement in the request record. The Request Terminate task rolls the shelved task back into memory and returns control to the return address of the task. Otherwise, it calls the Task Terminate routine with parameters set to return control to the task following the task which issued the job extension.
- (6) When the last task of a console-initiated job terminates, the Job Manager returns control to the operating system. The Task Terminate routine loads and executes the Request Terminate task. This element accomplishes the sixth transition — between the last task of a console-initiated job and the operating system. The Request Terminate task deletes the request record and any scratch data developed by the job. Since there is no reference in the request record to a parent task, control is returned to the operating system.

## 5.6 SERVICE PACKAGE

One other operational feature of DM-1 is the set of routines which perform input-output services for programs running as parts of jobs from the DM-1 Library. These and a resident interpreter constitute the Service Package. The Service Package interpreter occupies a reserved area of memory with some group of service routines whenever any DM-1 job is in progress. The same copy of the Service Package serves any number of DM-1 jobs in the time-shared environment.

The operating system ensures that the Service Package is in memory when a DM-1 job is requested. Any DM-1 system element or task program can call on the service routines to retrieve or store data in the data pool. A call to a Service Package routine is issued through the operating system. The program identifies the desired routine by a code selected from a block of codes assigned to DM-1 by the operating system. When the operating system is entered with any code from the DM-1 block, it passes control to the Service Package Interpreter, which determines the routine requested. If the routine is currently in memory, the Interpreter passes control to it. Otherwise, the requested routine is loaded from auxiliary storage, replacing an inactive routine of the Service Package.

To provide service to many programs with one copy of the Service Package, the services are performed by reentrant routines. A given routine can begin serving other programs while it is still working on earlier requests. For example, if a routine is interrupted while working for program A, the operating system's switching algorithm may give control to program B. Program B may call for the same service routine and the same copy of the routine will be executed. If another interrupt occurs, the routine may be restarted in its work for program A. Conflicts are avoided in serving several programs by keeping all parameters of the service routines in the memory of the programs requesting service. The service routines never modify their own area of memory. All modifications are made in the memory of the caller through index registers, which are saved and restored by the operating system with each interruption of the Service Package.

The input-output services performed by the Service Package routines transmit data between a program's buffer and the segmented data string of the data pool. The programs need not be concerned with the field structure of the data in the data stream.

They specify the format they want in their buffer and the service routines put the data into that format. The program must be aware of the hierarchical relationships among the items it reads and writes. However, discrepancies between the program's assumptions and the actual structure in the data pool can be accommodated by reformat specification in binding data-pool items to the program's parameters.

The programmer's view of the Service Package and a description of the input-output services provided are given in Section VIII.

## SECTION VI. DATA-POOL MANAGEMENT

The transition from magnetic-tape-oriented data processing installations to random-access mass-storage systems has already created the need for an administrator who can arbitrate among users with conflicting interests concerning the best use of on-line mass storage. Multiprogramming and time sharing have added data security and user priority to the administrator's scope of responsibility. The control of file names and program names across separate groups of users is not new. Operating systems have provided a wide range of services to aid the administrator who desires to make the most effective use of a computer center.

Data Management systems add a whole new dimension to the administrator's possibilities for greater efficiency. If the traditional separation of users, programmers, and operations is maintained, a data management system will just increase overhead costs. If the data pool is merely a collection of files and programs designed by autonomous groups, the potential of the new system will not be realized.

The Data Administrator's role can best be described by categorizing several time periods in the life of a project.

- (1) Data collecting, data structure planning, and program/job preparation.
- (2) Preparation for running the job on the computer.
- (3) Measuring performance and making modifications to improve efficiency.

This section will describe the administrator's functions through these various time periods with the emphasis on data. Section VII will cover programs and jobs from preparation, through library storage, and on to execution.

#### 6.1 DATA INDEPENDENCE

The biggest challenge offered by DM-1 is the opportunity to remove data structuring from the narrow purview of the departmental analyst whose only concern has been the efficiency of his application. If the data is susceptible to multiple usage, the data administrator can structure it without affecting departmental applications. Programs can be designed with optimum data structures assumed, and if the data in the data pool is arranged differently, the data management system can structure it automatically when the user job is run.

A series of Data Pool Maintenance Jobs is available to the Data Administrator for data structuring and the establishment of the data base.

#### 6.2 DIRECTORY MANIPULATION

The system jobs concerned with directory manipulation provide the Data Administrator with a data-description language for specifying and altering the logical structure of the data base. The physical handling of the data is separated from the definition of its logical structure. In fact, data cannot be added until the system directories contain the data description.

Paragraph 4.1 describes the various item types (files, statements, etc.) and how they can be arranged into logical structures. When an initial structure has been decided upon, this definition is submitted to DM-1 as input to the Define-Item job. As a result of the execution of this job, the term names for the new items are inserted into the Term Encoding Table and Term List, and the item types and sizes are stored in their proper logical position in the Item List.

Two specific system control functions should be considered at this time. Security restrictions on reading or writing classes of data can be enforced most efficiently if items are properly classified as they are defined. For example, if an employee rating file is never to be read except by the Personnel Manager, the file and its sub-items should be given a Security Restriction Level (SRL) code high enough to automatically prevent most users from seeing the data.

Problems caused by unintentionally duplicating item names will not arise if new item definitions are checked against a current Term Encoding Table list before the Define-Item job is run.

Indexing of fields should be under the Data Administrator's control. When a field is indexed, random retrieval is quicker, but mass-storage space is reduced because additional directories are required to provide the rapid access.

After the initial choices are made, the Index job is executed and the additional directory tables are constructed. If space becomes a problem, the Remove-Index job can be used to undo the indexing and release the space. There are options within the Index job itself which can be used. Indexing all field values (mode ALL) is most expensive. Perhaps only partial indexing through the LIST or RANGE options will provide a better balance between space and time.

Logical relationships between items in separate structures are defined through the Link job. Through linkage, it is possible to define several logical structures without the need to duplicate physical data. Since the Data Administrator will see all item definitions, he has the opportunity to consider:

- (1) Separating the structures so that the background or reference data will be available, when called for, but will not encumber the high activity data.
- (2) Duplicating the data, with its cost in space and maintenance versus nonduplication with its slower retrieval.

The initial choices are not critical because links can be removed by the Delete-Link job followed by a Define-Item job to add the item to the source structure.

Structures from which logically related data is removed get a source link defined through the Link job. The source link points to the target structure where the logically related data is stored. In the target structure a target link is inserted to identify the items which are logically related to the source items. These links are unidirectional; in proceeding down through a structure, a source link can cause branching to a new structure, but a target link is ignored.

### 6.3 DATA MANIPULATION

After an initial data structure has been defined to the system, data may be added to the data base. The Add-Data job will accept a small volume of data directly from the console. Large volumes of external (foreign) data are translated through an External-to-Internal Conversion job before Add-Data is executed.

Since most data is characterized by a rapid rate of change, there is a series of maintenance jobs to accomplish data deletion, data replacements, and selective explicitly stated modifications. The query capability and the random-access mass storage suggest that data changes should be input as they are received. The Replace-Data and Modify-Data jobs are designed for on-line maintenance. For the more traditional batch processing, the Update-Data job will apply a whole series of transactions to a file in one job run.

For files containing indexed fields, the Data-Delete job removes records by setting each of the subitems to a null state. This allows records to retain their original record numbers and thus the effect on the index tables in the directory is minimized. After a file has had a large number of such deletions, the File-Compress job can be used to adjust all the record numbers at one time and re-establish a contiguous set.

There is no single system job which will rearrange data to fit a change in the item definition. When the Data Administrator decides to redefine a structure, he first executes the R-format job. Reformat will restructure the data in accordance with the specifications given, and temporarily file the data in the working area of the data pool. Then the structure is redefined using the proper directory manipulation jobs. These jobs automatically delete data-base data when structural changes are about to be made. After he is satisfied with the definition, the Data Administrator uses the Add-Data job to return the restructured data to the data base.

If this combination of system jobs becomes a common occurrence, the combination can be entered as a single job through the use of the Job Description job (see Section VII).

#### 6.4 PREPARATION FOR USERS

All of the directory and data manipulation described in the preceding paragraphs should be considered as preparatory. The purpose is to make the data available for use. Before a user can run a job to interrogate the data, or read it for the purpose of performing computations, the user must be identified to the data management system.

The Add-User job has as its input user name, priority, and clearance levels. A Clearance Level (CL) is a code related to The Security Restriction Level (SRL) which has been assigned to the data class (see Paragraph 6.2). If the user's CL code is higher than the data's SRL code, the user may have free access to the data. The data administrator protects large portions of the data pool from unauthorized access or modification by careful assignment of users' CL codes.

When the relationship between data SRL codes and user CL codes is not precise enough, there is a second level of protection which can be used. Using the employee rating file as an example, the data may have been given a SRL code of 5, and the Personnel Manager a CL code of 6. Now, the manager of accounting may have been given a CL code of 6 so that he can gain access to certain data which he needs in order to perform his job. This CL code of 6 would allow the accounting man to see the employee rating file if no other protection scheme was available.

The Data Administrator can reduce the Clearance Level for both of these men to code 5 through the Delete-User job followed by the Add-User job. This makes the employee rating file and the accounting file unavailable to either manager. Then through the Add-Access-Rights job, the proper data is made available to the proper man by explicitly stating, in the Access Rights table of the directory, which data each man may read.

The writing or modification of data is controlled through parallel procedures to those described for access.



## 6.5 USAGE STATISTICS

After the data has been defined, added to the data base, and used for a period of time, the Data Administrator can begin to measure the performance of the system. DM-1 accumulates data-usage statistics in three separate areas to assist the Data Administrator in planning modifications to improve efficiency.

In the Segment Name List, a count is recorded to show how many times each segment of the data pool has been accessed. When hierarchies of secondary storage are provided, this count will guide the operating system in determining what level of physical storage should be used. Data base segments with counts above or below average will indicate where structural changes should be made to make active data more easily retrieved, and to delete inactive data as mass storage becomes full.

Since the directory segments are also identified in the Segment Name List, the ratios between directory and data accesses will provide interesting statistics for use in upgrading the system after usage patterns have been established.

The other two areas where tallies are maintained are special purpose; they measure the usage of index and linkage tables. Based on the rates of usage, the Data Administrator can modify his original decisions on what fields to index and what data to link.

There is a record in the FIELDS file for each indexed field. When queries or conditional searches are executed, this record is accessed en route to the FVT file. The tally in this record will indicate whether or not a field deserves to be indexed. Embedded in the same record is the FVT file which lists all of the field values by which the field is currently indexed. For each of these explicit values, there is a second tally. These minor tallies will suggest changes in the indexing option (ALL, LIST, or RANGE) which may reduce index table size without reducing access speed.

There is a record in the Linkage Table for each source- and target-link item. In the records for source links, there is a usage count which will show how often users, in reading through source data, have retrieved the target data via the source-to-target link. A very large count might suggest duplication of data in the source structure.

A very small count might indicate that the data relationship established by the link is an artificial one and that the link could be removed without loss.

Initially, the system will produce these usage statistics and depend on the Data Administrator to analyze them and make any suggested modifications by executing the appropriate Data-Pool Maintenance job. Ultimately, the system may be instructed to perform the analysis and modify itself automatically.

The current system design does not include usage statistics in the User Access/Modification Rights tables or in the Program/Job Library tables. After the installation grows to include many users and many programs, it might be worthwhile to add usage counts to these elements of the directory.

## SECTION VII. PROGRAM PARAMETERS AND THE JOB LIBRARY

Paragraph 5.2 of this report describes parameter binding as an operational feature of DM-1. Some of the concepts presented will be repeated here, in greater detail.

### 7.1 PARAMETER CATEGORIES

A major responsibility of the DM-1 system is to associate the input-output parameters of programs with actual data items in the data pool. This association is called parameter binding. It is accomplished, in the DM-1 system, in three distinct phases associated with three conventions of the system:

- (1) How a program accesses and stores data in the data pool.
- (2) How the individual inputs and outputs of programs are interrelated in a job.
- (3) How the inputs and outputs of a job are tied to data-pool items in a job-run request.

These conventions present the input-output parameters in different aspects, according to the context in which a parameter is viewed. The character of the input-output parameters changes as they are associated with programming, job description, or job execution.

Figure 7-1 is a branching structure which portrays the categories of input-output parameters considered in DM-1. The generic term input-output (parameter) applies to all categories and is shown in the figure as the highest node in the structure. There are two types of input-output parameters. Generalized parameters are idealized items that are associated with programs and jobs and for which no data exists.

Specific parameters are real items which correspond to logical positions in the data pool which can contain the data associated with the parameter.

The binding process ultimately results in the assignment of specific parameters to generalized parameters. The time axis in Figure 7-1 demonstrates this. At program time, all parameters are generalized; at execution time, all parameters are specific; in between, at job time, some parameters may be generalized and some may be specific. The progression from generalized to specific may take place in several stages:

- (1) When a new program is entered into the DM-1 library, its input-output parameters are defined to the system as generalized.
- (2) If the program is used as a component in a job, some of its input-output parameters may be bound to specific items. Others may remain generalized.
- (3) When a job is called for execution, any remaining generalized parameters must be made specific by the assignment of a specific node in the data pool to each.

#### 7.1.1 Generalized Parameters

There are two categories of generalized parameters, formal and individual. Formal parameters are input-output items associated with a program. Individual parameters are input-output items associated with a job. When a program is used as a component in a job, its formal parameters are bound to either specific parameters

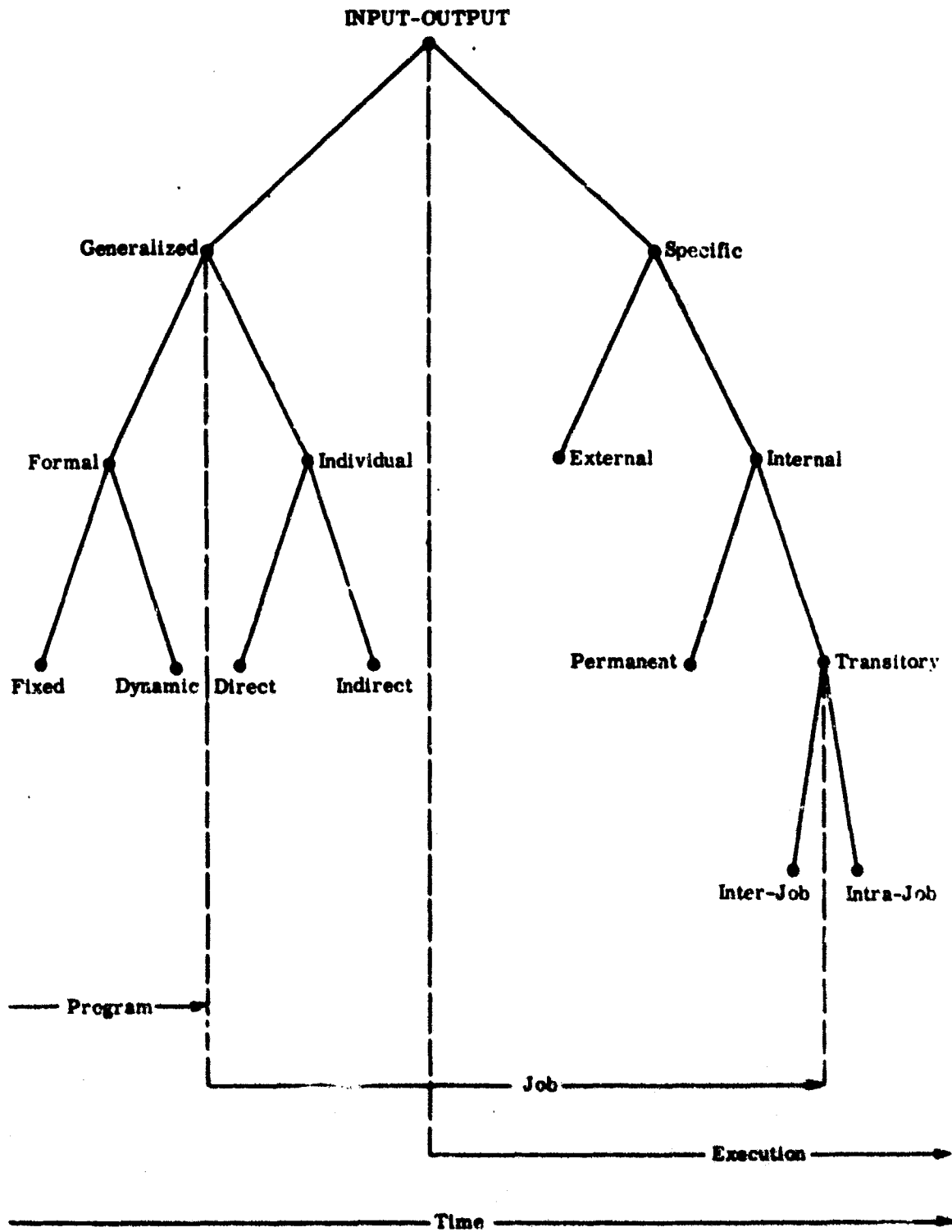


Figure 7-1. Categories of Input-Output Parameters

or individual parameters. That is, they are assigned either to actual data items in the data pool or to job parameters which will be made specific when the job is executed.

Program parameters (formal) may be fixed or dynamic. Fixed parameters have a fixed structure defined by the programmer. The program assumes that the data bound to a fixed parameter is formatted according to the specified structure. Dynamic parameters have no specified structure. The program is written to operate with the DM-1 directory information to determine the structure of dynamic input parameters or to provide an item definition to the system for dynamic output parameters.

Job parameters (individual) may be direct or indirect. Direct parameters are intermediate items in the list of component programs that constitute the job. That is, a direct parameter is an output of a program which is an input to other programs in the job, but not an output of the job. When the job is executed, the system must make all direct parameters specific by assigning a logical position in the data pool to them. This specific node is assigned in the scratch area of the data pool. Indirect parameters are parameters which are inputs or outputs of the job itself. A job output is an output of one of the component programs. Such an output may be an input to any of the other component programs. A job input is an input of one of the component programs and may also be an input to others. Indirect parameters are made specific by the assignment of actual data-pool items or literals to input items and actual data-pool items or working names for output items. These assignments are made by a user who issues a job-run request at a console or by programs which issue a job-extension request.

The types of generalized parameters, then, are formal parameters for program inputs or outputs and individual parameters for job inputs, outputs, and intermediates. Formal parameters are fixed when their structure is assigned by the programmer, or they are dynamic when the program determines the structure during execution. Individual parameters are indirect if they are inputs or outputs of the job, or they are direct if they are inputs and outputs of job components, but not of the job.

#### 7.1.2 Specific Parameters

Specific parameters are actually logical nodes in the data-pool structure. The categories of specific parameters are derived from the agency which assigns them and the nature of the parameter.

Specific parameters may be internal or external. An internal parameter is an item in the data pool. An external parameter is a literal to be used as the value for a generalized parameter when a job is executed. The literal may be bound to a generalized input parameter of a component program when a job is described, or to a job input when a job is executed. The system makes the external parameter specific by assigning a node in the scratch area to it and by mapping the literal into the item corresponding to that node.

Internal parameters may be permanent or transitory. A permanent parameter is an item in the common data base. A transitory parameter is an item in the work area or scratch area. Permanent items exist in the data base independently of any particular job or user. They may be assigned to generalized parameters when a job is described or when a job is executed.

Transitory parameters are items dedicated to particular jobs or users. An inter-job parameter is a transitory parameter which exists in the work area of the data pool. A user may define work-area items as private files and use them as specific input-output parameters to bind the generalized parameters of jobs. He may also assign a job output to the work area by naming the item with a flag to request that the system make the appropriate item definition. Either way, the inter-job parameter persists after the execution of the job which created it and may be bound to the parameters of other jobs. An intra-job parameter is a transitory parameter which exists in the scratch area of the data pool. It exists only for the duration of the job which uses it. Intra-job parameters are always assigned by the system. They are used whenever a generalized parameter must be made specific, without the specification of a data-pool item by a calling agent. All direct parameters are made into intra-job parameters by the system when a job is executed. This action assigns a scratch-area node to accept the output of one component of a job so that it may be used as an input to other components.

Likewise, literals are transformed to intra-job parameters by the system. They are written into a scratch item in preparing for a job execution so that they can be read by the appropriate job components during execution. Also, any job outputs which the calling agent fails to bind are converted by the system to intra-job parameters so that they will be written into the scratch area during the execution of the

program which creates them. When the job is completed, all intra-job parameters are automatically discarded.

### 7.1.3 Parameter-Binding Examples

Examples will be used to illustrate the various ways in which parameters may be bound.

Consider a program called REGRESSION which fits a line to a set of points. Each point is defined by an X-value and a Y-value. The input to REGRESSION is a file called POINTS which contains a record for each point. The output is a statement called LINE which contains two fields, M and B, defining a line. This is illustrated in Figure 7-2.

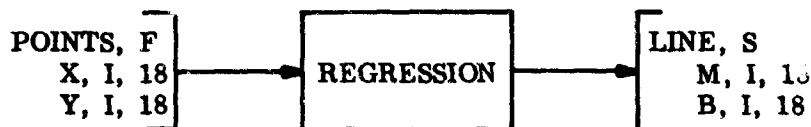


Figure 7-2. Example of Parameter Binding

During the design of REGRESSION, a programmer designed the generalized input parameter POINTS and the generalized output parameter LINE to meet the requirements of the process. These are formal parameters because they are parameters of a program. They are fixed because the structure of each is predefined by the programmer. Thus, the parameters POINTS and LINE are generalized, formal, fixed parameters.

During the implementation of REGRESSION, the programmer assigns the structures he has designed. He reads his input file by referring to the name POINTS and writes his output statement by referring to the name LINE. He knows that when the program is actually running it will operate on some file of points which was assigned to his formal parameter POINTS by a user. It is not significant to him that the specific parameter has some other name. He is indifferent to the other characteristics of the file: whether it is a data-base item, a work-area item or a scratch-area item; whether it was produced by some previous job, by a program which preceded



REGRESSION in the same job, or by a maintenance operation which converted it from some external medium; whether the specific item has the X before the Y or the Y before the X; etc. The programmer knows that the DM-1 system will arrange things so that when he reads a record of the file which he has called POINTS, his buffer will be set to the next X-value (18-bit integer) followed by the next Y-value (18-bit integer), regardless of the characteristics of the specific item being processed by the program. If these results cannot be achieved, the system will not permit the program to operate.

After REGRESSION is compiled, the programmer, or some other user, enters the program into the DM-1 library. This gives the system knowledge of the formal parameters POINTS and LINE. Entry into the library automatically makes the program into a one-task job. This converts the parameters into individual parameters; i. e., the input POINTS and the output LINE of the job REGRESSION are individual parameters because they are job items. They are indirect parameters, because they are job input-output parameters which may be bound to specific parameters when a request to execute the job is issued.

The indirect parameter (input) POINTS may be made specific by assigning one of the following specific parameters to it:

- (1) An external parameter. A literal value may be given for the file as part of a job-run request. The system assigns a node in the scratch area and maps the literal into the item to prepare for the execution of REGRESSION. The scratch-area item is then an intra-job parameter which will be discarded after the job is executed.
- (2) An internal, permanent parameter. The name of an item in the data base may be used to make POINTS specific. If the file is embedded in higher level files, a condition may be used in the binding specification to define the precise file to be used. If the format of the selected item does not conform to the structure of POINTS, a reformat clause may be used to direct the system to select the appropriate fields for the X and Y and to use them as 18-bit integers as directed by the item definition for POINTS.
- (3) An internal, transitory, inter-job parameter. This binding specification is made exactly like the preceding one. The only difference is that the item is selected from the work area instead of the data base.

When the program operates on its formal input POINTS, it is completely indifferent to the mechanism that made it specific.

The indirect parameter (output) LINE may be made specific by assigning one of the following specific parameters to it:

- (1) An internal, permanent parameter. An output item may be bound to a data base item under the same rules as those for an input item. However, no output parameter may be bound to a permanent item if it would cause changes in the values of an indexed field.
- (2) An internal, transitory, inter-job parameter. This kind of assignment may be made in two ways. If the work-area item to be used as the specific output item already exists, the binding is the same as that for a permanent parameter. If the item does not exist, the user may assign a name and instruct the system to create a work-area item to accept the output. For example, the user might assign the name MYLINE to the indirect parameter LINE. After the job is finished, a work-area item called MYLINE will exist with the values for the M and B created by REGRESSION during the run. The item MYLINE may be displayed to the user or used as an input to some other job. It will remain in the work area until the user deletes it.

The job REGRESSION may be used as a component in some other job. Suppose that a job called FAILURE ANALYSIS is constructed from the jobs SELECT, REGRESSION and PLOT. The situation is depicted in Figure 7-3.

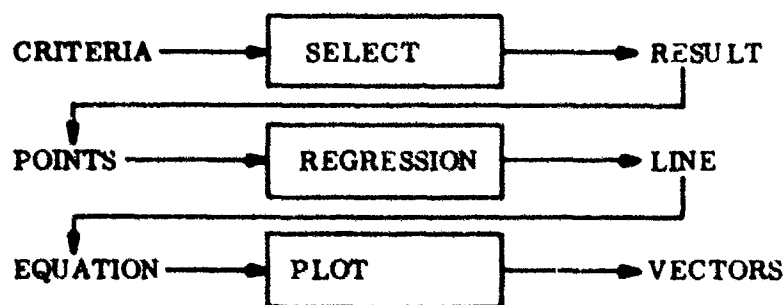


Figure 7-3. Job Construction Procedure

The job FAILURE ANALYSIS includes the job REGRESSION as a component. If the parameters of REGRESSION are neither inputs nor outputs of the new job, they become direct parameters in this context. That is, they are intermediate items in the job FAILURE ANALYSIS, which might be viewed as a job with one input and one output as shown in Figure 7-4.



Figure 7-4. Intermediate Form of the Job FAILURE ANALYSIS

The job has the indirect input parameter called CHOICES which is bound to the indirect parameter CRITERIA of the component SELECT. It has the indirect output parameter FAILURE LINE which is bound to the indirect output parameter VECTORS of the component PLOT. It also has two direct parameters: one results from binding the parameter RESULT of SELECT to the parameter POINTS of REGRESSION, and the other results from binding the parameter LINE of REGRESSION to the parameter EQUATION of PLOT. Whenever FAILURE ANALYSIS is executed, the system will assign an intra-job parameter in the scratch area to these direct parameters of the job. The indirect parameters will be bound to specific items by the same mechanisms as were used for the job REGRESSION.

## 7.2 PROGRAM ENTRY

A program comes under the control of the DM-1 system when it is entered into the library through the Program Entry job. Programs are compiled independently of the system and their object code is stored under the control of the operating system. They are entered into the DM-1 system through a program specification which includes the following elements:

- (1) Program Name
- (2) Program Input (formal)
- (3) Program Outputs (formal)
- (4) Program Executive Control Description

The formal input-output parameters are described by naming them and giving an item definition for the fixed parameters. Once the Program Entry job has processed the program specification, the program becomes a job in the DM-1 library. It may be called for execution by a job-run request issued by a user at a console or by a job-extension request issued by a running program. It may also be used as a component in a job description.

### 7.3 JOB DESCRIPTION

A job description defines a new job as a sequence of existing jobs. The components in the sequence are jobs from the DM-1 library. They are in the library because they were defined by a previous job description which was processed by the Job Description job or they were entered by a program specification which was processed by the Program Entry job.

To describe a job, the user names the job and its input-output parameters, identifies each component job, and binds the indirect input-output parameters of each component job. The job description includes the following elements:

- (1) Job Name
- (2) Job Inputs (indirect)
- (3) Job Outputs (indirect)
- (4) Job Components List

The job name is the name through which the new job will be called for execution. The job inputs and outputs are a series of parameter names for the bindable, indirect, input-output parameters for the new job. The job components list contains the name and binding specification for each job.

The job inputs and outputs are user-assigned names for component inputs and outputs which are not to be made specific by the job description. They are dummy names which are used in the components list to show the relationship between the inputs and outputs of the new job and the inputs and outputs of the component jobs. If the job description makes all component inputs and outputs specific, there are no job inputs or outputs.

The components list contains an entry for each component job in the sequence in which they are to be executed in the new job. It gives the name of the component and binds each of the component's indirect input-output parameters.

### 7.3.1 Component Input Parameter Binding

Each component input parameter is bound by the assignment of one of the following:

- (1) An external parameter. This is a literal value to be used as the value for the component input parameter.
- (2) A permanent parameter. This is the name of an item in the data base. The binding specification may include a condition clause and a reformat clause to direct the user to select a subset of the named item and to interpret the selected subset in a format which differs from its format in the data base. The resulting item is to be used as the input to the component each time the new job is run.
- (3) An inter-job parameter. This is the name of an item in the work area. The binding specification is the same as that for a permanent parameter.
- (4) An indirect parameter. The component input parameter is itself an indirect parameter which must be bound whenever the component is called as a job or used in a job description. The first three parameter types which may be bound to the component input parameter make it specific; i. e. , they specify an item in the data pool as the source of the input data. (A literal is an item in the data pool when the job is executed.) However, the new job might be more flexible if the binding for some of the component inputs can be deferred until the job is executed. This is accomplished by binding the component input parameter to an indirect parameter of the new job. The indirect parameter is assigned a name in the list of job inputs and this name is used to bind the inputs of some of the components. When the new job is executed, its indirect input parameter is made specific by the user. The associated component inputs are made specific at the same time.

- (5) A direct parameter. This is a name used to identify an output of a previous component in the component list. It is not specific because there is no node in the data pool corresponding to it. When the new job is executed, the system will assign a node in the scratch area (an intra-job parameter) to accept the output of the earlier component so that it may be used as the input to later components. The output of the earlier component may also be an output of the job. In this case, the name of the job output is used, and the system uses the node bound to the job output as the source of the input data for components whose inputs were bound this way. A condition clause and a reformat clause may be used to specify a subset of the source item and a change in its structure.

### 7.3.2 Component Output Parameter Binding

Each component output parameter is bound by the assignment of one of the following:

- (1) A permanent parameter. This is the name of an item in the data base and a condition, if necessary. It defines the unique node in the data base which is to receive the output item from the component.
- (2) An inter-job parameter. This is the name of an item in the work area and a condition, if necessary. It defines a unique node in the work area which is to receive the output from the component.
- (3) An indirect parameter. When the binding specification for a component output parameter is to be deferred until the new job is executed, the component output parameter is bound to an output of the new job. The user assigns a name in the job output list and uses this name to bind the component output.
- (4) A direct parameter. This is a name used to specify that the component output is to be used as an input to components which occur later in the components list.

### 7.3.3 Parameter-Binding Choices

The choices for binding the indirect input-output parameters may be summarized as follows:

- (1) Specific data-base and work-area items may be bound to some of the input and output parameters of the components.
- (2) Some of the component inputs are connected to outputs of previous components by assigning a name to the output and using that name for the inputs.
- (3) With either (1) or (2), a condition clause and a re-format clause may be used to specify a subset of the source item and a change in its structure when the source item is used as an input. A condition may be used with (1) to define a unique node if the named item is embedded in a file.
- (4) The specific binding of component input and output parameters may be deferred until the new job is executed by assigning a name in the job input or job output list and associating the component input or output with that name.

### 7.4 THE JOB DESCRIPTION LIBRARY

The Job Description job processes the job description to produce a new entry in the job description library. Such entries are also produced by the Program Entry job when a new program is defined to the system. The new program is treated as a one-component job and is known as a terminal job. These are the fundamental building blocks of all jobs.

Four major elements of information are maintained in the job description library for every job. Two additional elements are maintained for terminal jobs. These are:

For terminal jobs only:

- (1) Input-Output Item Description
- (2) Executive Control Description

For all jobs:

- (1) Job Item List
- (2) Static Task List

(3) Component List

(4) Usage List

The input-output item description contains an item definition for each of the formal parameters of a program. The structural information, which is identical to the information in the Item List and Term List in the system directories, is maintained. For fixed parameters, the item description is the definition specified to the Program Entry job. For dynamic parameters, the item description is a null node to be defined by the system if it is a dynamic input parameter or by the program, during its execution, if it is a dynamic output parameter.

The executive control description contains the identifier for the program's object code within the operating system. It is used by the DM-1 system to request that the program be loaded.

The job item list contains an entry for each item used in the course of a job's execution. There is one entry for each of the job's indirect input-output parameters, one for each data pool item used as specific input-output parameters for a component, one for each literal used as input to a component, and one for each item needed to effect the binding of a component output to the inputs of other components. Each item occurs only once in the job item list, even when it is used as an input to several components. The input-output items are classified in the job item list in one of the following categories:

- (1) Job Input-Output (indirect)
- (2) Intermediate Input-Output (direct)
- (3) Internal Input-Output (internal)
- (4) Literal Input (external)

The job input-output category defines the parameter as an indirect parameter which must be bound in any request for execution of the job. Parameters in this category are the only inputs or outputs of the job; the other categories specify items which are internal to the job. The intermediate input-output category defines a requirement for a scratch item so that one component can write an output item which will be read



by others. The internal input-output category is for data-pool items which were bound to inputs or outputs of components. The literal input category refers to literal values which were bound to component inputs in the job description.

A static task list is another element of the job description library. It is present for all jobs. The static task list contains one entry for each task (terminal job or program) which must be executed when the job is requested. The entries correspond to the fundamental programs which make up the job. If a multicomponent job is used as a component of a higher level job, the system reduces the set of components to the fundamental programs, or tasks, by copying the static task list of the component job into the static task list of the new job. Each entry in the static task list contains the executive control description of the program and a list of the formal parameter names used by the program. Each formal parameter name is equated to an item in the job item list.

The component list is an internal representation of the component list that was used in the job description. It permits the system to reconstruct the job description for display purposes. Each entry in the component list contains the component name and the binding specification that was used for the component in the job description. Each input and output is coded to specify its category in the job item list.

The usage list contains the names of all jobs which use the job described by this library entry as a component. This list is displayed when a job description is displayed, so that the user can determine the jobs which would be affected by a change in a given job.

#### 7.5 JOB AND PROGRAM DELETION

Jobs and, therefore, programs may be deleted from the job description library by the Job Deletion job. The deletion may be accompanied by a display of the job description so that the user may scan the usage list to determine which higher level jobs are affected by the deletion. The display may not be required because the deletion is frequently made to accommodate the entry of an updated version of the job. If an updated version involves no changes in the job's indirect input-output parameters, the change can have no effect on higher level jobs which use the changed job as a component.

7.6

LIBRARY DISPLAY

The job description of any job in the library may be displayed through use of the Display Job Description job. The entire description or a part of it may be displayed. This gives the user the means of uncovering the descriptive information he might need to execute the job or to use it as a component in another job.

## SECTION VIII. PROGRAMMING SYSTEM SERVICES

The DM-1 service routines offer the user programmer a comprehensive set of services for storing and accessing data. The service routines exist to permit the user to manipulate the data pool randomly and conveniently without paying any attention to the mechanics of packing, segmenting, indexing, etc. The program receives its data in the form of traditional, word-oriented fields. The mechanics and control are supplied by the service routine. The programmer need only be aware of the logical structure of the data which his program processes.

Figure 8-1 shows the DM-1 Service Package as the interface between the data pool on mass storage and the program's buffer within the program. By calls for service routines, the programmer directs the Service Package to transmit data from the data pool to his buffer, or from his buffer to the data pool. The parameters and storage space required in interpreting the data segments in terms of the structure definitions are controlled by the service routines.

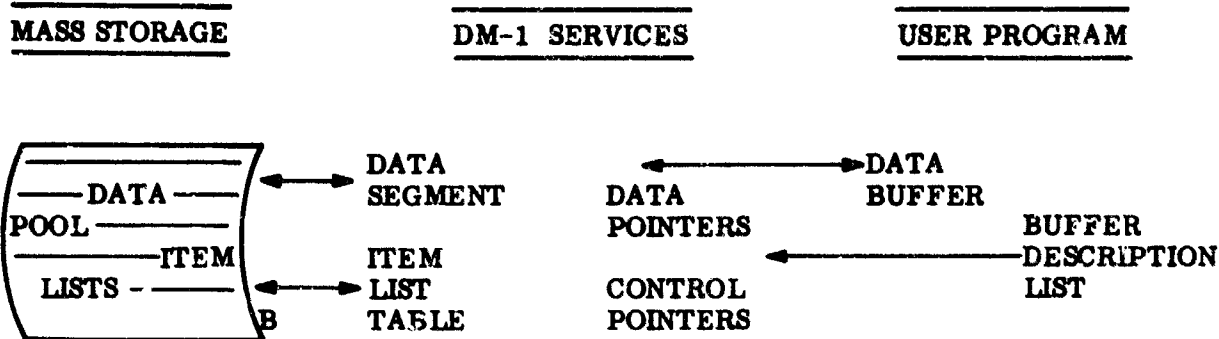


Figure 8-1. Interfaces of the DM-1 Services

## 8.1 DATA ACCESS SERVICES

The user may employ a variety of techniques for integrating service calls into his flow of processing. However, all of the techniques essentially consist of an Open operation, followed by a mixture of Seek's and Read's and terminated by a Close.

### 8.1.1 Open-For-Input

The programmer employs the Open-For-Input service to identify an area of the data pool from which he intends to read some data. The Open operation prepares the DM-1 system controls so that the system is able to respond to subsequent input requests for any item subsumed within the opened item.

In the Open request, the user supplies a symbolic name for the item to be opened. This symbolic name can be either the actual term name of a data-pool item or a formal name that was bound to a term name by the Request Processor (see Section V). The Open service retrieves the structure definitions (Item List) for the named item and its subsumed items. Open sets a control pointer, "j", at the first subsumed item. In the example shown in Table 8-1, Purchasing Orders, FILE, "j" would be set to 2, the Order Record.

Open also retrieves the segment containing the first data that is pertinent to the opened item. Then it steps over any irrelevant data, leaving the data pointers at the first data bit of the opened item. This physical retrieval of segments has no impact

TABLE 8-1. STRUCTURE DEFINITION FOR PURCHASING ORDERS FILE

| RELATIVE<br>ITEM NO. | ITEM NAME, TYPE, SIZE          |
|----------------------|--------------------------------|
| 1                    | Purchasing Orders, FILE        |
| 2                    | (Order, RECORD, 7)             |
| 3                    | PO Number, alphanumeric, 6     |
| 4                    | Due Date, decimal, 6           |
| 5                    | Requestor, alphanumeric, V     |
| 6                    | Vendor Number, decimal, 5      |
| 7                    | Vendor Label, statement, 4     |
| 8                    | Name, alphanumeric, V          |
| 9                    | Street, alphanumeric, V        |
| 10                   | City, alphanumeric, V          |
| 11                   | State, alphanumeric, V         |
| 12                   | Value, integer, V              |
| 13                   | Purchasing Items, FILE         |
| 14                   | (Item, RECORD, 3)              |
| 15                   | Item Sequence Code, integer, 8 |
| 16                   | Quantity, integer, 8           |
| 17                   | Cost, integer, V               |

on the logic of the user program. The program is affected only when data is moved into the user buffer area. The completion of the Open operation allows the programmer to issue other input requests, namely, Read's and Seek's.

#### 8.1.2 Read

The programmer employs the Read service to move selected data items into his buffer. This service operates under the control of a list, the Buffer Description List (BDL), which is supplied by the user with the Read call. The programmer uses this list to specify which items should be moved to the buffer (TRANSMITTED) and which items should be passed over (SKIPPED). The BDL describes the buffer fields, assigned by the programmer to accept the fields transmitted in the Read, by specifying the size of each buffer field.

The programmer considers first the data item at which the system pointers are currently set. The action code (Transmit or Skip) for this item is contained in the first entry of the list. The second entry defines the action for the second item, and so on. If the user calls for an item to be skipped, all items subsumed by this item are ignored. Consider the statement Vendor Label, which is Item No. 7 in Figure 8-2. If a BDL called for skipping this statement, the four subsumed items would be passed over. The next action code in the BDL would affect Item No. 12, Value.

However, if a programmer calls for an item to be transmitted, the entire item is not automatically transmitted. The programmer can select just those sub-items which he wants. He does this by including an entry in the BDL for each subitem. Thus, if a programmer called for transmitting Vendor Label, this could be followed by "Transmit, Skip, Skip, Transmit." This would cause the DM-1 Services to move Name into the buffer, followed by State. Since the programmer does not explicitly name the fields in the list, it is important for him to be mindful of the position of the system pointers.

Each entry in the BDL contains space for buffer field size and status indicators, in addition to the Skip/Transmit action code. The user supplies the size of the buffer field for every field which is to be transmitted. This enables the DM-1 system to justify each piece of data in a buffer field of convenient size for the user's processing. In the status indicators, the system supplies information such as:

end of file, optional item missing, and item-size error. Figure 8-2 shows the Buffer Description List for a program that needs the fields Name and State from the Vendor Label statement. The list assumes that the input pointer is at Item No. 7, Vendor Label.

| <u>ITEM</u>        | <u>BDL</u>    |             |                  | <u>NOTES</u>   |
|--------------------|---------------|-------------|------------------|--|
|                    | <u>ACTION</u> | <u>SIZE</u> | <u>INDICATOR</u> |  |
| 7. VENDOR LABEL, S | T             | -           |                  | T = Transmit<br>S = Strip<br>Size gives number of<br>units; the unit is<br>identified in item type<br>of Item List |
| 8. NAME, A         | T             | 24          |                  |  |
| 9. STREET, A       | S             | -           |                  |  |
| 10. CITY, A        | S             | -           |                  |  |
| 11. STATE, A       | T             | 16          |                  |  |

Figure 8-2. Sample Buffer Description List

When the Read operation is completed, the system pointers remain set at the item which follows the last item treated in the Buffer Description List.

#### 8.1.3 Seek

The programmer employs the Seek service to access data in a completely random manner within the opened item. In a Seek call, the programmer identifies a desired data item; the DM-1 service sets the data pointers and the control pointers to that item. The item can be identified by relative item number (see Table 8-1), or by relative item number and record number. In the latter case, the programmer may replace the record number with an end-of-file indicator or with a key value for an ordered file.

Depending on the item desired, Seek can move the pointers forward or backward. If the item is not in the currently available segment, the service routine retrieves the segment containing the first data of the desired item. Seek, like Open, does not move any data into the user buffer.

#### 8.1.4 Close-For-Input

The programmer employs a Close-For-Input call to indicate that he has no further need for the opened item. The Close operation releases the storage areas which the system was using for this item.

### 8.1.5 Retrieve-Item

Retrieve-Item is both the most comprehensive DM-1 access service and the simplest to use. It provides the programmer with a single call which will fetch data from any area of the data pool and deliver it to the user's buffer. Retrieve effectively combines the operations of Open, Read, and Close.

For a Retrieve-Item call, the programmer supplies the symbolic name of the desired item together with a Buffer Description List which controls the editing of the item. The symbolic name can be either the actual term name of a data pool item or a formal name that was bound to a term name by the Request Processor (see Section V). The Buffer Description List is the same as that described for the Read service in this section.

Retrieve is convenient to use inasmuch as it permits the user to get an input item with a single call. Moreover, since Retrieve is a completely random operation, the programmer is relieved of responsibility for knowing the initial position of the system pointers. However, the programmer cannot rely on Retrieve for all data accesses. Since this service delivers the data to the user's buffer, the user cannot call for an item that is larger than his buffer. Even for small items the programmer may choose to use Open with Read's and Seek's. To cite an extreme case, the programmer would not want to Retrieve-Vendor Label (Table 8-1) and then Retrieve-Value. This pair of calls would cause the DM-1 system to repeat unnecessarily several operations, such as retrieving the Item List and building the Item List Table. The use of Open permits any number of accesses of the subsumed items, while the major control operations are performed only once. Each user can apply these considerations to his own program and select the method of accessing which offers him the greater advantage.

## 8.2 WRITING SERVICES

The DM-1 data storage services are divided into two categories: the writing services, which are treated in this paragraph, and the updating services, which are covered in Paragraph 8.3. Writing is strictly an output operation, while updating is a combination of input and output operations.



Writing admits of fewer variations than reading. To begin with, writing proceeds always in a forward direction. In addition, writing does not permit skipping to an item, as in a Seek operation where many items of different levels may be logically passed over. In writing, only the current item may be skipped. Although this could involve the skipping of many subsumed items, it is still a rather straightforward operation.

#### 8.2.1 Open-For-Writing

The programmer employs the Open-For-Writing service to identify an area of the data pool into which he intends to put some data. The structure of the data item being opened must have been defined earlier, either by means of the Define-Item job or by means of the Fix-Item service. The Open-For-Writing service includes all of the operations performed in Open-For-Input: translating the symbolic item name to a system identifier (IPC), retrieving the structure definitions (Item List), and retrieving the data segment in which the opened item logically belongs (based on its IPC). In addition, Open-For-Writing sets up a new output segment and copies from the retrieved data segment any data which has an identifier lower than the IPC of the opened item. At this point the system is ready to receive new data from the user program via Write calls.

#### 8.2.2 Write

The programmer employs the Write service to put new data into the data pool. A Buffer Description List must accompany the Write request. In each entry of this list, the programmer sets an action code to indicate whether the item is present in the buffer (Transmit) or missing (Skip). For each field which is present the list must also contain the item size. The first entry of the BDL pertains to the first item subsumed by the opened item. If this item is declared to be missing, all of its subitems are assumed to be missing. If the item is declared to be present, the next BDL entry pertains to its first subitem. This is completely parallel to the stepping of the pointers in the Read operation, described in Paragraph 8.1.2.

When an item is missing, this may be represented in the data pool in a variety of ways, such as: a file with 0 records, a variable length field of size 0, a fixed length field with its null bit set, or as an optional item which is absent. The

particular representation is chosen automatically by the Write service depending on the type of the item (defined in the Item List). The user need only set the action code for the item to Skip.

When a Write request is satisfied, the system pointers remain set at the item which follows the last item treated in the Buffer Description List.

### 8.2.3 Close-For-Writing

The programmer employs a Close-For-Writing call to indicate that he had finished writing the opened item. The Close request causes the DM-1 system to retrieve the data segment containing the item which logically follows the closed item. This item and the rest of its segment are copied into the output stream following the closed item.

After this is accomplished, the system directories are updated with the segment names of all the segments which were recorded from the Open procedure down to and including the Close procedure. Then the storage areas which the system was using for this item are released.

### 8.2.4 Insert-Data

The Insert-Data service parallels the input service Retrieve-Item. The programmer employs Insert-Data to accomplish an Open, a Write, and a Close all in one request. The user supplies a symbolic item name and a Buffer Description List with the call. All of the data which is to be written must be in the user's buffer when the request is issued. The system writes the data into the data pool and copies as much of the logically adjacent data as is required to maintain the integrity of the data stream. The service routine then updates the system directories and the operation is complete.

## 8.3 UPDATING SERVICES

The DM-1 updating routines provide the programmer with a convenient set of services for modifying an item which is in the data pool. The updating services combine input and output operations. While updating is in progress, two items are said to be active: the item from the data pool (input item) and the new version being created (output item). While updating, the user can issue a wide range of requests.

Some requests affect only the input item (Read and Seek), Insert affects only the output item, and the remaining calls affect both items (Replace, Delete, and Seek-With-Copy).

As in the writing procedure, the user must progress through the output item in a forward direction. However, in updating, he does not have to prepare all the output data; he can direct the system to copy data from the input item. The user is free to move the input item in any direction. But, before he calls on the system to perform a copy operation, the user must have his input item in the logically correct position. Copying requires that the structure of the input match the structure of the output. This does not imply that the record number of the input item must match that of the output item; only the structures must coincide.

#### 5.3.1 Open-For-Updating

The programmer employs the Open-For-Updating service to identify a section of the data pool which he intends to modify. The Open operation prepares the DM-1 system controls so that the system is able to respond to subsequent update request for any item subsumed within the opened item.

The Open-For-Updating operation begins in the same manner as Open-For-Writing. An Item List Table containing the structure definitions of the opened item is built. The data segment which should contain the first data of the opened item is retrieved. An output segment is initialized and any data in the input segment having an IPC less than the identifier of the opened item is copied to the output segment. Certain system pointers are then duplicated to permit the input item to be moved independently from the output item.

#### 8.3.2 Read, Seek

Read and Seek are identical to the services described under Input in Paragraph 8.1.

#### 8.3.3 Insert

Insert consists of a write operation; it affects only the output item. Consequently, it is logically limited to writing a new record. Any other type of data item would have at least some mark in the input item, and, therefore, a replace

operation would be used. The Insert call must be accompanied by a Buffer Description List. The user interface is exactly as that described for Write in Paragraph 8.2.

#### 8.3.4 Replace

The programmer employs the Replace service to substitute data in his buffer for the next item of input. Replace moves the data from the buffer to the output item under control of a BDL as in a write operation. Then the system skips the input pointers over the current input item. The user should account for this entire item in his BDL by means of either Transmit's or Skip's.

#### 8.3.5 Delete

The programmer employs Delete to eliminate the data corresponding to the current item. No BDL is needed with this call. The system simply marks the entire item as missing in the output; the particular mark used depends on the item type, as explained in Paragraph 8.2.2. On the input side, the system pointers are moved past the current item.

#### 8.3.6 Seek-With-Copy

The programmer employs the Seek-With-Copy service to copy a portion of the input item into the output. No BDL need accompany this call, since the data is completely defined in the input item. The user simply identifies the input item which is to terminate the copy operation. As with Seek, he may identify the item by a relative item number, or by a relative item number and a record number. In the latter case, the record number can be replaced by an end-of-file indicator or by a key value for an ordered file.

The Seek-With-Copy service begins by copying the current input item and continues down to, but excludes the item identified in the request.

#### 8.3.7 Close-For-Update

In many respects this operation parallels Close-For-Writing. The unique aspect of this service is that it copies any data remaining in the input item. If the input pointers are already set at the item following the opened item, this step is skipped. Next, any data following the opened item in the input segment is copied into the output stream. The system directories are updated to include the new segments written during the update procedure. Then the storage areas which the system was using are released.

### 8.3.8 Replace-Item

The programmer employs the Replace-Item service to substitute a new data item for an existing data item in a completely random manner. This single call causes the DM-1 system to edit data from the user's buffer and put it into the data pool. The system directories are updated so that the new data logically takes the place of the named item. In addition to the data, the user supplies a symbolic item name (as in the Open calls) and a Buffer Description List.

### 8.3.9 Delete-Item

This is a completely random service, similar to Replace-Item, but here the named item is replaced with a mark indicating that the item is missing. It follows that no Buffer Description List is required.

UNCLASSIFIED  
Security Classification

| DOCUMENT CONTROL DATA - R&D   |   |   |
|---|---|---|
| <i>(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)</i>   |   |   |
| 1 ORIGINATING ACTIVITY (Corporate author)<br>Auerbach Corporation<br>Philadelphia 3, Pennsylvania 19107   |   | 2a REPORT SECURITY CLASSIFICATION<br>Unclassified |
|   |   | 2b GROUP  |
| 3 REPORT TITLE<br>Reliability Central Automatic Data Processing Subsystem   |   |   |
| 4 DESCRIPTIVE NOTES (Type of report and inclusive dates)<br>Final Report  |   |   |
| 5 AUTHOR(S) (Last name, first name, initial)<br>Dr. J. Sable, W. Crowley, M. Rosenthal, S. Forst, P. Harper   |   |   |
| 6 REPORT DATE<br>August 1966  | 7a TOTAL NO. OF PAGES<br>780  | 7b NO. OF REFS                                    |
| 8a CONTRACT OR GRANT NO.<br>AF 30(602)-3820   | 9a ORIGINATOR'S REPORT NUMBER(S)<br>1280-TR   |   |
| b PROJECT NO 5519   | 9b OTHER REPORT NO(S) (Any other numbers that may be assigned this report)<br>RADC-TR-66-474 (3 Vols)             |   |
| c   |   |   |
| d   |   |   |
| 10 AVAILABILITY/LIMITATION NOTICES<br>This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMIL), GAFB, NY 13440.   |   |   |
| 11 SUPPLEMENTARY NOTES  | 12 SPONSORING MILITARY ACTIVITY<br>Rome Air Development Center (EMIID)<br>Griffiss Air Force Base, New York 13440 |   |
| 13 ABSTRACT<br><p>This is a three-volume final report produced for the Rome Air Development Center (RADC) under Contract AF 30(602)-3820. Volumes I and II are the Design Specification Report for the Automatic Data Processing Subsystem (ADPS) of Reliability Central, known as Data Manager-1 (DM-1). Volume III is a survey of major, computer-oriented on-line information and fact retrieval systems.</p> <p>The system design specification will be used for the implementation of the computer programs required to operate the RADC Reliability Central. The work reported in these volumes is an extension and detailing of the functional system design developed by Auerbach Corp. under Contract AF 30(602)-3433 and reported in RADC-TR-65-189, Design of Reliability Central Data Management Subsystem, July 1965. The DM-1 design provides for the incorporation of the reliability data collected by the Illinois Institute of Technology Research Institute (IITRI) under Contract AF 30(602)-3621 with Auerbach Corp. as subcontractor.</p> |   |   |

DD FORM 1473  
1 JAN 64

UNCLASSIFIED  
Security Classification

| 14. KEY WORDS  | LINK A |    | LINK B |    | LINK C |    |
|--|--------|----|--------|----|--------|----|
|  | ROLE   | WT | ROLE   | WT | ROLE   | WT |
| <p><b>File Structures</b><br/><b>Programming Languages</b><br/><b>Data Processing</b><br/><b>Storage and Retrieval</b></p> |        |    |        |    |        |    |

**INSTRUCTIONS**

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (corporate author) issuing the report.
- 2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.
- 2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.
3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.
4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.
5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.
6. **REPORT DATE:** Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.
- 7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.
- 7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.
- 8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.
- 8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.
- 9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.
- 9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers (either by the originator or by the sponsor), also enter this number(s).
10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those

imposed by security classification, using standard statements such as:

- (1) "Qualified requesters may obtain copies of this report from DDC."
- (2) "Foreign announcement and dissemination of this report by DDC is not authorized."
- (3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through \_\_\_\_\_."
- (4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through \_\_\_\_\_."
- (5) "All distribution of this report is controlled. Qualified DDC users shall request through \_\_\_\_\_."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.
12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring (paying for) the research and development. Include address.
13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (TS), (S), (C), or (U).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.