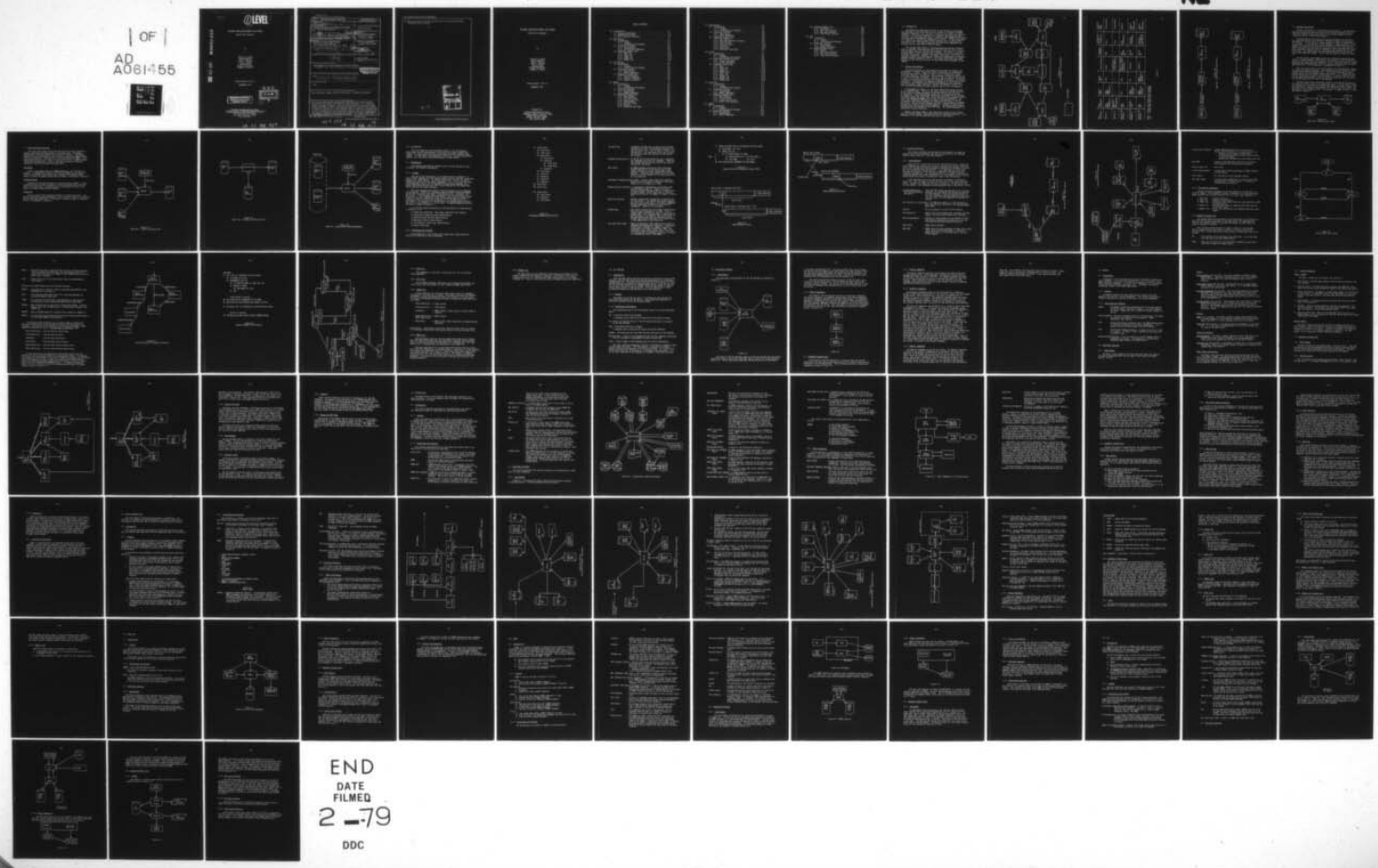
AD-A061 455 MICHIGAN UNIV ANN ARBOR GRADUATE SCHOOL OF BUSINESS--ETC F/G 9/2
MICHIGAN TRANSLATOR PROGRAM LOGIC MANUAL. VERSION IIB, RELEASE --ETC(U)
SEP 77 C E BURPEE, D DESMITH, L A HUTCHINS DCA100-75-C-0064
UNCLASSIFIED WP-77-DT-3.7 SBIE-AD-E100 110 NL

| OF |
AD
A061455



END
DATE
FILMED
2-79
DDC

(1) LEVEL II

MICHIGAN TRANSLATOR PROGRAM LOGIC MANUAL

Version IIB, Release 1

by

Charles E. Burpee
Donald DeSmith
Linda A. Hutchins
Eric L. Kintzer
Kenneth Moore
Michael Stolarchuk
Gregory J. Wolfe

Working Paper 77 DT 3.7

September 1977

Database Systems Research Group
Graduate School of Business Administration
The University of Michigan
Ann Arbor, Michigan 48109
(313) 763-1100

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: Working Paper, 77 DT 3.7	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): MICHIGAN TRANSLATOR PROGRAM LOGIC MANUAL. Version IIB, Release 1.	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER: WP-77-DT-3.7
7. AUTHOR(s): Charles E. Burpee, Donald DeSmith, Linda A. Hutchins, Eric L. Kintzer, Kenneth Moore, Michael Stolarchuk & Gregory J. Wolfe	8. CONTRACT OR GRANT NUMBER(s): DCA100-75-C-0064
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Database Systems Research Group, 276 Business Administration, Univ. of Michigan, Ann Arbor, MI 48109	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 32017K, 27400, 27402
11. CONTROLLING OFFICE NAME AND ADDRESS: DEFENSE COMMUNICATIONS AGENCY, C422, WASHINGTON D.C. 20305	12. REPORT DATE: September 1977
	13. NUMBER OF PAGES: 83
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 12) 86p.	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

~~This document is not yet approved for public release.~~
CLEARED FOR OPEN PUBLICATION, DISTRIBUTION UNLIMITED

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A (18) SBIE (19) AD-E100 110

18. SUPPLEMENTARY NOTES

none

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Translator, Reader, Writer, Restructurer , Software maintenance

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

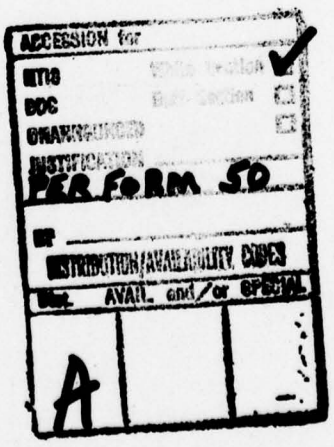
This Program Logic Manual provides the information for maintaining the software for the Version IIB, Release 1 Data Translator. It is directed toward two groups of people: users of the Data Translator who desire some insight into "how it works" and programmers who will maintain the software. The manual is written in a manner that it could be understood by people with no prior experience with data translation. However, the reader should have a firm grasp of database concepts and terminology and should be knowledgeable of Honeywell software systems (especially IDS/1). The reader should also →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. have some familiarity with the facilities of the World Wide Data Management System (WWDMS).



1A73B

MICHIGAN TRANSLATOR PROGRAM LOGIC MANUAL

Version IIB, Release 1

by

Charles E. Burpee
Donald DeSmith
Linda A. Hutchins
Eric L. Kintzer
Kenneth Moore
Michael Stolarchuk
Gregory J. Wolfe

Working Paper 77 DT 3.7

September 1977

Prepared for

Defense Communications Agency
Command & Control Technical Center
WWMCCS ADP Directorate
Reston, Virginia 22090
DCA 100-75-C-0064

TABLE OF CONTENTS

1.0 INTRODUCTION

The Database Systems Research Group at the University of Michigan has developed a Data Translator which is capable of reorganizing WWDMS databases (Sequential, ISP, and IDS) by altering the logical and physical structure. The work was completed on a Honeywell 6060 computer under a contract from the Command and Control Technical Center WWMCCS ADP Directorate (Code 400) of the Defense Communications Agency. The Data Translator is a complex and sophisticated software package, yet is understandable and easy to use. This manual describes the high-level components and program logic for each Data Translator module.

This program logic manual is directed toward two groups of people: users of the Data Translator who desire some insight into "how it works" and programmers who will maintain the software. The manual is written in a manner that it could be understood by people with no prior experience with data translation. However, the reader should have a firm grasp of database concepts and terminology and should be knowledgeable of Honeywell software systems (especially IDS/1). The reader should also have some familiarity with the facilities of the World Wide Data Management System (WWDMS).

1.1 History and Overview

This Data Translator is the product of many years of research and development. Previous Translators lacked the features and performance of the current release. Refer to Table 1-1 for a comparison of Data Translator features. The Version I was developed to show the feasibility of a generalized Data Translator. Version II was an extension of Version I since it was capable of handling tree-type data structures. The Version IIA Release 1 was designed in response to demand for a restructuring Data Translator. The Release 2 had improved performance and more features than the Release 1 Data Translator. Version IIB Release 1 offers still more improved performance and more features than Version IIA Release 2.

As diagrammed in Figure 1-1, the data translation process consists of six separate steps. The first two steps involve writing descriptions of the source and target databases using their MD sections and level 61 extensions. Those descriptions should be run through the IDS Analyzer to produce source and target SDDL (Stored Data Definition Language) tables. (The terms source and target refer to the old and new databases.) The third step is encoding the source-to-target transformation in the TDL (Translation Definition Language) and running the TDL Analyzer to produce TDL tables. The fourth step, Reader execution, creates the Restructurer Internal Form (RIF) of the source database(s). The Restructurer produces the Relational RIF database in the fifth step. The target database(s) is produced by running the Writer.

Overall, the basic steps of the translation process can be divided into two distinct phases - data description and translator execution. The remainder of this section describes the steps in more detail.

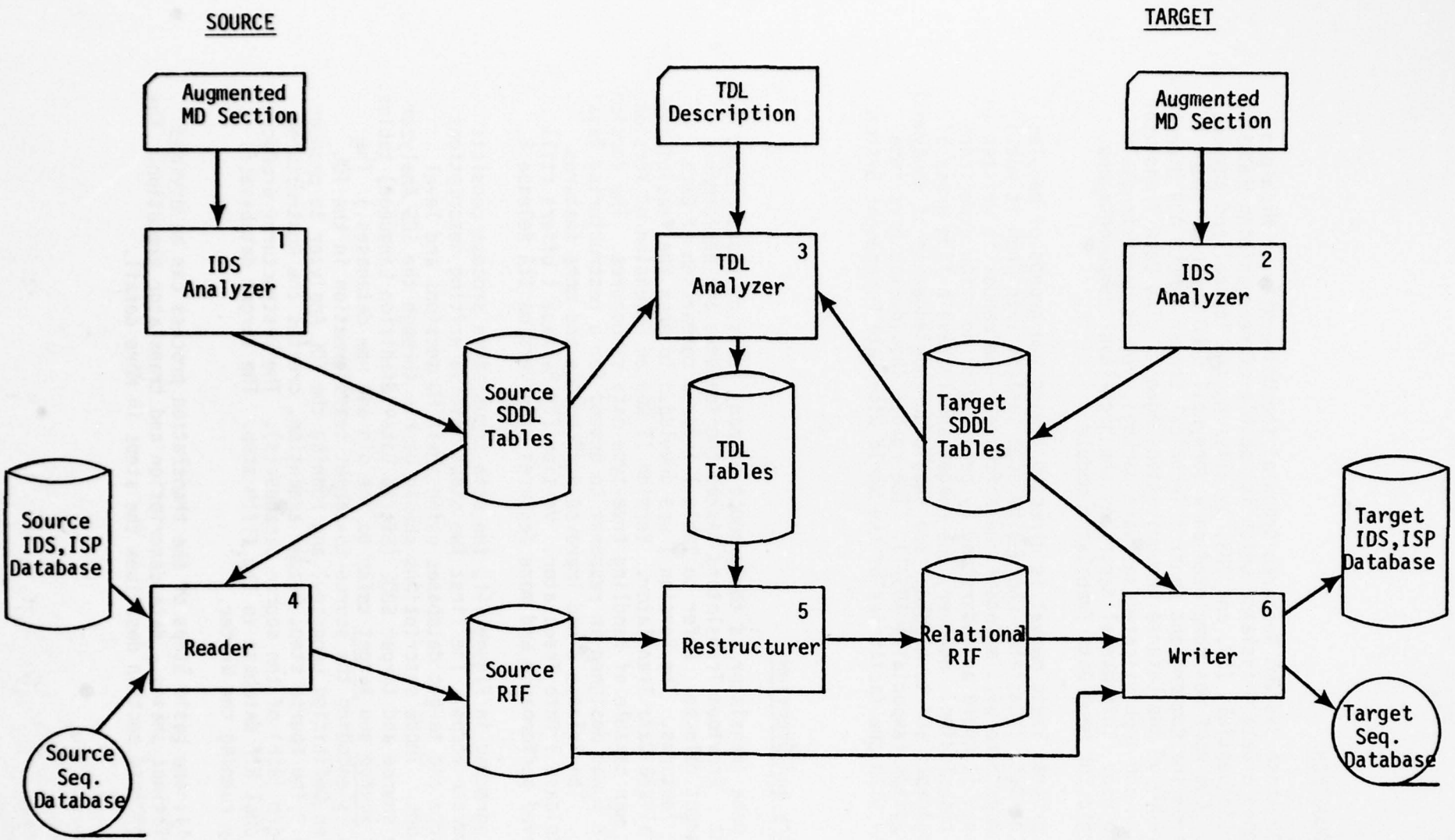
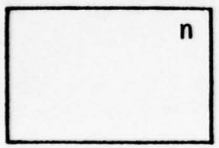


Figure 1-1
Version IIB Release 1 Architecture



n - step number

Data Translator	Date Completed	Source DBMS	Target DBMS	Restructuring	Language Analyzers*	Incremental/ Partial Translation	Internal Data Model
Version I	1973 WWDMS Sequential	NIPS, WWDMS Sequential	NIPS	Some	SDDL	none	Hierarchical
Version II	Designed Not Implemented	----	----	Hierarchical Transformations	----	none	Hierarchical
Version IIA Release 1	June 1976	IDS	IDS	Powerful Network Database Trans-formations	SDDL IDS MD TDL	Some Incremental	Network
Version IIA Release 2	December 1976	WWDMS Sequential, ISP,IDS	IDS	Complete	IDS MD TDL	Complete Incremental	Network and Relational
Version IIB Release 1	July 1977	WWDMS Sequential ISP,IDS	WWDMS Sequential ISP,IDS	Complete	IDS MD TDL	Complete Incremental/ Some Partial	Network and Relational

*SDDL - Stored Data Definition Language
TDL - Translation Definition Language

Table 1-1

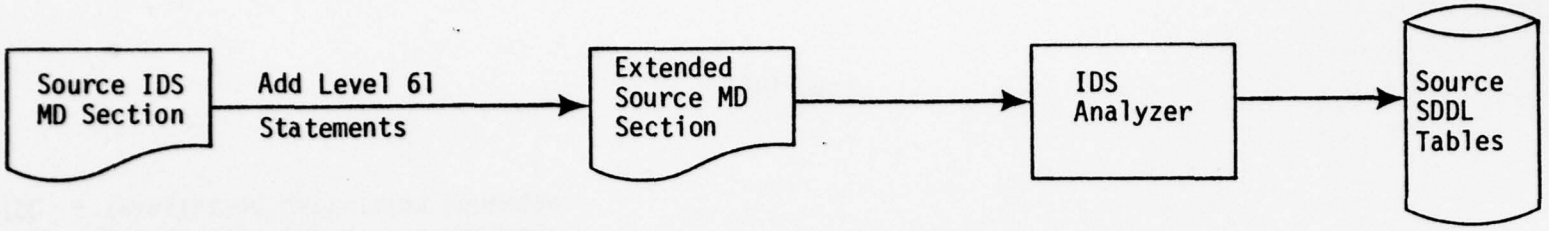


Figure 1-2a
Step One: Create Source SDDL Tables

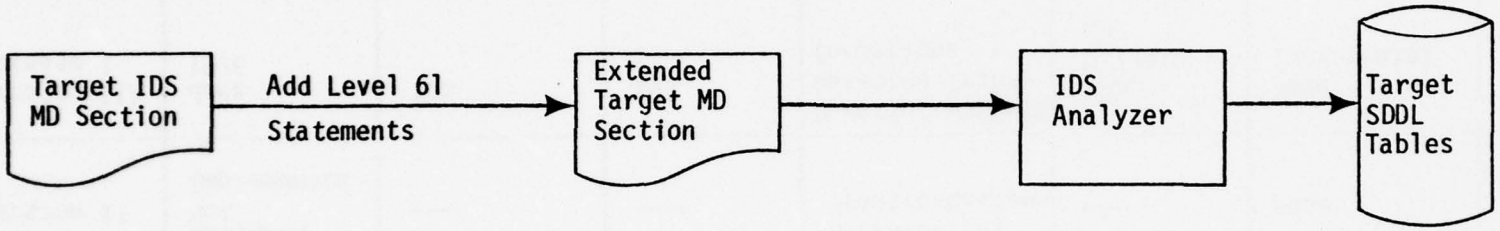


Figure 1-2b
Step Two: Create Target SDDL Tables

Figure 1-2
Steps One and Two of Data Translation

1.2 Database Description

The Data Translator is a description-driven process. The Translation modules must know the format of the source and target databases and the rules for creating records and chains in the target database from the records and chains in the source database.

The descriptions of the source and target databases are the source and target MD sections and additional information needed to restructure the database. If the source or target database is IDS, no new MD section is necessary. If the source or target database is WWDMS Sequential or ISP, however, an IDS MD which describes the database will have to be written. After the MD sections have been collected or written, and additional information encoded as special level 61 statements, the extended MDs are ready to be used. The IDS Analyzer uses the extended MD sections to produce source and target SDDL tables. These first two steps are shown in Figure 1-2.

The SDDL tables are databases which hold information describing the source or target database. SDDL tables are analogous to the IDS Definition Structure which describes IDS databases. Even if multiple source or target databases have been described, there will be only one source and one target SDDL table file. The IDS Analyzer is documented in Section 2.

The final description to be written details the transformation between the source and target databases. That description must be written in the Translation Definition Language (TDL). The TDL describes how to create target records using the source records and chains. It is imperative that the TDL description be correct and validated. If the user does not write the TDL description properly, the output of the Translator will be invalid and the Restructuring will have to be repeated.

The third step of the Translation process is running the TDL description through the TDL Analyzer to produce TDL tables. The TDL tables cannot be created until both source and target SDDL tables have been created. The TDL tables are an internal representation of the TDL description which the Restructurer can conveniently use. TDL analysis is shown in Figure 1-3 and described in Section 3.

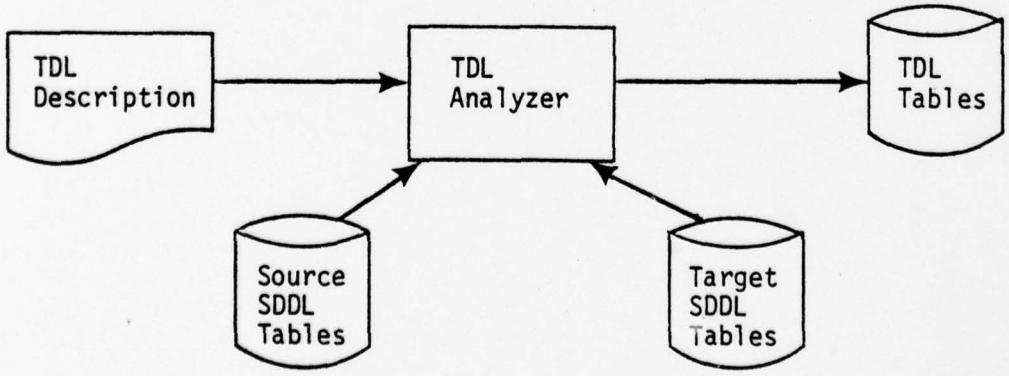


Figure 1-3
Step Three - Creating TDL Tables

1.3 Data Translator Execution

The final three steps involve the execution of the Data Translator, thereby translating the source data to its target form. The Reader, Restructurer, and Writer modules are executed in sequence. The Reader converts the source database(s) into a logically equivalent ADBMS database called the Source RIF. (See Section 7 for a description of ADBMS). The Restructurer converts all or part of the Source RIF into an ADBMS database called the Relational RIF. Finally, the Writer produces the desired database(s) from the Relational and Source RIFs.

The Reader

There is one Reader module for WWDMS Sequential, ISP, and IDS file systems. It sequentially accesses the source database(s) and produces a logically equivalent Source Restructurer Internal Form database (SRIF). Figure 1-4 shows the Reader process and the Reader is described in Section 4.

The Restructurer

The heart of the Data Translator is the Restructurer module. It uses information stored in the TDL tables to create a Relational RIF from the data in the source RIF. The Restructurer is described in Section 5 and Figure 1-5 shows the Restructurer process.

The Writer

The last step of the translation process is running the Writer. The Writer populates the target database(s) using the target (source + relational) RIF and the target SDDL tables. The Writer is shown in Figure 1-6 and the Writer is documented in Section 6.

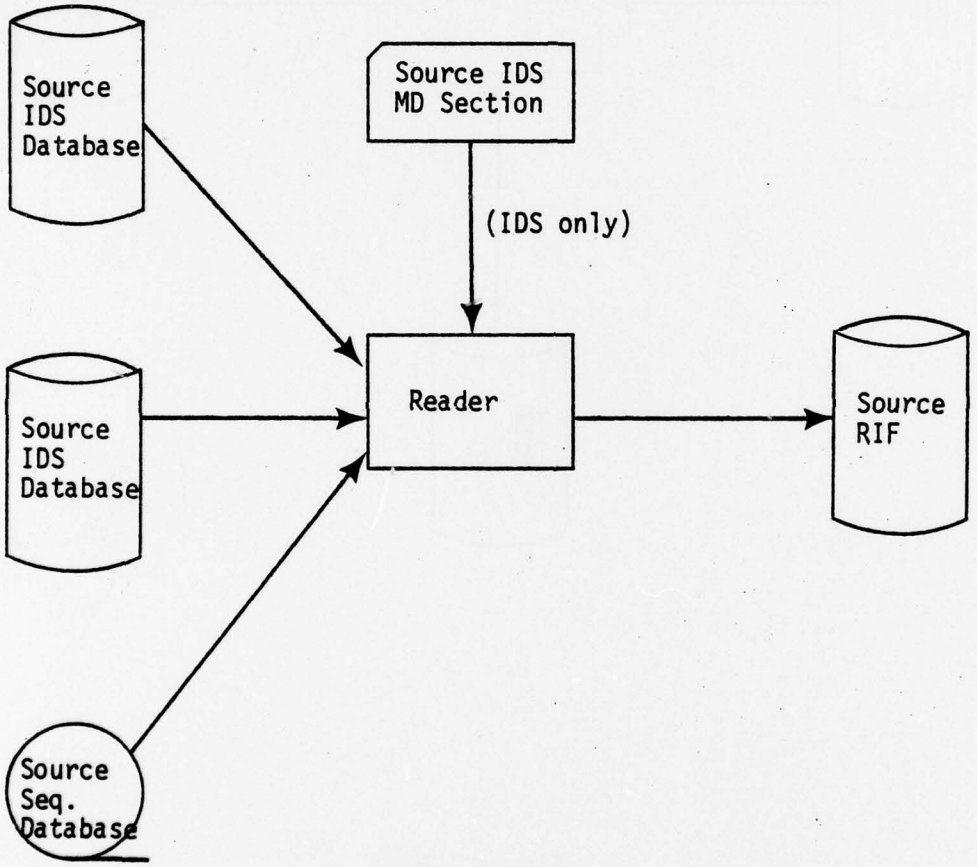


Figure 1-4
Step Four: Creating the Source RIF

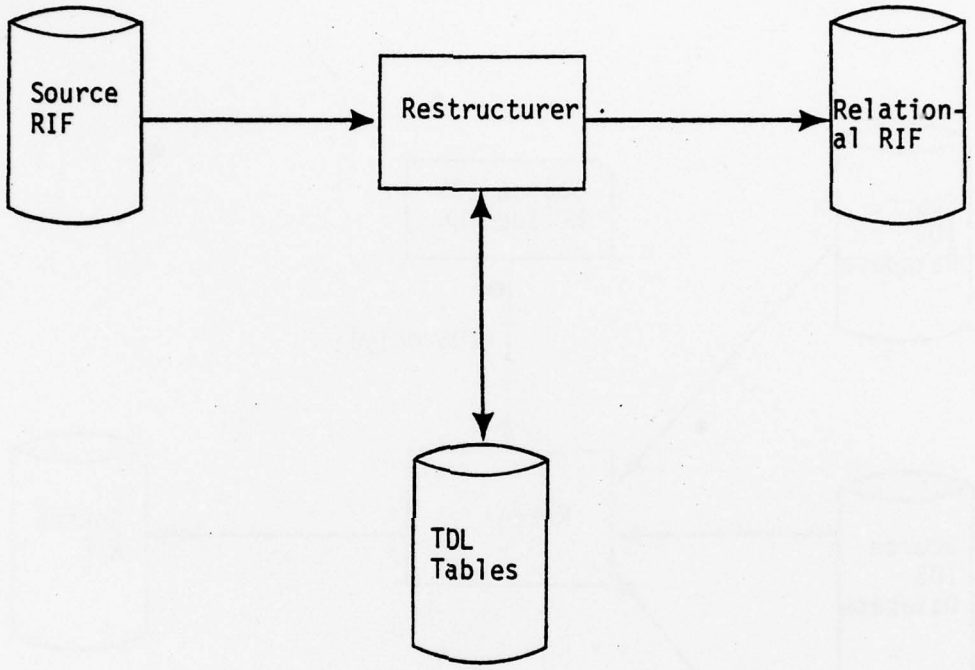


Figure 1-5
Step Five: Creating the Relational RIF

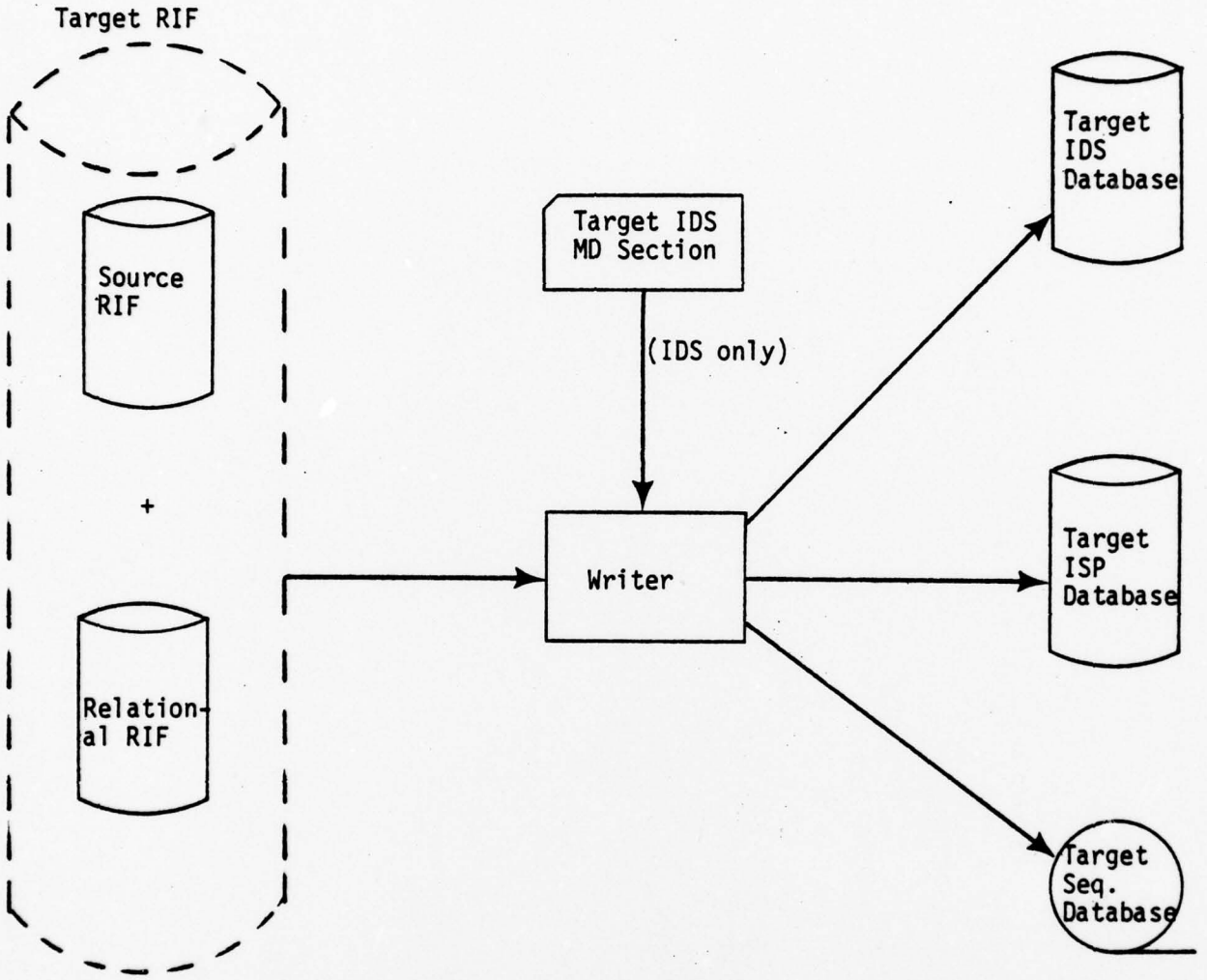


Figure 1-6
Step Six: Creating the Target Database(s)

2.0 IDS ANALYZER

The first module of the translation process is the IDS Analyzer which takes the user's description of the source or target database and converts it to the Data Translator Stored Data Definition Language (SDDL) tables. The SDDL tables are subsequently used by all remaining translation modules, e.g. the entire translation process is table driven.

2.1 Introduction

This section describes the capabilities of the IDS Analyzer as well as some of the relevant terminology.

2.1.1 Purpose

The role of the IDS Analyzer is to compile/analyze statements in a user friendly language into the internal tables used to drive the translation process (e.g. SDDL tables). Two tables (SDDL) exist, one for the source database(s) and one for the target database(s); e.g. if the user has three source databases that are to be combined into one target database, the source SDDL tables describe all three input databases and the target SDDL tables describe the output database.

The input database descriptions to the IDS Analyzer are an augmented IDS MD section. Extension is necessary because certain information needed by the Data Translator algorithms is unobtainable from a "straight" IDS MD section. If the database being described is IDS, the MD section already exists, but for ISP or sequential files the MD section must be written according to specific rules given in Section 3.7 of the Version IIB R.1 User Manual. All extensions to the MD section are accomplished by adding a new level to the language, e.g. the 61 level. A sample extended MD section prototype is shown in Figure 2-1.

A list of the required extensions to the MD section is (terms defined in Section 2.1.2):

a) Primary key definition (the unique identifiers for records)

b) Contained-in-repeating group identification

c) Phantom pointer relation identification

d) Match-key relation identification

e) ISP/sequential record type identification

f) Special data types

2.1.2 Terminology and Concepts

The following is a list of most major terms used in describing the function and algorithm of the IDS Analyzer.

```
01  record entry
    02  field entry
    02  field entry
    02  group entry
        03  group entry
        61  extension
            04  field entry
            04  field entry
        61  extension
    02  field entry
    61  extension
    61  extension
    61  extension
    61  extension
98  chain entry
98  chain entry

01  record entry
    02  field entry
    61  extension
    02  field entry
        .
        .
        .
```

Figure 2-1
Prototype Extended IDS MD Section

IDS Data Query - A Honeywell package used to convert the extended IDS MD section into an intermediate database, the IDS Data Query File. This is used as input to the IDS Analyzer. The Data Query File is an IDS database which contains the extended IDS MD section as its data.

Extended IDS MD section - An IDS MD section describing the user's databases but augmented with level 61 statements. There are two extended IDS MD sections, one for the source and one for the target.

SDDL tables - An ADBMS database containing a Data Translator useable description of the source (source SDDL tables) database or target (target SDDL tables) database. A complete data definition of the SDDL tables is given in Section 11.0 of the Version IIB R.1 Low Level PLM.

Contained-in-repeating group (CIRG) - Certain record types have repeating groups within the physical confines of the record. Figure 2-2 is an example of a CIRG.

Phantom pointer relations - A technique of implementing an IDS chain without IDS knowing about it. The user selects a field and places reference codes in it which point to other records. No 98 level chain is used to define the relation. An example is shown in Figure 2-3.

Match key relations - Another technique for implementing relations without IDS control. Two records are related together by having item values match. The parent, dependent and key items must be described using level 61 statements. Figure 2-4 is an example.

Primary keys - Each record instance must be uniquely identifiable from all other instances of its type. This is accomplished by designating items in the record as primary keys. It is possible that a record may not have the primary key items in the record itself, instead item values from owner record types assist in the composition of the primary key.

Set-significant items - These are additional items added to the records to implement the Restructurer algorithm. A set-significant item is created in a member instance to correspond to all owner primary key items. Set-significant items may be primary keys. Their use in restructuring is completely explained in Section 2 of the Version IIB R.1 User Manual.

```
01  STATES-IN-UNION TYPE IS 5 RETRIEVAL VIA CALC CHAIN.
    02  STATE-NAME  PIC X(10).
    02  URBAN SIZE 336.
        03  CITY-DATA OCCURS 14 TIMES.
CIRG    04  POPULATION-OF-CITY  PIC 9(8) COMP-1.
        04  CITY NAME           PIC X(10).
98  CALC CHAIN DETAIL RANDOMIZE ON STATE-NAME.
```

Figure 2-2
Sample Contained-in-Repeating Group (CIRG)

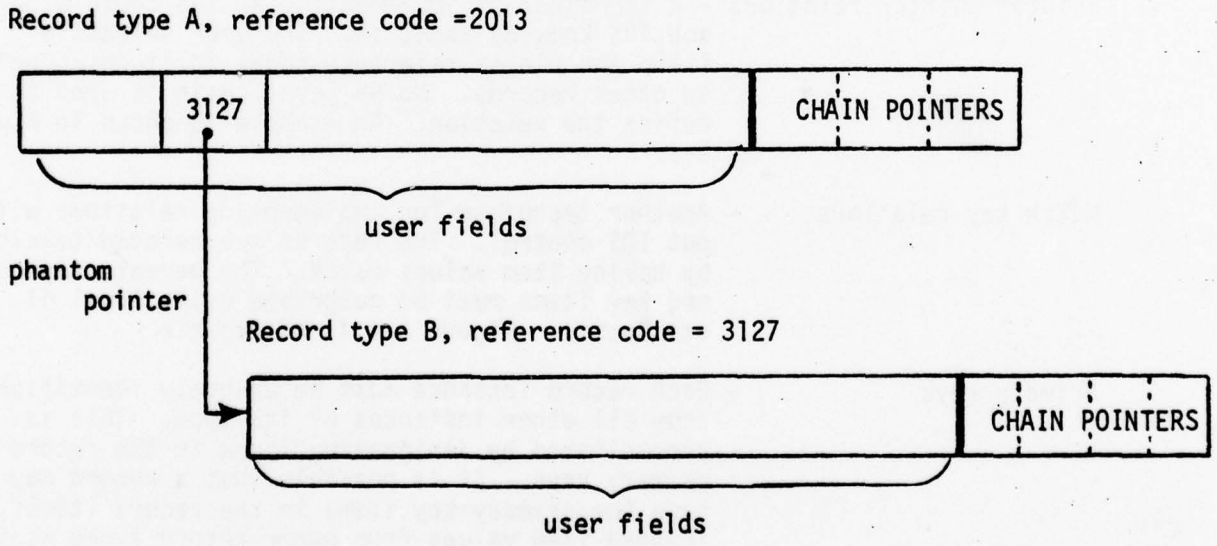


Figure 2-3
Sample Phantom Pointer

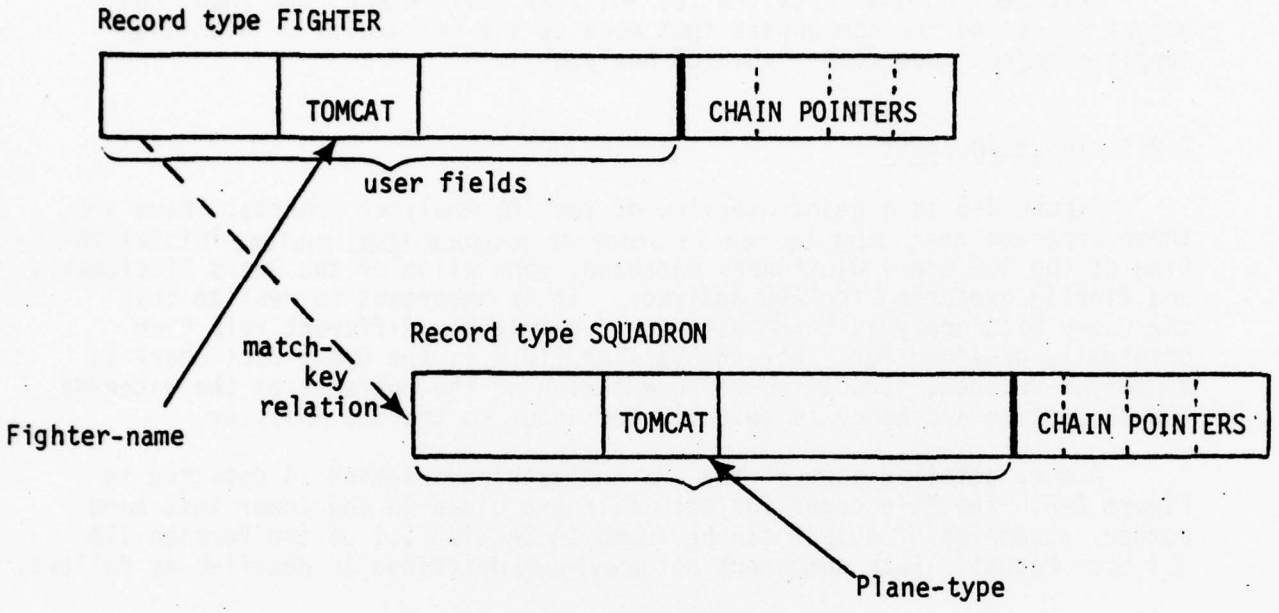


Figure 2-4
Sample Match-key Relation

2.2 Functional Overview

This section describes the IDS Analyzer environment, the input and output files and the components that make up the University of Michigan supplied object code (e.g. "the IDS Analyzer").

2.2.1 Inputs/Outputs

Figure 2-5 is a grand overview of the IDS Analyzer process. There are three programs that must be run in order to produce SDDL tables; initialization of the IDS Query Dictionary database, population of the Query Dictionary, and finally executing the IDS Analyzer. It is important to realize that the Query Dictionary is being used for a completely different role than originally designed for. For the Version IIB R.1, the Query Dictionary is a machine-encoded, structured representation of the contents of the extended IDS MD section and hence is suitable for input to the IDS Analyzer.

A more detailed view of the IDS Analyzer environment is depicted in Figure 2-6. The file codes for each file are given in the lower left-hand corner. Examples of output can be found in Section 5.1 of the Version IIB R.1 User Manual. Each component not previously described is detailed as follows.

Extended MD/database and combined extended MD section - Since the SDDL tables can describe up to five user databases, an extended MD section must be prepared for each database. These are merged together by the user (resolving duplicate names and record id's) into one input file, the combined IDS extended MD section.

IDS Translator in query mode - The COBOL-IDS compiler ($ IDS) modified to build the IDS Query Dictionary (see IDS Data Query, DD46, DD47)

IDS QUTI - Utility program to initialize IDS databases.

IDS Analyzer R* - Object code of the program that constructs the SDDL tables. Written by the University of Michigan.

Run-time parameters - Defines all record types and the database to which they belong (there being no other way of telling them from the combined extended MD section)

SDDL tables - ADBMS output database

SDDL DBTF - ADBMS table file that describes the SDDL table layout prior to populating the database. The role of ADBMS table files is fully described in Section 7 of this manual.

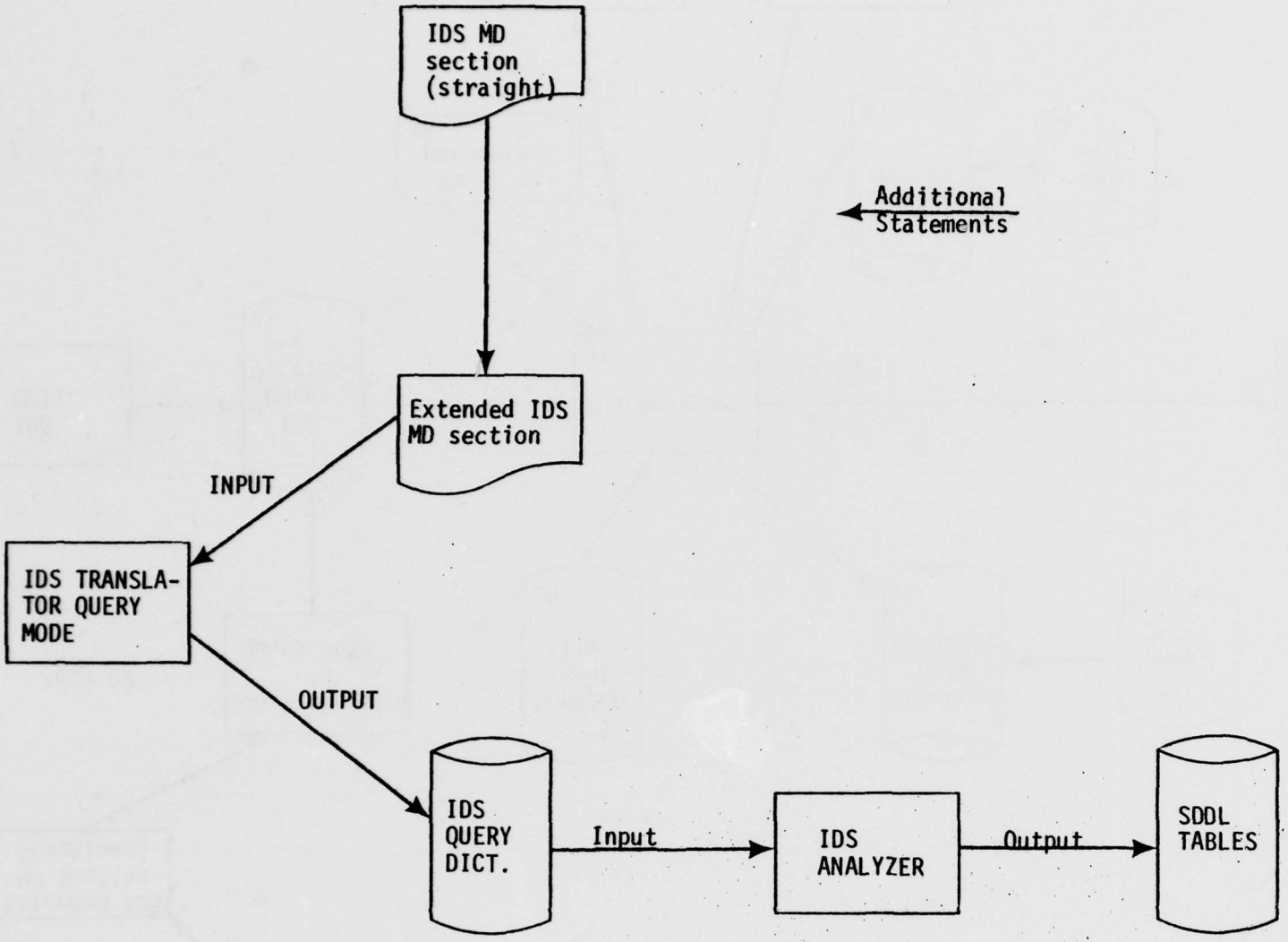


Figure 2-5
Overview of SDDL Table Creation

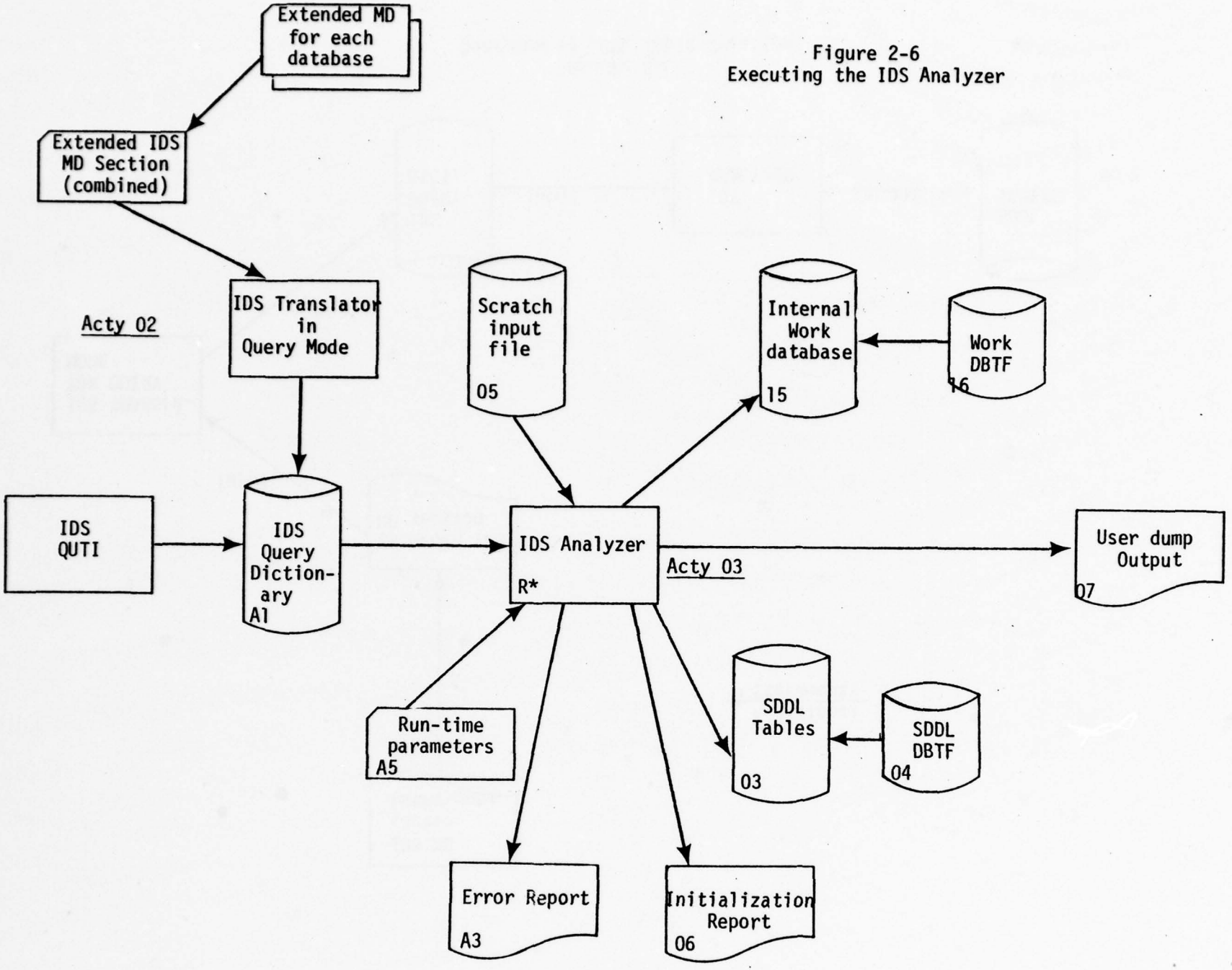


Figure 2-6
Executing the IDS Analyzer

Internal Work database - Another ADBMS database used for a variety of uses by the IDS Analyzer. It contains tables for
- keeping track of symbols encountered
- collecting information on which records belong to which databases
- primary key information as described by the user.

Work DBTF - Similar to the SDDL DBTF except that it defines the structure of the Internal Work database

Scratch input file - Null input

Initialization report - A brief page containing messages of ADBMS database initialization.

Error report - All IDS Analyzer error messages printed.

User dump output - A useful report summarizing in easy-to-read format the contents of the SDDL tables. It serves as a reference when writing TDL.

2.2.2 IDS Analyzer Components

The most convenient breakdown of the IDS Analyzer is by the way it is stored in memory. Each component occupies a separate link overlay and at any given time only one link plus the main link is in core. Each component is:

1. Main link - Control structure
2. INIT link - Database initialization
3. IDSAN link - Constructs SDDL tables except for set-significant items and primary keys
4. BLDPK link - Finishes SDDL tables by adding primary keys and set-significant items.
5. REPORT link - Writes the IDS Analyzer user-readable SDDL table dump report.

2.3 Component Program Logic

The essence behind the algorithm of the IDS Analyzer is the transfer of data from the IDS Query Dictionary to the SDDL tables. To understand the process, the database schemas of the Query Dictionary and SDDL tables are shown in Figures 2-7 and 2-8.

Only a partial schema diagram is shown in Figure 2-7 for the SDDL tables, complete details are given in section 11 of the Version IIB R.1 Low Level PLM. A brief definition of the contents of each record type is as follows:

DB - One instance per user database being described. It has the name and type (IDS, ISP, SEQ) stored within.

GROUP - Each user 01 record plus all contained-in-repeating groups have their own instance of a GROUP record.

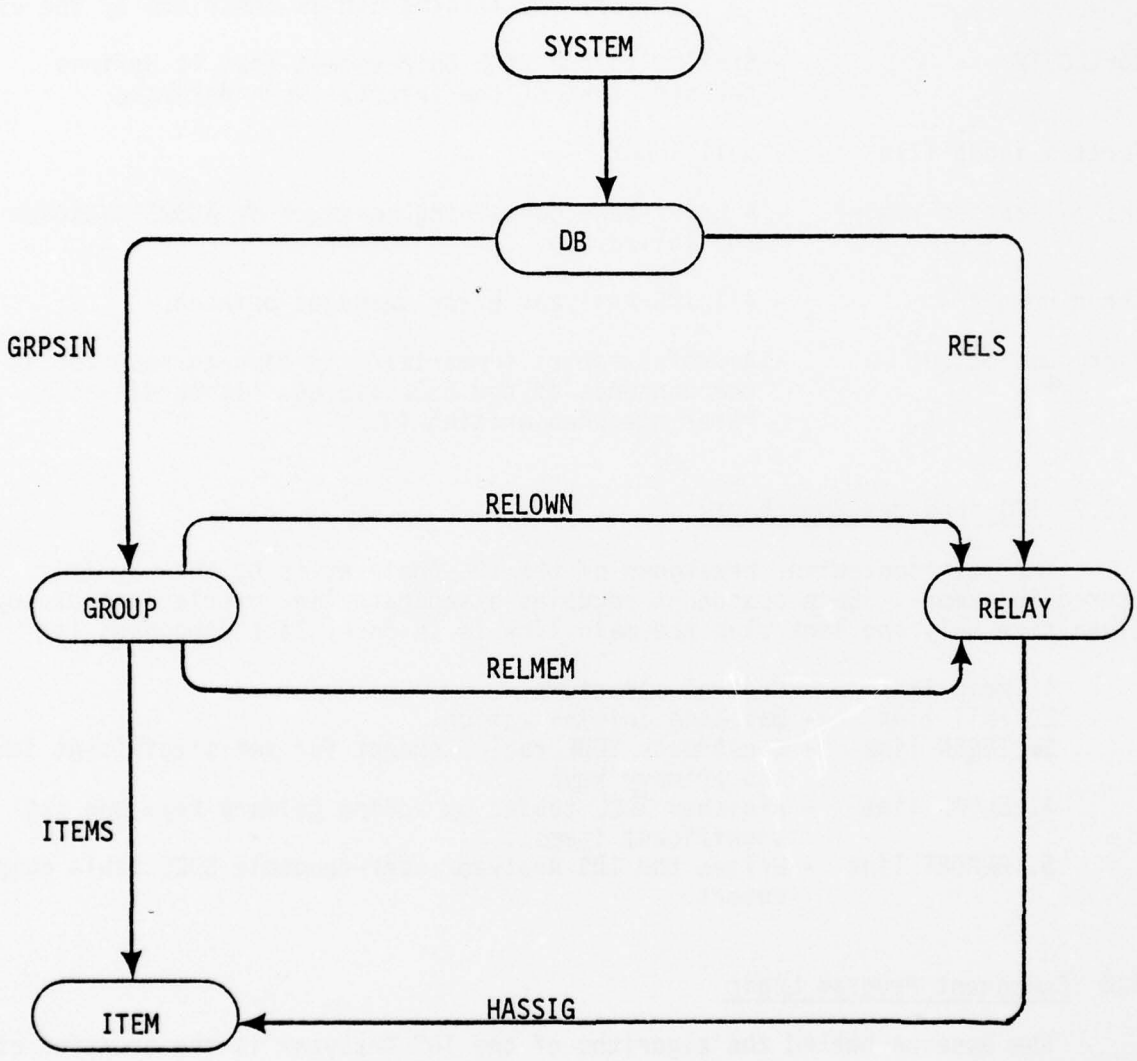


Figure 2-7
Partial SDDL Table Schema

RELAY - Each 98 level chain, phantom pointer relation, match-key relation, relation between the containing and contained-in group and all entry point relations (e.g. SYSTEM to CALC records) have their own RELAY record instances

ITEM - Every field plus all set-significant items are represented by ITEM records

The sets in the SDDL tables have the following functions:

GRPSIN - Each group (e.g. record or CIRG) is attached along GRPSIN to the database that it belongs to.

RELS - All relations whose owner record is in the same database are collected together along RELS.

ITEMS - All user and set-significant items belonging to the same group are attached to the corresponding GROUP record along ITEMS.

RELOWN - Each relation that a group owns is found along RELOWN. Similarly, given a relation, the owner group is identified by 'heading' the RELOWN set.

RELMEM - Same as RELOWN except for relations that a group is a member of.

HASSIG - All set-significant items that were derived from the same relation are collected together along HASSIG.

The IDS Data Query Dictionary has a structure shown in Figure 2-8. Only the portions of the structure actually used by the IDS Analyzer are displayed. The contents of the records are briefly described here, a more complete explanation is given in IDS Data Query Installation DD47.

Record-Definition - One for every 01 record type

Field-Definition - One for every 02 field

Validation - One for every 02-49 entry

Description - One for every level 61 entry

Master-Definition - One for every 98 chain master entry.

Detail-Definition - One for every 98 chain detail entry.

As can now be seen, the extensions to the IDS MD section that are coded by the user are stored internally as Description records. The Description records for CIRG definition are located immediately beneath the Validation record that defines the CIRG. Primary key, match-key and phantom pointer relation information are always coded directly following a special field in every record, 02 TRANSLATION-INFORMATION SIZE 0. An example extended MD section is given in Figure 2-9 with the corresponding IDS Query Dictionary contents shown in Figure 2-10.

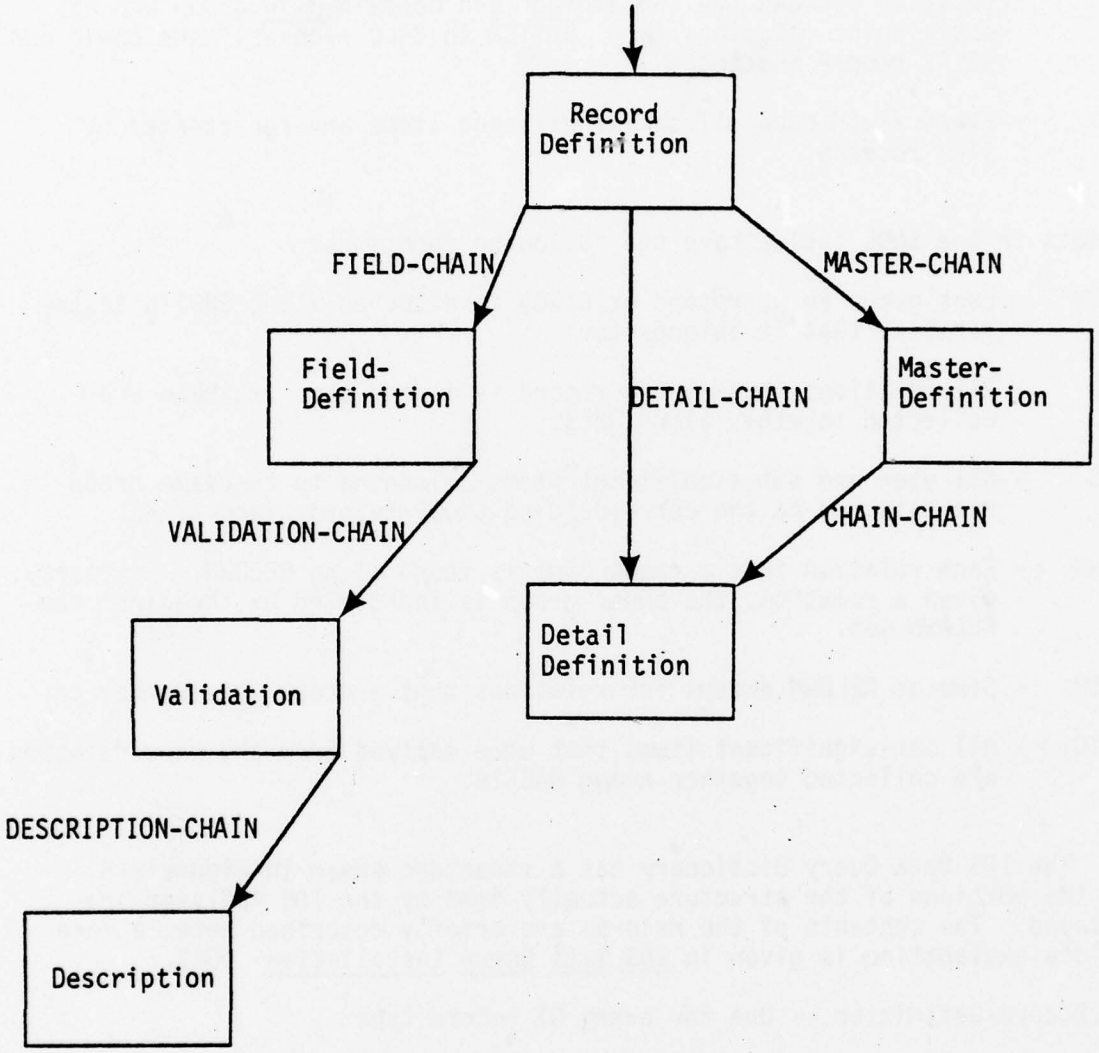


Figure 2-8
"Used" Portions of the Query Dictionary

```
IDS QUERY
01  CITY TYPE IS 1 RETRIEVAL VIA CALC CHAIN.
    02  CITY-NAME PIC X(15).
    02  DEPARTMENTS SIZE 100.
        03  CITY-DEPTS OCCURS 10 TIMES SIZE 100.
        61  OCCURS 10 TIMES.
            04  DEPT-NAME PIC X(10).
        61  EOG.
    .
    .
    .   other 02-49, 61 entries
98  CALC CHAIN DETAIL RANDOMIZE ON CITY-NAME.
98  HAS-OFFICIALS CHAIN MASTER CHAIN-ORDER IS AFTER.

01  OFFICIALS TYPE IS 2 RETRIEVAL VIA HAS-OFFICIALS CHAIN.
    .
    .
    .   02-49 + 61 entries
98  HAS-OFFICIALS CHAIN DETAIL SELECT CURRENT MASTER.
```

Figure 2-9
Sample Extended IDS MD Section

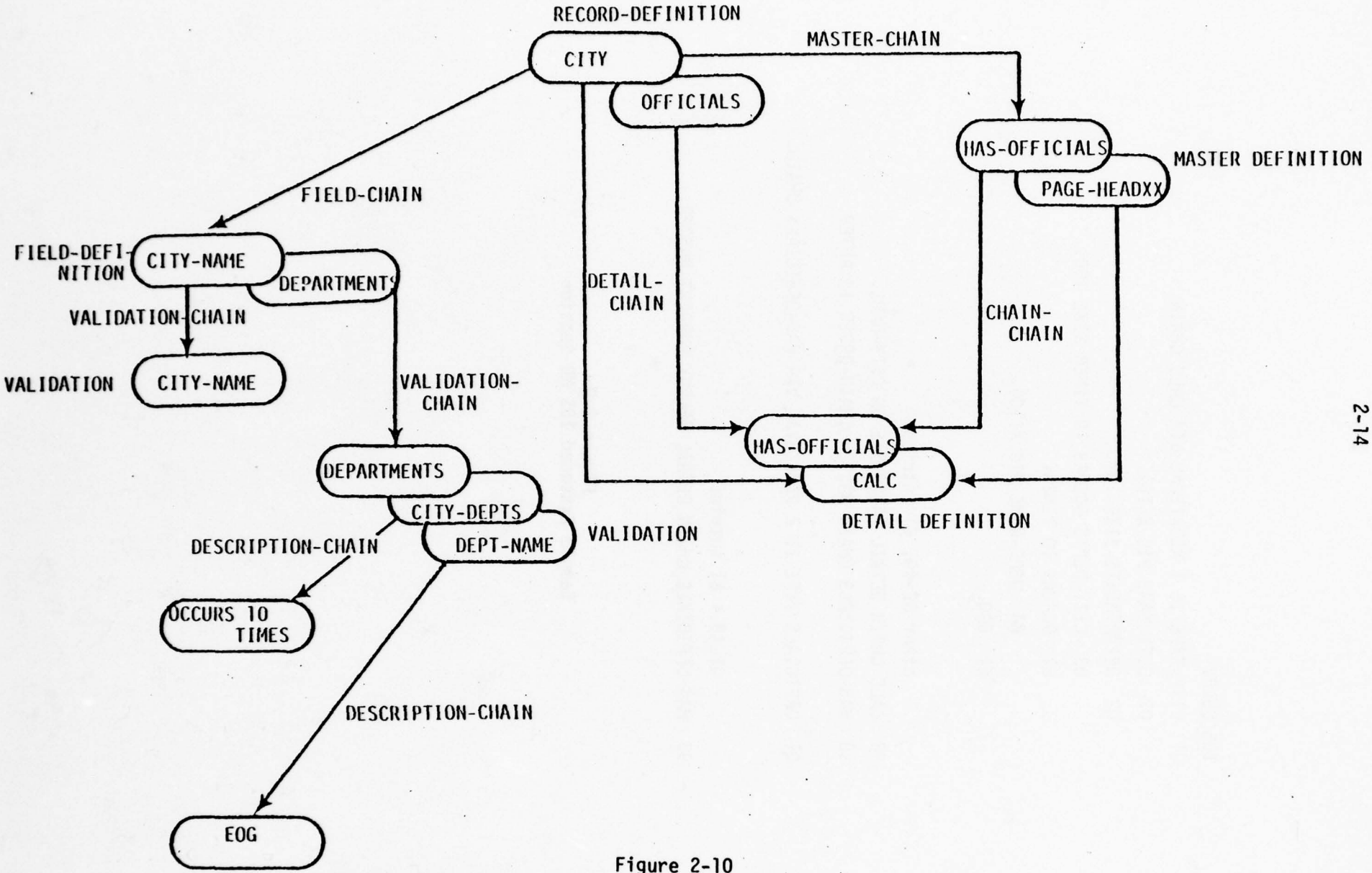


Figure 2-10
IDS Query Dictionary Built from Extended MD Section of Figure 2-9

2.3.1 MAIN Link

This component is very small, serving only to link in the other three components.

2.3.2 INIT Link

The two ADBMS databases, SDDL tables and Internal Work database are initialized using the contents of their respective ADBMS table files.

2.3.3 IDSAN Link

Ninety percent of all processing takes place within this component. Conceptually, the algorithm is simple. The program traverses the IDS Query Dictionary building the appropriate SDDL table records as it plods along. The actual transformations between the Query Dictionary and SDDL tables are as follows:

Record-Definition	=>	Group records
Field-Definition	=>	ITEM records
Validation	=>	GROUP (CIRG) or ITEM records or RELAY (CIRG relations)
Master-Definition, Detail-Definition	=>	RELAY records
Description	=>	GROUP (CIRG), RELAY (match-key or phantom pointer relations)

Additionally, Description records that identify primary keys are entered in a different format within the Internal tables for use by the next link.

2.3.4 BLDPK Link

Set-significant items can only be created once every group's primary keys have been defined and since set-significant items may also be primary keys, all processing related to these entities is deferred until this component which is executed upon the completion of the IDSAN link.

The basic algorithm is to locate groups whose primary keys are solely user items and indicate that in the SDDL tables. Given this it is now possible to generate set-significant items for record types whose owner record types have already had their full primary key built. The process is repeated recursively until all set-significant items and all primary keys have been generated. The process can be visualized as working from the top of the database downwards.

2.3.5 REPORT Link

The SDDL tables, by now completely constructed are traversed so that a report summarizing the contents is produced. By database, every group is listed (with each of its items and with each relation that it is an owner or member of). Then all relations with owner and member groups are summarized.

3.0 TDL ANALYZER

3.1 Introduction

After the SDDL tables have been successfully constructed using the IDS Analyzer, the user must specify his desired restructuring in a translator-recognizable form. This user-written restructuring specification is called the TDL (Translation Definition Language) description. Using the TDL, the user directs the construction of target records by specifying a traversal of the source records in the source RIF. This TDL Description is then analyzed and tables are produced which drive the translator.

3.1.1 Purpose

The purpose of the TDL Analyzer is to process the user-supplied TDL Description and produce the TDL tables. These tables are then used to drive the translator during the execution phase.

3.1.2 Terminology and Concepts

The terminology used in the TDL Analyzer section is briefly described below.

TDL - Translation Definition Language

A language used to describe the translation from source to target.

TDL Tables - An encoded version of the TDL description which is produced by the TDL Analyzer

SDDL - Stored Data Definition Language

A language used to describe the source and target databases.

Grammar - The structure and rules that describe the syntax of the language.

Reserved Word - A word in the language that has specific semantics associated with it, and hence is unsuitable for use as a user word.

Token - A basic symbol in the language such as a word or punctuation.

The legal syntax of sentences in the TDL is defined by a grammar written in Backus-Naur Form (BNF). This BNF grammar is processed by a grammar analysis program which produces a set of parsing tables. It is this set of tables that drives the TDL Analyzer as it analyzes a TDL description. All of the legal sentences of the language are encoded in these parsing tables.

3.2 Functional Overview

3.2.1 Input/Output

The input/output relationships for the TDL Analyzer are depicted in Figure 3-1.

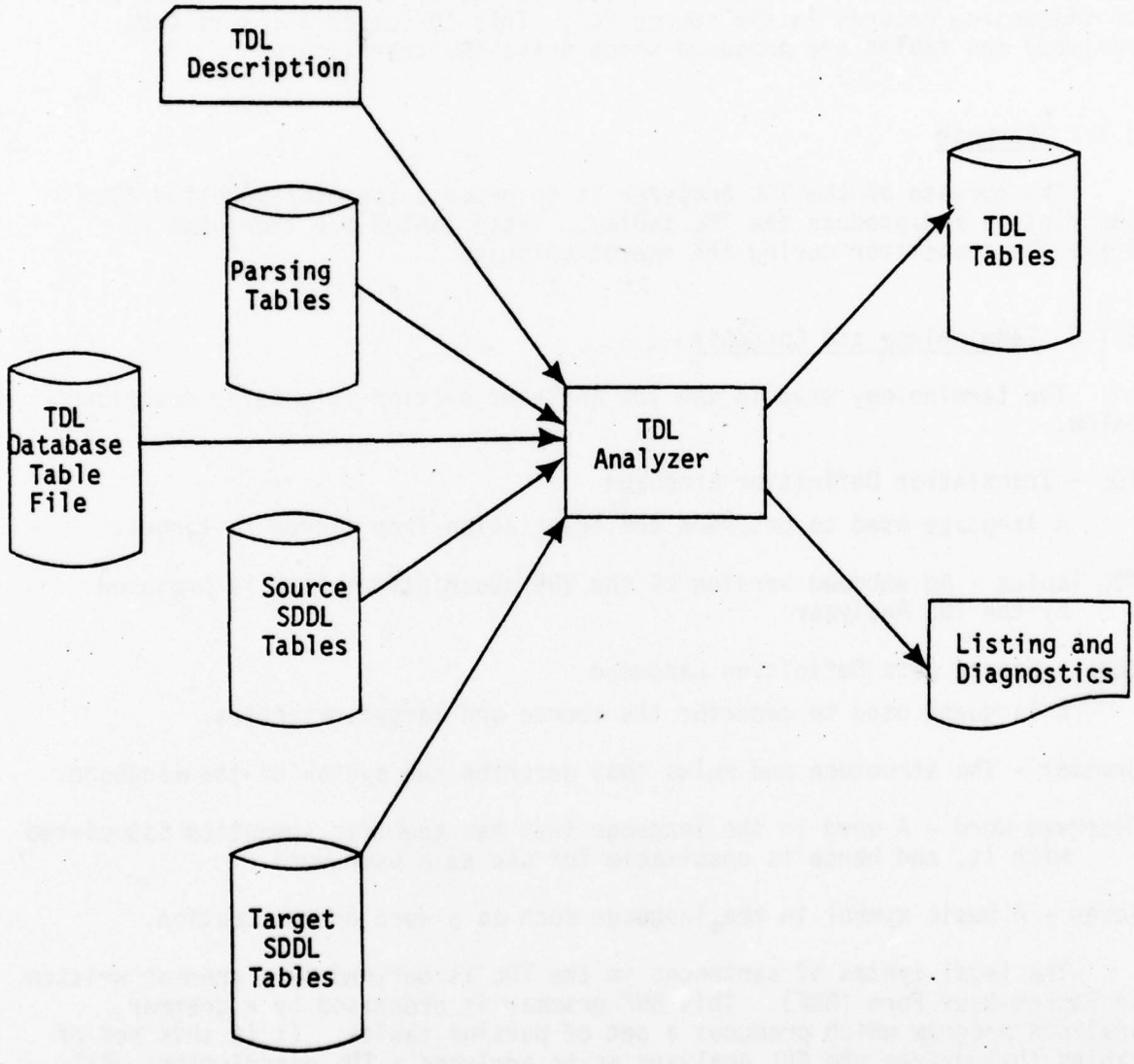


Figure 3-1

The inputs to the TDL Analyzer consist of the user-written TDL description, the parsing tables, source and target SDDL tables, and the TDL tables database table file. The TDL description describes the translation to be performed in

a translator-recognizable form. The parsing tables are a set of tables derived from the TDL grammar rules which define the legal sentence forms. The source and target SDDL tables are output from the IDS Analyzer, and the TDL tables database table file is used to initialize the TDL tables databases which are produced during the TDL Analyzer run.

The output from the TDL Analyzer consists of the TDL tables and a listing of the TDL description. The TDL tables are then input to the translator execution phase. The listing of the TDL description will have error/warning messages where appropriate and a set of timing statistics at the end.

3.2.2 Module Components

Functionally, the TDL Analyzer consists of three main components. The Control, Syntactic, and Semantic modules make up the major components. The Control Component performs initialization and wrapup. The Syntactic Component is responsible for analyzing the syntax of the TDL description, while the Semantic Component analyzes the semantics of the TDL description and constructs the TDL tables. The relationship of these three components is depicted in Figure 3-2, and their functions are described in greater detail in Section 3.3.

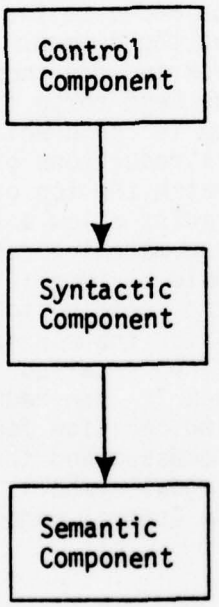


Figure 3-2

3.3 Component Program Logic

The function of the TDL Analyzer is to take as input the parsing tables and a TDL description and produce a set of TDL tables containing the information from the description. The TDL Analyzer consists of three main components as shown in Figure 3-2.

3.3.1 Control Component

The control component performs initialization and wrap-up functions for each Analyzer run. System dependent variables are initialized by a FORTRAN BLOCK DATA section at load time. TDL Grammar dependent variables are set by an initialization routine as the parsing tables are read into storage. After completely processing a TDL description, the Control Component provides wrap-up functions, which include closing all databases and providing timing statistics.

3.3.2 Syntactic Component

The Syntactic Component reads the TDL description as a sequential stream of characters and separates it into the basic symbols of the language called tokens. These tokens can be a user name, a reserved word, an integer, a literal, or a float. The Syntactic Component takes the tokens and the grammar rules encoded in the parsing tables and makes a call to the Semantic Component whenever a phrase of the language is recognized. This process continues until the entire TDL description has been processed.

The Syntactic Component uses the MSP(2:1;1:1) syntactic parser described by McKeeman et.al.[GG1]. The parser is driven by the grammar tables and based on a set of parallel stacks. Each incoming token is examined to determine whether it should be added to the parse stack or not. The decision is made by consulting the grammar tables and considering, at most, the top two tokens on the stack and the incoming token (i.e.,(2,1) context). When the token is stacked, the next token from the input is evaluated in the same manner. When a token is found which is not to be stacked immediately, the list of legal reductions of the grammar is examined to find the reduction(s) which match the top of the stack. If more than one match is found, the grammar tables allow a (1,1) context check. The potential result of each of the matching reductions is considered in relation to the token that would be immediately below it on the stack if the reduction were made and the incoming token were stacked. Illegal combinations are rejected. After the correct reduction has been determined, the appropriate semantic routine is called to process the grammar rule being applied. The parse stack is then reduced and the process is continued with another stacking decision for the incoming token. When all the input tokens have been processed and the parse stack has been reduced to a single token (called the goal symbol), syntactic processing is complete and control is returned to the Control component.

3.3.3 Semantic Component

The Semantic Component builds the TDL tables. The Semantic Component is called by the Syntactic Component with a parameter indicating what reduction is being made, and also which tokens are on top of the stack. There also exist several associated stacks in parallel with the parse stack. They contain information from the description related to the corresponding token on the parse stack. For names and literals, the associated stacks contain the index into free storage of the character string and a flag to indicate whether this is the first time the name has

been used. For integers, the associated stacks contain the values. These values are set by the Syntactic Component when it stacks a token. The Semantic Component manipulates the values in the associated stacks and uses them to build the output tables.

4.0 READER

4.1 Introduction

The Reader is the first module of the Data Translator to access the user's database. It can read any combination of up to five IDS-I, ISP or WWDMS sequential databases. It can be run incrementally and its progress can be controlled through a job status file. The Reader is driven by information stored in the source SDDL tables.

4.1.1 Purpose

The Reader produces the source Restructurer Internal Form (RIF) database from the user's source database(s). The source RIF database is logically equivalent to the sum of the data in all source databases.

4.1.2 Terminology and Concepts

Reader - The Reader is physically represented by two software modules called the Phase 1 and Phase 2 Readers. For each database, the Phase 1 Reader must be run before the Phase 2 Reader can be run.

Source Database - The user's database which will be translated. The Reader can accept three types of source databases: WWDMS sequential, ISP, and IDS.

SRIF - (Source Restructurer Internal Form) - An ADBMS database which is logically equivalent to the source database(s). It is passed along to the Restructurer and Writer modules.

CIRG - (Contained-in-Repeating-Group) - A named collection of items in a record. In COBOL, a CIRG is defined as any field with an OCCURS clause.

Phantom Pointer Relation - A set or relation in an IDS database which is maintained by the user. Set membership is based on IDS reference codes stored in user data fields.

4.2 Functional Overview

4.2.1 Input/Output

The Phase 1 and 2 Readers do not have the same inputs and outputs. This section describes the files for Phase 1 and 2 which are inputs, outputs, or both.

Inputs:

Source Database (PH1 and PH2) - The source database is needed in both Reader phases. Sequential databases may be on tape, ISP databases require the ISP index file, and IDS databases must include all subfiles.

Source SDDL Tables (PH1 and PH2) - The description of the source database is stored in the source SDDL tables. It is an ADBMS database produced by the IDS Analyzer.

Run-time Parameters (PH1 and PH2) - The run-time parameter file contains the name of the database to be read, the number of records to be read, the name of the job status file and how often it should be updated and Keywords specifying whether this will be the first, last, or an intermediate run.

Accessor Object (PH1 and PH2) - The Accessor (IDS, ISP or SEQ) is compiled before every Reader run. This is done so that the IDS structure tables for the database being read will be available for the IDS Accessor. The ISP and Sequential Accessors are also compiled although they never change.

Outputs:

Report (PH1 and PH2) - The Reader produces a report which contains the run-time parameters, initialization information and selected record dump of the SRIF. All error messages are also on this report.

Source RIF (PH1 and PH2) - The primary output of the Reader is the source RIF. It is the logical representation of the source databases and is input to the Restructurer.

Inputs and Outputs:

Deleted Records - The Phase 1 Reader produces a file of SRIF keys of all records which were logically deleted in the source database(s). Those records are deleted from the SRIF at the end of Phase 2.

Internal File (PH1 and PH2) - The Reader's internal file is used to maintain information needed by the Reader between increments of a database and between Phase 1 and Phase 2.

Other Inputs and Outputs:

The Reader uses some other inputs and outputs during only the first run of the Phase 1 Reader. The Work Database used by the SRIF DLL Writer is initialized using the ADBMS Database Tables File (see Section 7.0). The SRIF DDL Writer produces the ADBMS DDL for the SRIF, which is then used by the ADBMS DDL Analyzer to initialize the SRIF file.

4.2.2 Reader Components

Phase 1 Reader

The Phase 1 Reader has six modules (see Figure 4-1).

1. Main Program - Performs major Reader initialization and controls the link overlays.

2. SRIF DDL Writer - The SRIF DDL Writer produces the ADBMS DDL statements for the SRIF using information stored in the source SDDL tables.

3. Table Initializer - The Reader's in-main memory tables are initialized by this module. The internal tables allow the Reader to access data stored in the SDDL tables in less time than going through the SDDL tables.

4. Data Movement - This module transfers the data from the source database to the source RIF.

5. Accessor - There are three Accessors, one each for sequential, ISP, and IDS databases. The Accessor retrieves every record from a database and returns it to the Data Movement module.

6. Wrapup and SRIF Dump - When the Data Movement module is done, the Wrapup module sets up the Reader for the next run and then a few records from the SRIF are dumped.

Phase 2 Reader

The Phase 2 Reader (see Figure 4-2) has five modules of which four (Main Program, Table Initialization, Accessor, and Wrapup/SRIF Dump) are essentially the same as in Phase 1. The only difference is the Relation Linker which links all the records together in the SRIF in the same manner in which they were linked in the source database.

4.3 Component Program Logic

4.3.1 Main Program

The Main Programs for both Reader phases are very similar. They read and analyze the run-time parameters and set up internal variables before calling either the Data Movement or Relation Linker modules. The Main Programs also control the link overlay structure for the Reader.

4.3.2 SRIF DDL Writer

The first module to be executed after the Phase 1 Main Program is the SRIF DDL Writer. It produces the ADBMS DDL for all of the records, items

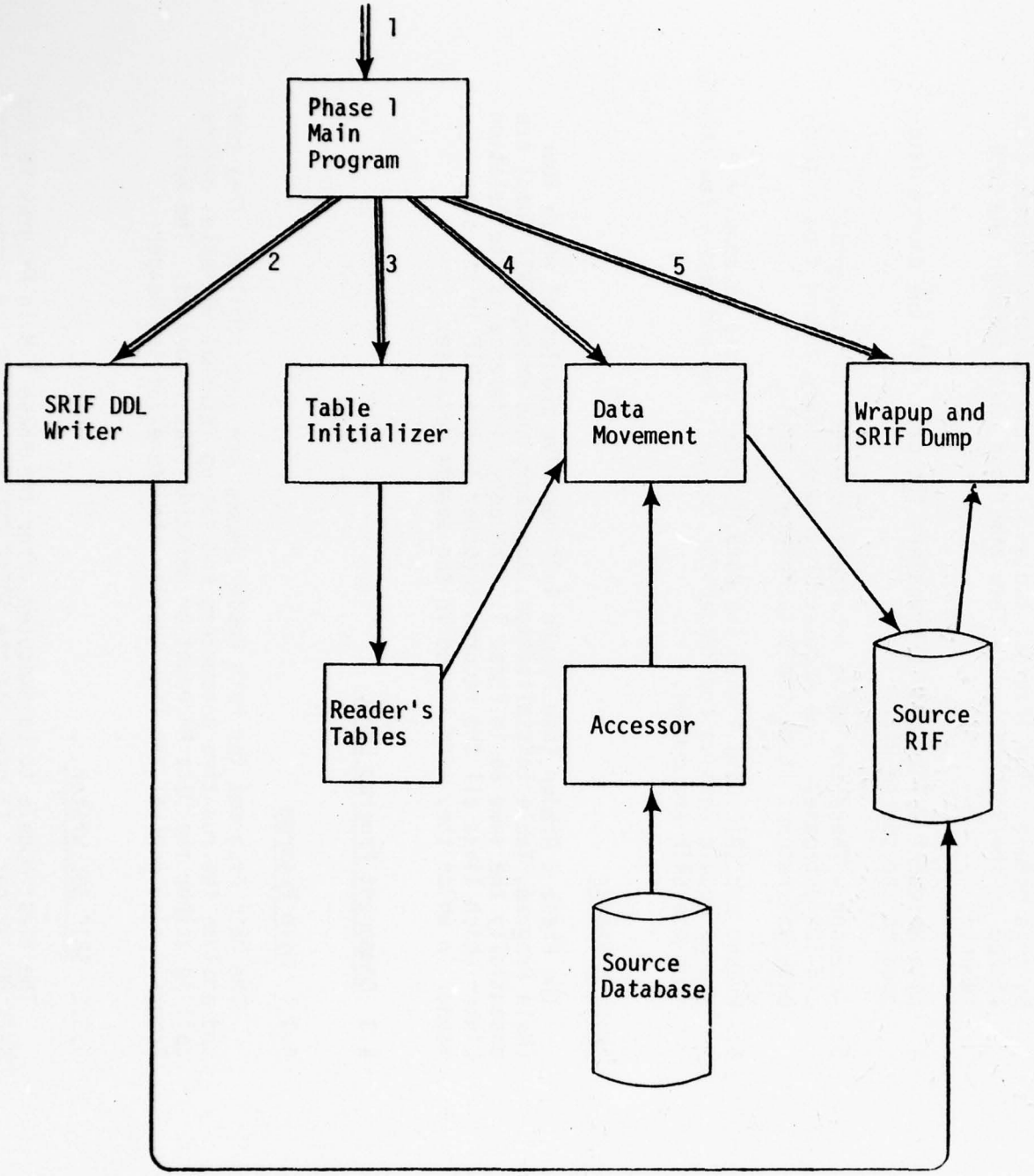


Figure 4-1
Phase 1 Reader Modules

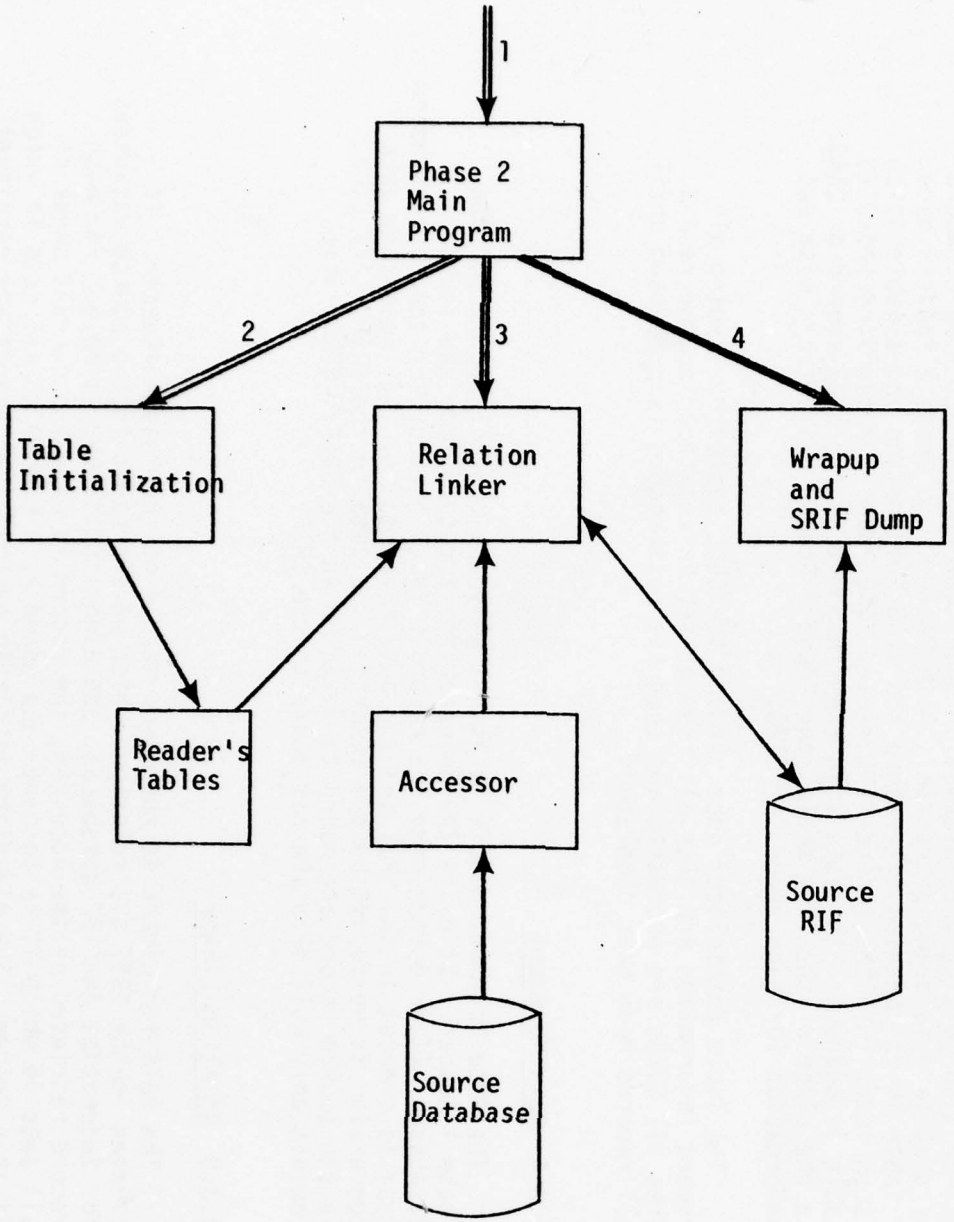


Figure 4-2
Phase 2 Reader Modules

and sets in the source RIF. The records, items and sets in the source RIF correspond to logically equivalent records, fields and chains in the source database. The SRIF DDL Writer is executed only once (during the first Phase 1 Reader run). After the SRIF DDL Writer is finished, the SRIF file is initialized by the ADBMS Database Initializer.

4.3.3 Table Initializer

Processing time for the Reader is minimized by getting all the information the Reader requires out of the source SDDL tables once for all records rather than getting the information about one record each time a record of that type is being processed. Both Phase 1 and Phase 2 Readers have a table which contains all record and CIRG information and a table which is an index into the record-CIRG table. The Phase 1 Readers record-CIRG table is connected to table with item information for each record or CIRG. In the place of the item table, the Phase 2 Reader has a table with set information for each record or CIRG.

The Table Initializer goes through the SDDL tables by finding all record information and then all item or set information for the record. Then all CIRGs are processed and then the next record is processed until all records have been completed.

4.3.4 Data Movement

The Data Movement module is essentially the control module for the Phase 1 Reader. It calls the Accessor for a record from the source database. After determining the record's type, a record of the corresponding type is created in the SRIF and the data moved to it. Any necessary data conversion is performed before inserting the data in the SRIF. Any CIRGs in the source record are moved to records in the SRIF. CIRGs which contain only null data are not moved to the SRIF.

4.3.5 Relation Linker

The Relation Linker is the major module of the Phase 2 Reader. It creates in the SRIF all relations which were present in the source databases. For Sequential and ISP databases, the algorithm is very simple. For each record retrieved by the Accessor, the record is made the current owner of all sets in which it is an owner and added as a member to all sets in which it is a member. The algorithm is simple because the records are retrieved in hierarchical order.

The IDS (network) relation linker is significantly more complex than the ISP/sequential linker. It connects the records in the SRIF together in the same order as they were along the IDS chain NEXT fields. Phantom pointer and match-key relations are also constructed by the Relation Linker.

4.3.6 Accessor

There are three Accessors; one each for sequential, ISP, and IDS databases. The sequential and ISP Accessors are FORTRAN programs which retieve all records in a straight-forward manner (an unformatted read for sequential databases, a call to GETSEQ for ISP databases). The IDS Accessor is an IDS-COBOL-GMAP program which is compiled before every Reader run with the MD section for the database being read. It uses information in the SICT (Slave IDS Control Tables) and the IDS common area .QWRA to get every record in the database using the IDS RETRIEVE DIRECT verb.

4.3.7 Wrapup and SRIF Dump

Both phases of the Reader dump the first and last records of every SYSTEM owned set so data values can be roughly validate. The Reader's internal file is updated so succeeding Reader runs will know where to start in the database (if the Reader was not finished). The ADBMS set and record tables are written back to the SRIF to maintain currency across runs.

5.0 RESTRUCTURER

The Restructurer is the second of the three major modules of the Data Translator. It must be executed after the Reader has completed and before the Writer can begin.

5.1 Introduction

This section defines the purpose of the Restructurer and some of the terminology needed to understand the algorithm explanations.

5.1.1 Purpose

After the Reader has converted the user's source database(s) into a standard physical format known as the source RIF database, the Restructurer is called upon to perform the logical transformations specified by the user's TDL description. The TDL description specifies how the source data is to be transformed into the desired target format. The Restructurer only creates target data records that are different from all source data records; any parts of the source database which will be in the same form in the target database as in the source database are not transformed, but remain in the source RIF database. Target data records created by the Restructurer are stored in another standard physical format, the relational RIF database. The Writer then uses the data from the source and relational RIF databases to write out the user's target database(s).

5.1.2 Terminology and Concepts

This section gives definitions of the terms and concepts used in the descriptions of the Restructurer algorithms.

Access path - A hierarchical substructure of the source RIF database which defines a representation of a target record type in the source RIF. An access path must begin at the SYSTEM record of the source RIF schema.

ADBMS - The internal DBMS used by the Restructurer and other modules of the Data Translator.

ADBMS DBTF - (ADBMS Database Tables File) A sequential file which contains the database tables for an ADBMS database. This file is output by the DDL Analyzer and contains a description of the database schema in tabular format.

ADBMS DDL Statements (ADBMS Data Description Language Statements) - Textual statements used to describe the schema of an ADBMS database. They serve as input to the DDL Analyzer.

Compatible - An access path is said to be "compatible below" a second access path if it shares a subtree, starting at the SYSTEM record, from which the primary key data values are

drawn for the target record represented by the second access path. If this condition holds, it becomes advantageous to process both access paths simultaneously, since the same record instances are used by both of them.

Database Initializer - A program that initializes a file so that it can be used as an ADBMS database.

DDL Analyzer - A language analyzer that accepts as input ADBMS DDL Statements and produces the ADBMS DBTF.

DDL Writer - A program that scans SDDL tables and produces ADBMS DDL Statements describing the database, suitable for input to the DDL Analyzer.

Node - Each record on an access path is referred to as a node of the access path.

Primary key - A collection of data items in an ADBMS record type whose combined values uniquely identify an instance of that record type from all other instances of that record type.

RIF - (Restructurer Internal Form) - The name given to the standard physical format database on which the Restructurer operates. All user databases must be converted to this form before they can be restructured.

Stack - The Stack Builder module takes access paths from the TDL tables and sets up a data structure, similar to a stack, which is used by the Source Accessor to retrieve source record instances from the source RIF database. While somewhat more complex than a simple stack, this data structure will still be referred to as simply a stack throughout this section.

SYSTEM record - Each ADBMS database has one instance of a record type known as SYSTEM placed in it by the Database Initializer. The SYSTEM record defines the "top" of the database schema; entry into the data structure via ADBMS sets (relationships between record types) starts at the SYSTEM record. All access paths must also start at the SYSTEM record.

5.2 Functional Overview

This section describes the inputs and outputs of the Restructurer, along with its major components.

5.2.1 Input/Output

Figure 5-1 illustrates the inputs required and the outputs produced by the Restructurer. A brief explanation of each follows.

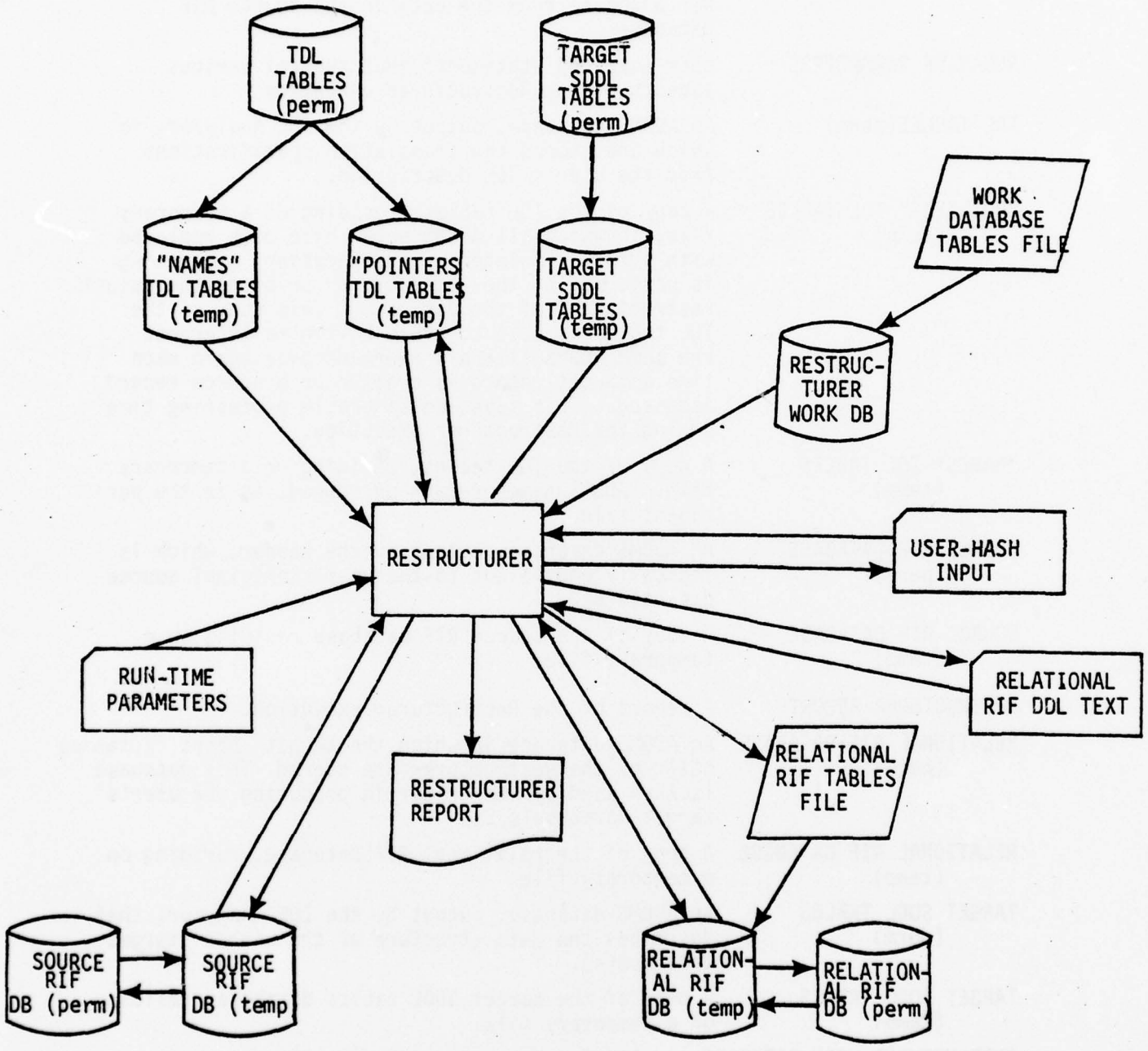


Figure 5-1: Restructurer Inputs and Outputs

RESTRUCTURER	Driven by the translation information in the TDL tables, the Restructurer builds the relational RIF database from the data in the source RIF database.
RUN-TIME PARAMETERS	User-supplied statements that control various aspects of the Restructurer execution.
TDL TABLES(perm)	An ADBMS database, output by the TDL Analyzer, in which are stored the translation specifications from the user's TDL description.
"POINTERS" TDL TABLES (temp)	A copy of the TDL tables, residing on a temporary file, in which all ADBMS names have been replaced with symbolic pointers. The necessary processing is performed by the Restructurer prior to the actual restructuring of the database. This copy of the TDL tables is used to avoid having to interpret the same ADBMS literals over and over again each time a target record is created or a source record accessed. This saves considerable processing time during the Restructurer execution.
"NAMES" TDL TABLES (temp)	A copy of the TDL tables, residing on a temporary file. ADBMS names remain unchanged, as in the permanent file.
SOURCE RIF DATABASE (perm)	An ADBMS database, output by the Reader, which is logically equivalent to the user's original source database(s).
SOURCE RIF DATABASE (temp)	A copy of the source RIF database residing on a temporary file.
RESTRUCTURER REPORT	A report of the Restructurer execution.
RELATIONAL RIF DATABASE (perm)	An ADBMS database in which the target record instances built by the Restructurer are stored. This database is then used by the Writer in producing the user's target database(s).
RELATIONAL RIF DATABASE (temp)	A copy of the relational RIF database, residing on a temporary file.
TARGET SDDL TABLES (perm)	An ADBMS database, output by the IDS Analyzer, that describes the data structure of the user's target database(s).
TARGET SDDL TABLES (temp)	A copy of the target SDDL tables database, residing on a temporary file.
RESTRUCTURER WORK DATABASE	An ADBMS database used by the DDL Writer to produce ADBMS DDL text.
WORK DATABASE TABLES FILE	A sequential file containing the ADBMS DBTF for the Restructurer Work Database. This file is used to initialize the work database before it is used by the DDL Writer.

RELATIONAL RIF DDL TEXT — A sequential file, produced by the DDL Writer, containing ADBMS DDL statements for the relational RIF database. It also serves as input to the DDL Analyzer.

RELATIONAL RIF TABLES FILE — A sequential file, produced by the DDL Analyzer, containing the information from the relational RIF DDL text in tabular format. Also serves as input to the Database Initializer.

USER-HASH INPUT — A sequential file produced by the Restructurer while processsing the run-time parameters. If user-hash statements were given in the run-time parameters, they are rewritten to this file and serve as input to the Database Initializer.

The inputs to and outputs of the Restructurer can by summarized as follows:

Inputs:
1) TDL tables database
2) target SDDL tables database
3) source RIF database
4) relational RIF database file
5) Restructurer Work Database file
6) Work Database tables file
7) run-time parameter statements

Outputs
1) source RIF database
2) relational RIF database
3) Restructurer Work Database
4) Restructurer report

5.2.2 Major Components

The Restructurer is implemented as a main control program and six major components. The relationships between the components are illustrated in Figure 5-2, with the arrows indicating the direction of the flow of execution. A brief explanation of each component is given below.

MAIN CONTROL — Directs the execution of the other Restructurer components; controls most of the link-overlaying of the code; performs general housekeeping functions.

RUN-TIME PARAMETER PROCESSOR — Reads in the user's run-time parameter statements and executes the appropriate initializations, etc.

STACK BUILDER — Retrieves access paths from the TDL tables and sets up a data structure, similar to a stack, which is later used to drive the Source Accessor.

SOURCE ACCESSOR — Driven by the stack produced by the Stack Builder, it retrieves the indicated records from the source RIF database and moves the data into the Restructurer's buffers for subsequent processing.

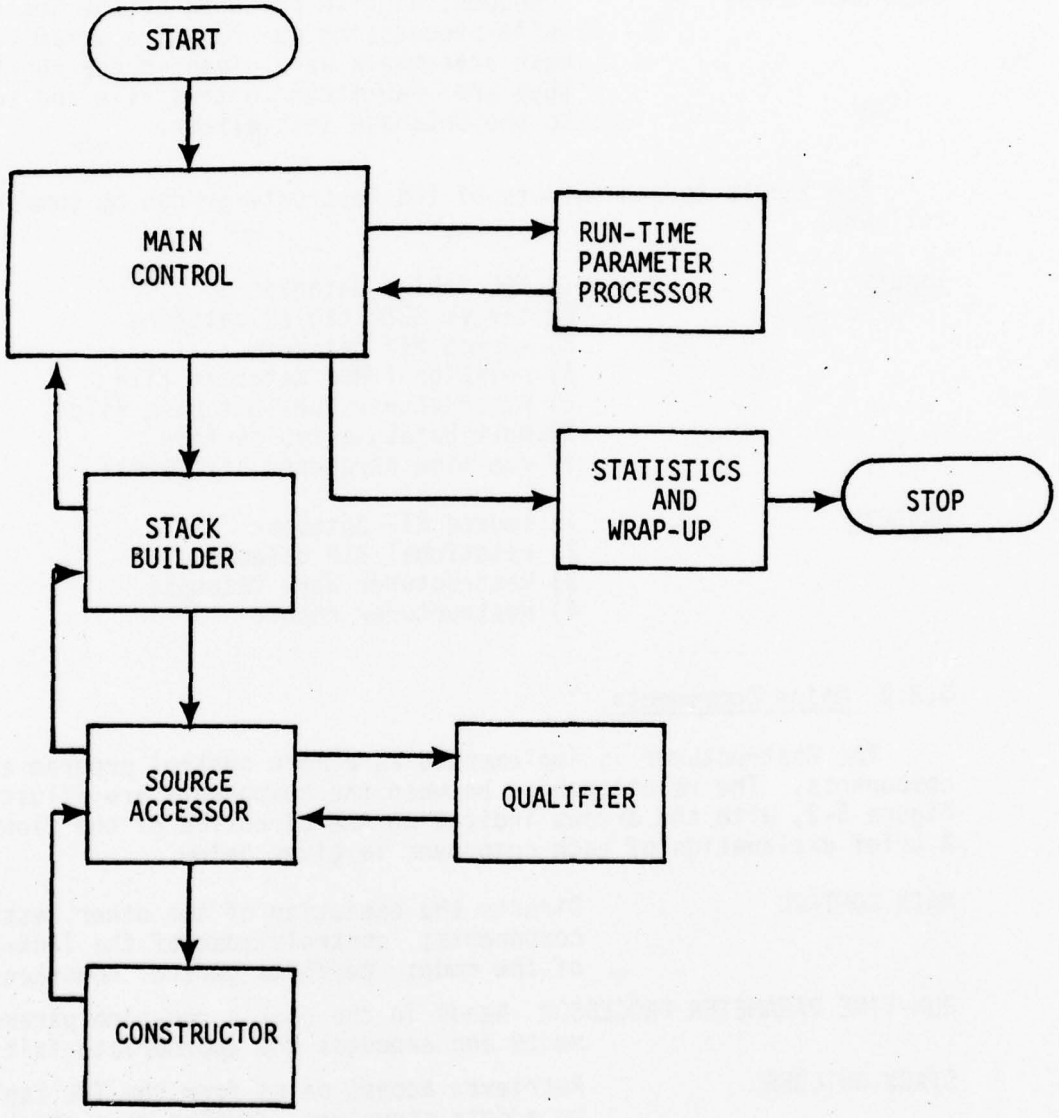


Figure 5-2: Major Components of the Restructurer

QUALIFIER	Decides whether or not each record instance retrieved by the Source Accessor fulfills the requirements specified in the user's TDL description.
CONSTRUCTOR	Retrieves the appropriate data values from the Restructurer buffers, constructs target record instances, and stores them in the relational RIF database.
STATISTICS AND WRAP-UP	Writes out a summary of the Restructurer execution from data accumulated during the run.

The Restructurer execution can be considered to be a two phase process. Phase I is controlled by the Main Control program and the Run-time Parameter Processor, and consists mainly of general housekeeping functions. In general, the Restructurer will always attempt to recover from errors and continue executing in order to find as many errors as possible in each run. However, the Main Control program also has the repsonsibility of terminating Phase I early if a severe error occurs.

The Run-time Parameter Processor is called by the Main Control program. It reads in textual statements, written by the user, which control various aspects of the run: e.g., whether or not the relational RIF database is to be initialized for the first of a series of incremental runs, whether or not debug output is to be produced, etc. When finished, it returns control to the Main Control program, along with a flag indicating whether or not any errors were detected in the run-time parameters. The Main Control program will terminate the run at this point if errors have occurred.

When Phase I has successfully completed, Phase II begins with the Main Control program calling the Stack Builder. The Stack Builder remains in control until either all restructuring has been completed, or a serious error forces premature termination of the run. The Stack Builder is responsibile for setting up a data structure, referred to as a "stack," for groups of compatible access paths from the TDL tables. When a stack has been set up, it is passed to the Source Accessor, which uses it to retrieve record instances from the source RIF database. When all record instances represented by a particular stack have been retrieved, the data structure is discarded, and control returns to the Stack Builder. The Stack Builder then sets up a stack for the next group of access paths, and the cycle continues until all access paths have been used. Control then passes back to the Main Control program.

The Source Accessor, driven by the stack, accesses the source RIF database and retrieves the source records needed to construct the cor-

responding target records. As each source record instance is retrieved, it is passed to the Qualifier. The Qualifier consults the TDL tables and tests the source data to see if it satisfies all (if any) qualification criteria specified in the user's TDL description. If the record instance passes qualification, the Source Accessor continues by retrieving an instance of the next record type on the stack. If the record instances fails one or more qualifications, the next instance of the same record type on the stack is retrieved and the above process is repeated.

When a complete set of record instances for an access path on the stack has been retrieved, the Constructor is called. The appropriate source data values are moved from the Restructurer's buffers into a newly created target record instance; any required data type conversions are also performed at this time. Duplicate record instances are discarded by the Constructor and are not stored in the relational RIF database. WHen all target record instances that can be built using the current source record instances have been constructed, control is returned to the Source Accessor, which then retrieves the next set of source record instances so that new target record instances can be created.

When all source record instances for a stack have been exhausted, the Source Accessor returns to the Stack Builder. Similarly, when all access paths have been used, the Stack Builder returns to the Main Control program, which then calls the Statistics and Wrap-up routines. A report of the execution is written, all files are closed, and the Restructurer terminates.

5.3 Component Program Logic

Section 5.2.2 gave an explanation of the interaction of the various Restructurer components. In this section, an explantation of the processing that occurs within each module is given.

5.3.1 Main Control

The Main Control program oversees the Restructurer execution and controls most of the link overlays that keep the memory requirements to a minimum. It is a simple module that performs the following tasks sequentially:

(1) Sets up the abort wrap-up procedure;
(2) Makes temporary copies of the TDL tables database and the target SDDL tables database;
(3) Initializes ADBMS;
(4) Opens the temporary "Names" and "Pointers" TDL tables databases;
(5) Calls the Run-time Parameter Processor;
(6) Opens the source and relational RIF databases;
(7) Calls the name-to-pointer conversion routine that produces the "Pointers" TDL tables from the "Names" TDL tables;
(8) If no errors have occurred in Phase I, initiates Phase II of the Restructurer execution by calling the Stack Builder;

(9) When the Stack Builder finishes, calls the Statistics and Wrap-up routines; and
(10) If the run was successful, copies the results back to the source and relational RIF database files.

5.3.2 Run-Time Parameter Processor

In order to vary certain parameters at run-time, the user must write Restructurer run-time parameter statements. The options controllable via these statements are:

(1) total vs. incremental Restructurer runs;
(2) production of debug output;
(3) which access paths are to be processed during the run;
(4) labelling the output with a "translation name";
(5) whether or not temporary copies of the source and relational RIF databases should be used, as opposed to operating directly on the permanent database files;
(6) altering the hashing algorithm used to store records in the relational RIF database; and
(7) the use of the Restructurer job-status file and job-status report.

These statements are read and analyzed for consistency by the Run-Time Parameter Processor. Various switches and flags are set, depending upon those options requested via the run-time parameter statements. In particular, if the user requested the use of temporary copies of the source and relational RIF databases, they are prepared now, including initialization of the relational RIF database. A flag is returned to the Main Control program indicating whether any errors were detected while processing the run-time parameters.

5.3.3 Stack Builder

To make the definition of "access path" presented in section 5.1.2 somewhat easier to understand, we can say that an access path is simply a tree structure within the source RIF database schema. Since stacks are very well suited to processing trees, a data structure very similar to a stack is set up by the Stack Builder (using FORTRAN arrays) for use by the Source Accessor.

The Stack Builder retrieves an access path from the TDL tables and all other access paths compatible below it. Since each access path is a tree, and since compatible access paths share subtrees among them, the combined access paths (taking into account those parts that are shared, or "overlap") are also a tree. This larger, combined tree, representing one or more access paths, is used to set up a stack-like data structure that can be used to exhaustively access the source record instances represented by the combined access paths. The modified stack structure contains pointers and flags, indicating each access path of which a source record is a node, for example. Also, each source record type is allocated space in the ADBMS work area; this storage is used to keep a copy of each current source record instance in main memory so that it can be referenced without additional accesses to mass storage.

When the Source Accessor has finished with a stack, the stack storage and internal buffer space are released, and the next stack is set up. If an access path is compatible below more than one other access path, it is only processed once; i.e., it may be skipped over later on when the second access path with which it is compatible is used to start a new stack. When all access paths in the TDL tables have been used in a stack, the Stack Builder returns to the Main Control program.

5.3.4 Source Accessor

The Source Accessor uses the data structure set up by the Stack Builder to drive the record retrieval process. In order to understand the internal logic of this component, the reader must first be familiar with the standard method of processing a tree structure using a stack. While the data structure set up by the Stack Builder is more complicated than a traditional stack, the standard method of processing a tree with a stack can be easily extended to a Restructurer stack having only one access path on it. The idea of compatibility, however, introduces further complexities into the algorithm: in general, each access path on a stack will not include all source record types on the stack. The extension of the algorithm to inculde multiple access paths is not trivial, and is beyond the scope of this document. However, it should be noted that this capability is not vital to the Restructurer's ability to perform restructuring, but simply increases its efficiency.

5.3.5 Qualifier

Each record instance retrieved by the Source Accessor is passed to the Qualifier to determine if it satisfies all the qualifications specified in the user's TDL description. The Qualifier consults the TDL tables and tests all data item values in the source record instance that must be qualified for the access path currently being processed by the Source Accessor. Qualifications are of four basic types:

(1) Comparison of a source data item and a constant value (e.g. data item not equal to zero);
(2) Comparison of a source data item with another data item somewhere on the access path. If the record instance containing the comparison data value has not yet been pushed onto the stack, the qualification is considered successful. It will actually be performed later when the other data item becomes available on the stack;
(3) Acceptance or rejection of a "null record instance": the user can specify that a target record instance may be built even though certain source record instances do not exist. If so specified, a "null instance" is allowed. However, if any other qualifications have been specified, but no source record instance exists, the record fails qualification;
(4) User qualification: the user may supply a FORTRAN subroutine to perform unusual qualifications (e.g., table look-up). Any such routines are dynamically loaded and executed by the Qualifier.

5.3.6 Constructor

When the Constructor is called by the Source Accessor, it is passed a pointer indicating for which access path a target record instance is to be constructed. The Constructor retrieves the needed source data values from its buffers, performs any required conversions, and stores the data in the appropriate target items. The user may also provide a FORTRAN subroutine to perform unusual data conversion functions; any such routines are dynamically loaded and executed by the Constructor. When all target items have been assigned, the record is stored in the relational RIF database; duplicate record instances (i.e., instances whose primary key item values are the same as those of a previously stored record instance) are discarded.

5.3.7 Statistics and Wrap-up

When all restructuring to be performed during the run has been completed, the Main Control program calls the Statistics and Wrap-up routines to write out a report of the execution. During the run, several subroutines are used to keep track of various types of information, such as the number of target record instances of each type that were created. This data is written out as part of the Restructurer report in a user-friendly tabular format. Also, two record instances of each type stored in the relational RIF database are written out to allow the user to examine the data for possible errors.

6.0 WRITER PROGRAM LOGIC

The final module of the translation process is the Writer. Its function is to produce the target database for subsequent normal use by the database administrator who initiated the whole translation process.

6.1 Introduction

This section describes the features of the Writer available to the user and defines some terms which will be referenced later on in Section 6.

6.1.1 Purpose

The most important role of the Writer is to produce a target database according to the user's specifications. Two types of databases may be produced; IDS/1 and sequential (unsorted) file. The sequential file is subsequently sorted by SORT/MERGE to yield a standard WWDMS sequential database, or if ISP is desired, the utility XUTIL is loaded to generate an ISP database.

In producing a target database, two goals must be met by the Writer. These are:

1. Preserve the logical relationships present in the internal form databases produced by the Reader and Restructurer. All of the transformations specified in the TDL between the SRIF and RRIF must have their effect remain intact when the user's target database is written.
2. For IDS databases, the target database must, of course, be a legal IDS file, but perhaps equally important, the Writer will not produce (or propagate) poor database design techniques; specifically, contained-in-repeating groups, phantom pointers/ chains and match-key relations are not allowed in a target database.

Some of the more detailed Writer features are listed below:

1. Because target databases can be enormous, it is not always possible to write an entire database in one block of computer time. Hence, the target database can be written incrementally, each run can be broken off at the completion of a certain number of user-specified record types.
2. The Data Translator internal form databases can hold up to five user target databases which may be of differing types. Although the Writer can only output one database at a time, it has the ability to select any one of the databases within the internal form (RIFs) based on user selection.
3. A debug feature is provided which enables the user to obtain a snapshot look at every record instance written. This facilitates the checking of the TDL without resorting to WWDMS or application programs.

6.1.2 Terminology and Concepts

The following is a formal list of most of the major terms used in the Program Logic Manual in reference to the Writer.

Writing - A verb used to indicate the process of transferring records from the TRIF (see below) to the target (user) database.

TRIF - Target RIF, a logical concept composed of two physical databases, the SRIF and RRIF. As mentioned in Section 5.0, if records in the SRIF are unchanged in their final target database form, there is no need to restructure them. Hence, the Writer will retrieve instances of changed records from the RRIF and unchanged records from the SRIF.

ASDDL - Aggregate Schema Data Definition Language. A special DDL, created by the DWTR (see below) which is a super-set of the DDLs that individually describe the SRIF and RRIF. It serves as the link between the user target record, item and set names and the RIF record, item and set names. An example is given in Figure 6-1.

```
<  RECORD ADMINISTRATION IS ADMINI IN RRIF ;
<  ITEM
<  ADMINISTRATION-NUMBER
<  IS ADMINI ;
<  ITEM
<  MONTH-INAUG
<  IS MONTH- ;
<  ITEM
<  DAY-INAUG
<  IS DAY-IN ;
<  ITEM
<  YEAR-INAUG
<  IS YEAR-I ;
<
<  SET SERVED-WITH-CONGRESS IS SERVED IN SRIF
<  OWNER IS PRESIDENT
<  MEMBER IS RELATOR-PRES-CONG ;
```

Figure 6-1
Sample ASDDL

ASDDLA - Aggregate Schema DDL Analyzer. A program that accepts input ASDDL statements and produces a set of tables used by the Aggregate Schema Processor (ASP, see below). The tables contain all the information needed by the Writer to correctly locate in the SRIF or RRIF the record and set instances needed in the target database.

ASP - Aggregate Schema Processor, a collection of routines which logically sit on top of ADBMS routines. The Writer uses ASP routines to retrieve record instances, item values or to traverse sets. ASP in turn determines which ADBMS calls must be made (remember that the SRIF and RRIF are ADBMS databases) in order to satisfy the requests.

DWTR - DDL Writer, Target RIF. This component writes the ASDDL statements.

Direct Reader to Writer - A mode of translation in which the Restructurer and TDL play no part. It is used for simple database reformatting jobs such as re-calcing of records, changing the page size, deleting items. Complete rules for Direct Reader to Writer translations can be found in Sections 2.6 and 5.5.3 in the Version IIB release 1 User Manual.

IDS Structure Table - A common area (.IDS..) produced by the IDS Translator ($IDS) from the target IDS MD section. It is modified at runtime to "trick" IDS into processing as the Writer wishes.

Masterless records - For IDS databases only, the writing process is broken down into two steps, writing record types which are details of no chains (except CALC chains) and writing out all remaining record types. The former class of records is known as masterless records.

6.2 Functional Overview

This section describes the input and output files to the entire Writer as well as the major components of the "Writer" (e.g., the object file of code written by the University of Michigan).

6.2.1 Inputs and Outputs

There are three modes to the Writer and one wrapup mode for ISP/sequential target databases. Each is defined below with a reference to the accompanying figure.

1. IDS target, Reader-Restructurer-Writer translation (Figure 6-2). This is the regular mode for target IDS databases; e.g., record or relation instances were changed substantially enough to warrant using TDL.
2. IDS target, Reader-Writer translation (Figure 6-3). Used for Direct Reader to Writer translations, it is the same as #1 except for input files which would have been created by the Restructurer or TDL Analyzer and hence do not exist in this configuration.

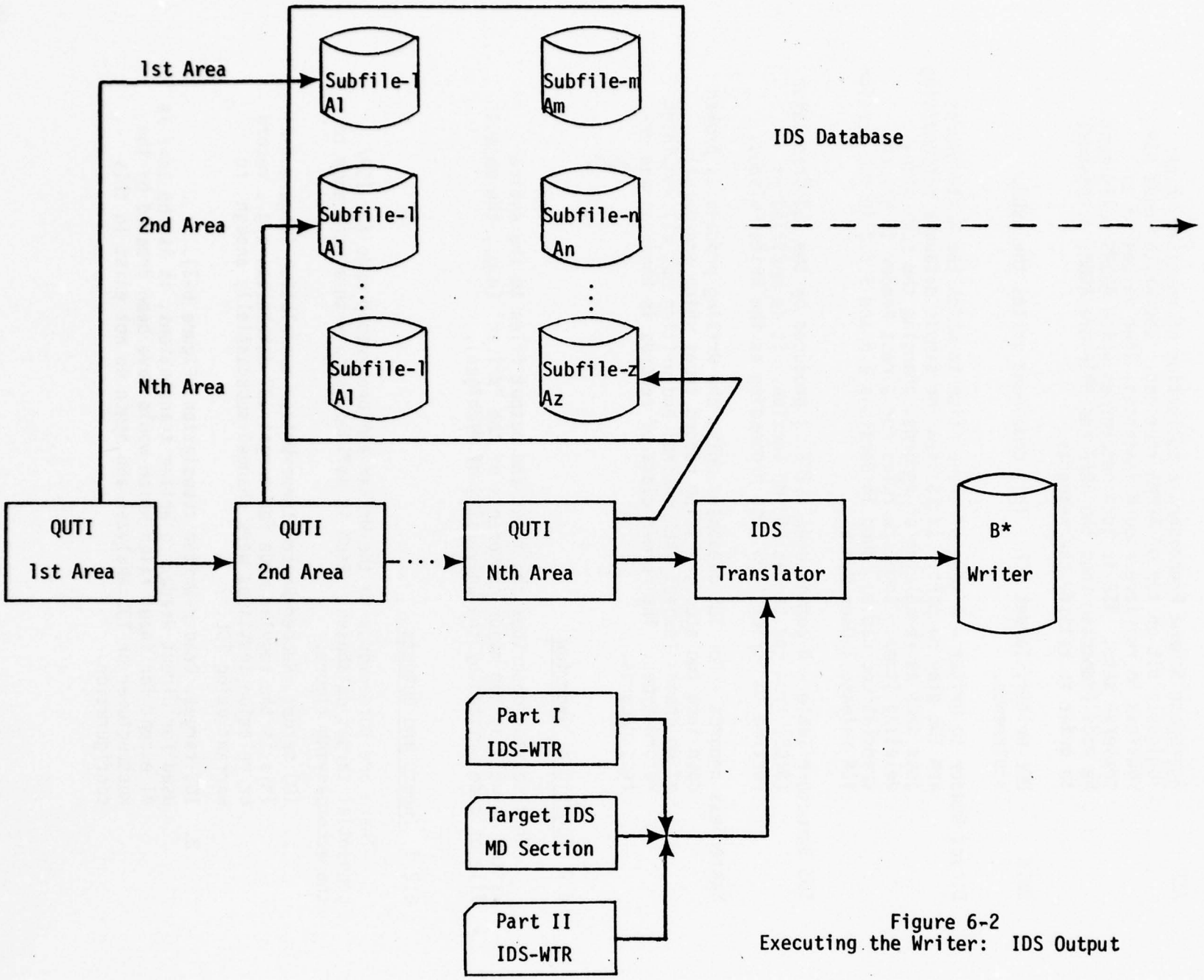


Figure 6-2
Executing the Writer: IDS Output

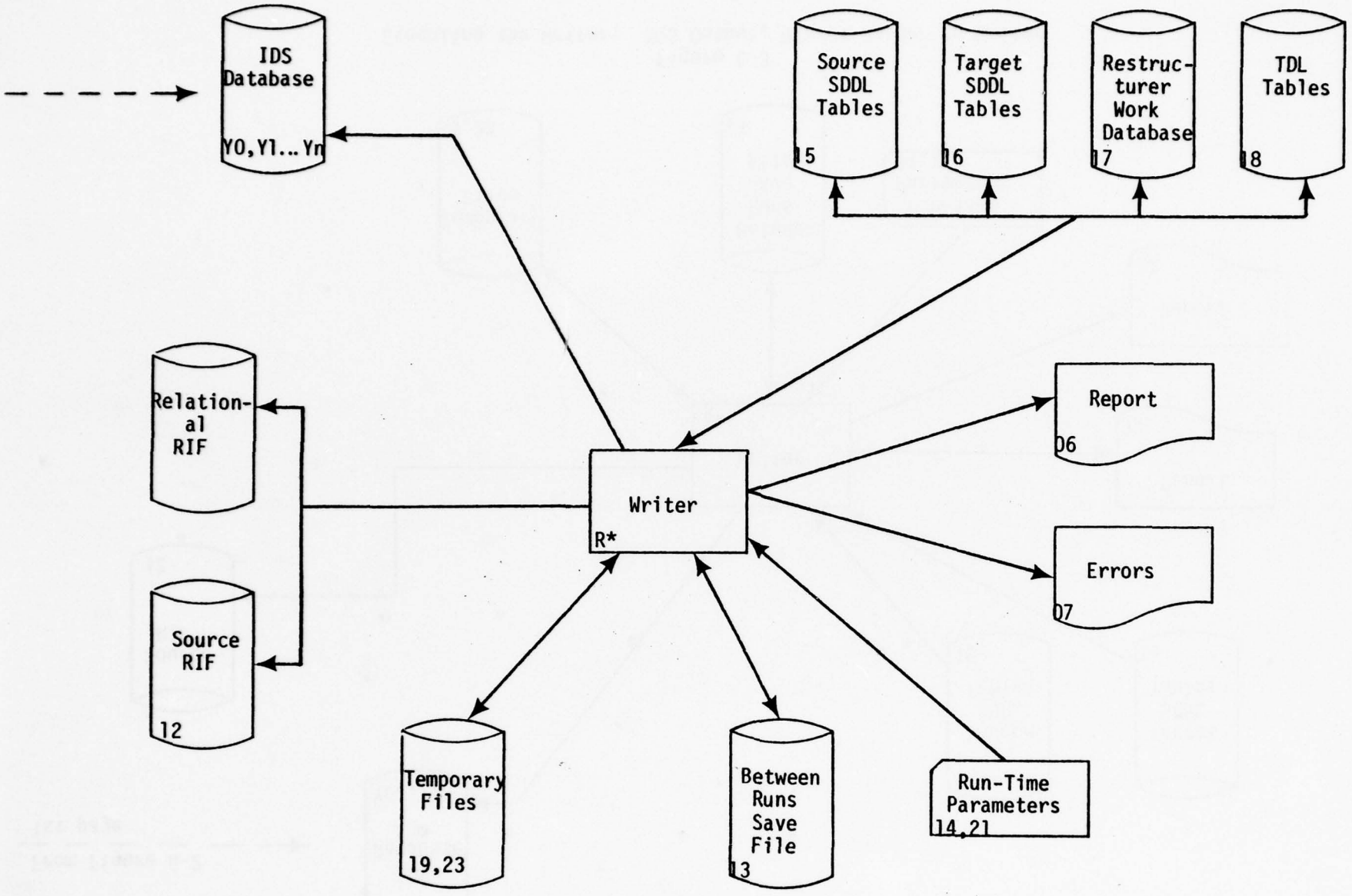


Figure 6-2 (cont'd)
Executing the Writer: IDS Output

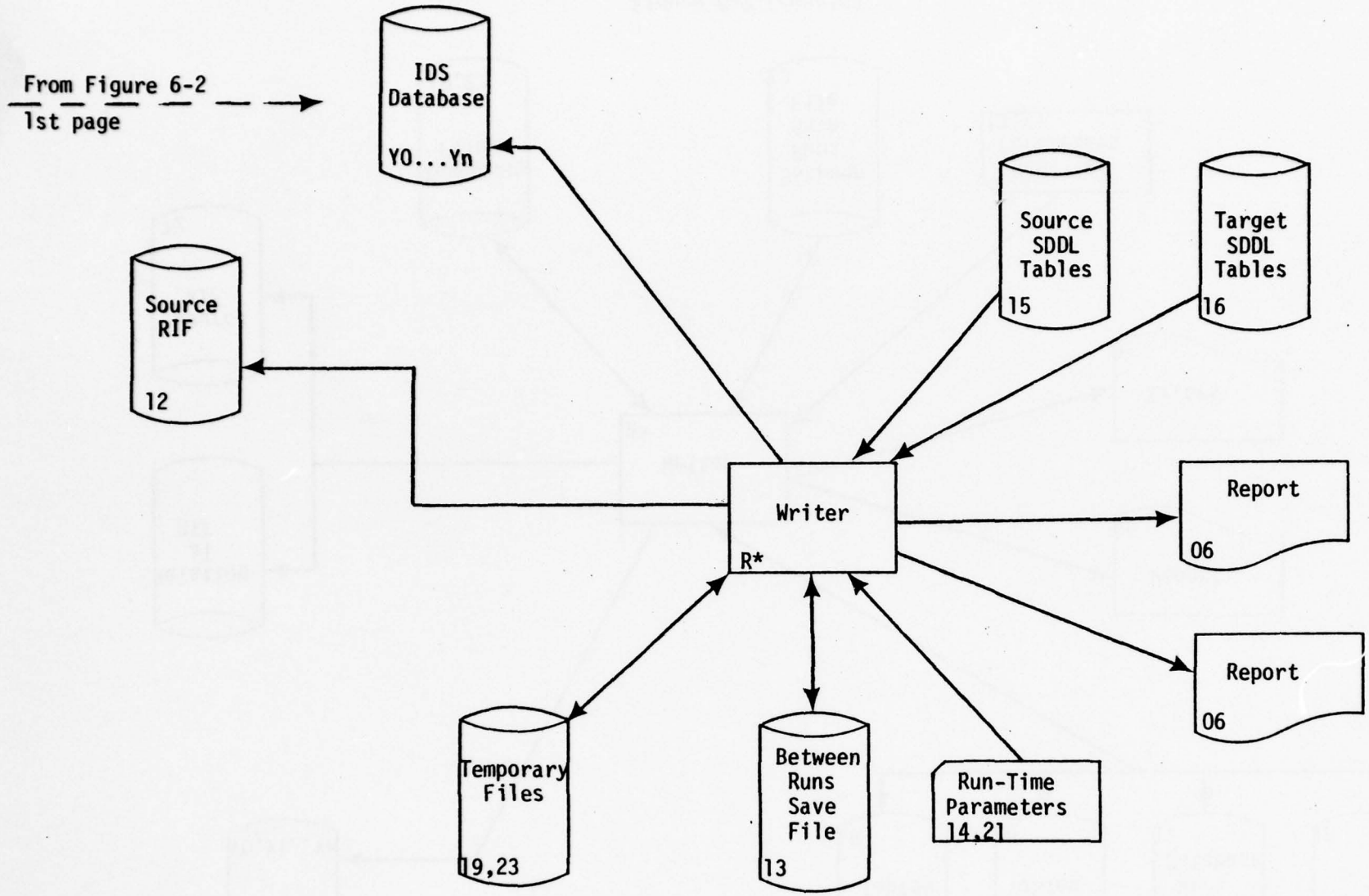


Figure 6-3
Executing the Writer: IDS Output, Direct Reader to Writer

3. ISP/sequential target, Reader-Restructurer-Writer translation (Figure 6-4).
   Regardless of whether or not the user wishes an ISP or sequential target database, the Writer will output an unsorted sequential file. It is the user's responsibility to ensure that record instances have the correct sort key values via the TDL (and probably user routines).

4. Sorting (and ISP only, loading an ISP file) the sequential output (Figure 6-5).
   For sequential output, the file must be sorted. This may then be used to load an ISP database. Although the programs used to perform these functions are Honeywell utilities, the Data Translator provides the control cards to run the utilities.

The major components shown in Figures 6-2, 6-3, 6-4, and 6-5 are explained in detail below.

IDS Database - The target file(s). It may consist of multiple areas, or multiple subfiles. Created by the user with the desired page ranges, page sizes, inventory, etc.

QUTI - The first activity of the writing process. For the initial incremental run only, the IDS database must be initialized by the QUTI activity. A separate activity is required for each area.

IDS Translator - The COBOL-IDS compiler, it compiles the target IDS MD section plus a skeleton COBOL-IDS program into a B* file which is loaded into the next activity.

Target IDS MD section - Written by the user, it is the DDL for the target database. All records, items and chains are defined exactly as the user wishes them to be in the database. Physical design parameters chosen in the target MD section are basically irrevocable once the Writer has executed.

Part I & II IDS-WTR - COBOL-IDS source code for the Writer. It is a skeleton only with no executable code and is provided solely to compile the IDS MD section with. Part I is the IDENTIFICATION, ENVIRONMENT and DATA divisions, Part II is the PROCEDURE division.

Writer - An R* file, it contains the Writer object code which is assisted by the Translator library (not shown). Its components are described in detail in succeeding sections.

Relational RIF (RRIF) - Output ADBMS database from the Restructurer. It contains all "changed" record instances. This file is not present for Direct Reader to Writer translations.

Source RIF (SRIF) - Output ADBMS database from the Reader. All source database(s) record instances are present here.

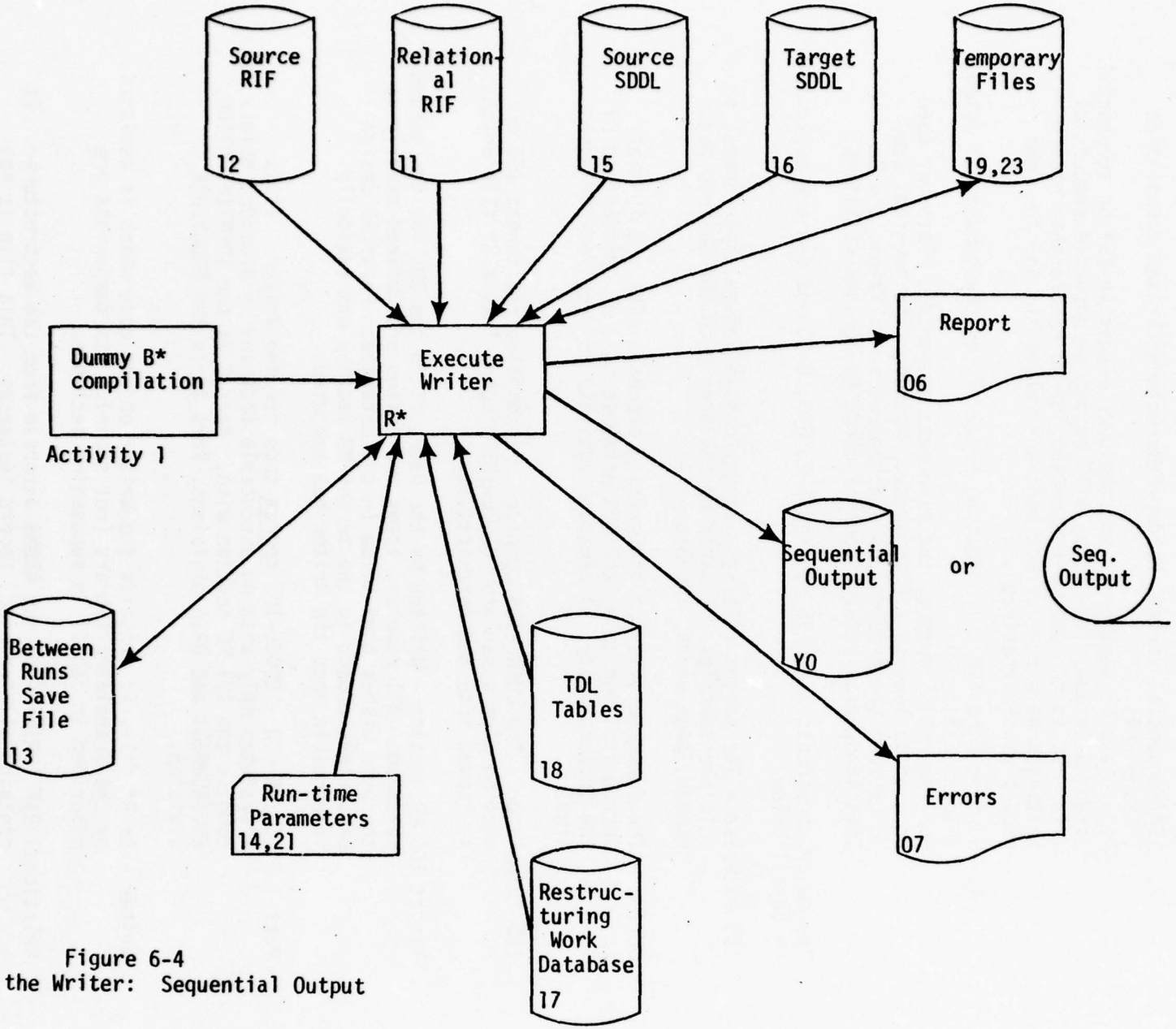


Figure 6-4
Executing the Writer: Sequential Output

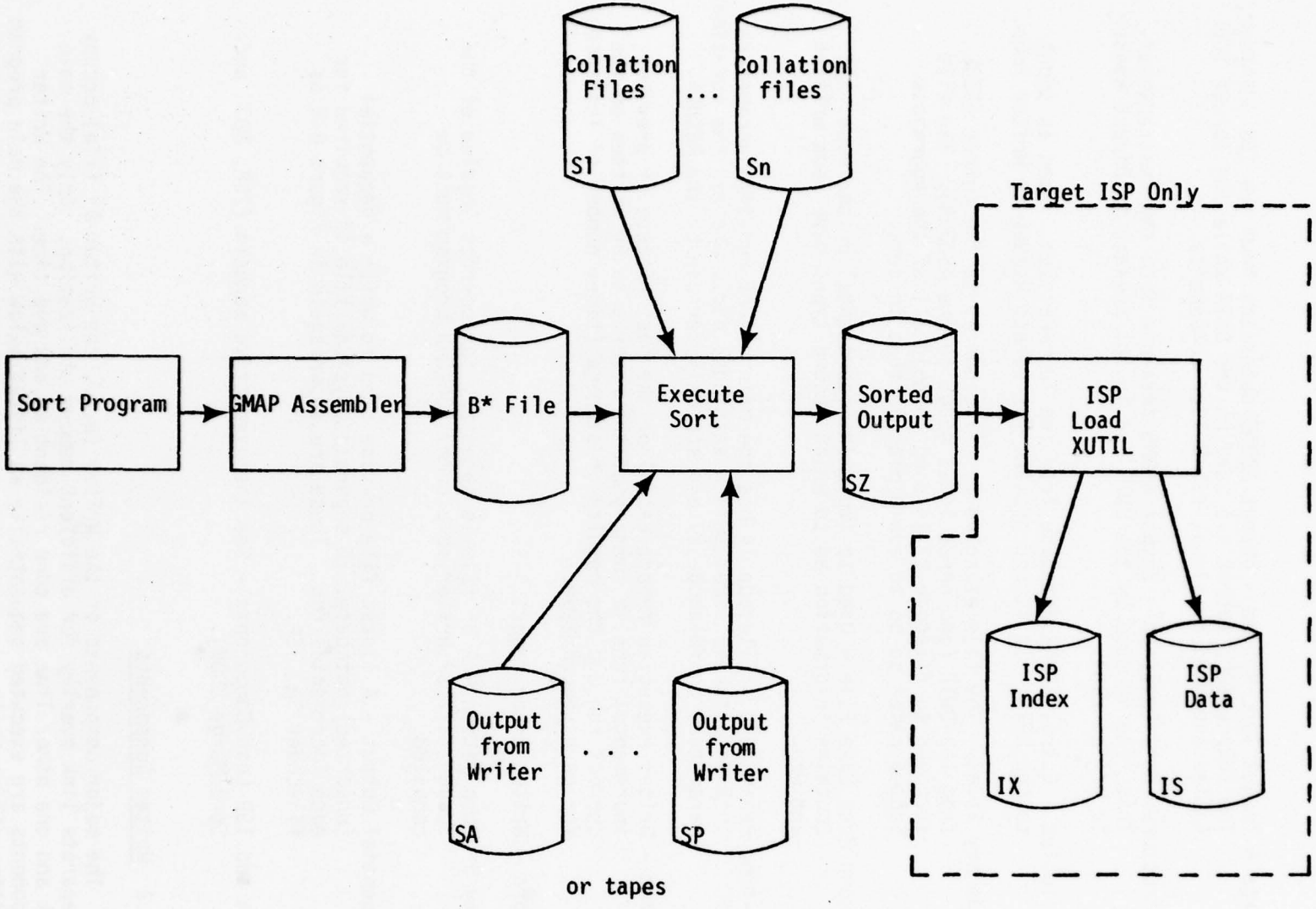


Figure 6-5
Sorting Sequential Writer Output plus ISP Load

Source & Target SDDL Tables - Output ADBMS databases from the IDS Analyzer. The Source SDDL tables are used by the DWTR while the Target SDDL tables are used by almost all Writer components.

Restructurer Work Database - Output ADBMS database from the Restructurer. This file is used by the DWTR and is not present in Direct Reader to Writer runs.

TDL Tables - Output ADBMS database from the TDL Analyzer. Used as input to the DWTR. It is not present for Direct Reader to Writer runs.

Temporary Files - The file attached to filecode 19 is the output ASDDL from the DWTR (and hence input ASDDL to the ASDDLA). The file attached to filecode 23 is used to hold all of the Aggregate Schema names so as to avoid placing them in core.

Between Runs Save File - Used to implement incremental Writer runs. It contains information as to which record types have been written and when.

Run-time Parameters - Filecode 14 has the main set of run-time parameters such as database name, switch settings, etc., whereas the run-time parameters on filecode 21 are strictly for use by the ASDDLA.

Report - Writer execution report detailing the past history of previous incremental runs, a summary of the records to be written on the current run and the results of storage (time, number of instances) for each record type.

Errors - Writer error report.

Dummy B* Compilation - A B* file is necessary for correct loading of the Sequential/ISP Writer so a dummy FORTRAN program must be compiled.

Sequential Output - A linked file or tape used to write a sequential (unsorted) database. A separate physical file is required for each incremental run. These are shown again in Figure 6-5 as filecodes SA...SP.

Sort and ISP Load Components - See the respective manuals (ISP, DD38 and Sort/Merge DD09).

6.2.2 Writer Components

The major components of the Writer (e.g., the Writer R* file) occupy a separate link overlay for efficient memory utilization. Only the main link and one other link are core resident at any one time. The Writer components are executed sequentially as listed below with the main program handling all control.

Main Program - Controls all link overlays. Decides whether or not to continue when errors occur.

Link Overlays

1. SETUP - Reads some of the runtime parameters.

2. DWTR - Writes the ASDDL.

3. ASDDLA - Analyzes the ASDDL to produce ASP tables.

4. INIT1 - Opens all ADBMS databases and locates the correct database.

5. INIT2 - Opens the target file(s), initializes the data structures, handles setup for incremental writing and modifies the IDS Structure Table.

6. PHASE1 - Writes out all masterless records (IDS) or all record types (seq/ISP).

7. PHASE2 - Writes out remaining record types (IDS).

8. PHASE3 - Closes all files and writes a new copy of the between runs save file.

Each component is described in further detail in Section 6.3.

6.3 Component Program Logic

This section highlights the algorithms and methods of the Writer. For the case of sequential file output, writing presents no difficulties, the records retrieved from the TRIF are simply passed to the standard GFRC I/O routines. Since the records presumably have the correct sort key values, the order of record instances outputted is irrelevant as they must be subsequently sorted anyway. However, for IDS databases, it is a different problem. The Writer lets IDS do the actual physical storage of record instances. Similarly IDS does all the chain linking. Because IDS is doing the low level work, the Writer must insure that all IDS routines are "guided" into executing as desired. This means that a considerable effort is made by routines that modify the IDS Structure Table dynamically prior to executing the IDS subroutines. Normally, IDS is used in a COBOL program with the record storage areas occupying a section of the object code in the IDS Section. Any reference to an IDS record or chain is made explicitly in the source program. In the Writer, a generalized program usable for any IDS database, specific record names cannot be coded into the source code as their values are not known until run-time. The solution used was to modify the IDS Structure Table so that records are retrieved from a Writer buffer before storage and that currency of chains is set by placing the correct reference codes directly into the IDS Chain tables.

6.3.1 Main

The function of the main program (in reality a set of programs always core resident) is to provide a controlling structure to the entire writing

process. Each link is called into core sequentially replacing the preceding link. After each routine is called and has returned, its return code is checked. The main program attempts to forge ahead with Writer execution despite errors in an attempt to discover as many mistakes as possible in one run. If this is not possible due to the severity of the mistake, the Writer is shut down.

6.3.2 SETUP Link

The SETUP link solely reads the first three lines of Writer runtime parameters which consist of:

a) database name

b) job-status-file information

c) flags determining whether
   - Writer is on a first or subsequent incremental run
   - How much of the database is to be written (ALL or PART)
   - Is it a Direct Reader to Writer run
   - Is the debug output to be produced

6.3.3 DWTR Link

As previously mentioned, the DWTR writes ASDDL which defines where target records, sets, and items are located (e.g., SRIF or RRIF). The DWTR operates in two modes, normal and Direct Reader to Writer. In normal mode, the ASDDL statements are written by first getting a name from the target SDDL tables and then using the TDL tables and Restructurer Work database to ascertain where the construct for this name resides (SRIF or RRIF). In Direct Reader to Writer mode, all records, sets, and items are by default in the SRIF and hence only a correspondence between target names and their SRIF names is needed. All ASDDL statements are written to a temporary file for use by the next link.

6.3.4 ASDDLA Link

The Aggregate Schema DDL Analyzer (ASDDLA) is fully described in Section 9.0 of this manual. The Writer calls the ASDDLA to analyze the ASDDL produced by the DWTR. Upon completion, everything is ready for use by the ASP.

6.3.5 INIT1 Link

Two small functions are performed in this component:

1. The target SDDL tables database is re-opened for new use by the Writer.
2. The database name supplied as a runtime parameter is checked against legal database names in the target SDDL tables.

6.3.6 INIT2 Link Program Logic

All final initialization functions are performed here as detailed below.

a) Target database is opened for writing.

b) The Writer common areas are initialized. This mainly consists of transferring data from the target SDDL tables to an in-core format for quick access.

c) The past history of all prior incremental runs is printed for the user, the information being obtained from the Between Runs Save File. All setup necessary to tell the PHASE1 and PHASE2 links about which record types should be written on the current run is also performed.

d) The IDS Structure Table is modified as follows:
- All pointers to field locations are altered to point to a Writer record image buffer. When IDS stores a record, it follows the pointers to get the field values unaware that it is retrieving data from the same buffer area irrespective of record type.
- All Select Unique Master bits (except for CALC chains) are changed to Select Current Master bits. This trickery is allowed since the Writer will insure before storing a detail record that the currency is properly set, obviating the need for IDS processing of match key fields.

Additionally, the addresses of records within the IDS Structure Table are obtained for quick use in later processing.

6.3.7 PHASE1 Link Program Logic

This component stores all masterless record types for IDS and all record types for seq/ISP. Each record type is completely stored before any instances of other record types are stored. Because each record type is linked together along one SYSTEM-owned set in the TRIF, retrieval is easy. Record instances are built a data item at a time because item conversion may have to be performed. Once the record is constructed, the appropriate routine (IDS = .QSTOR, seq = .GPUT) is called to do the actual storage. For IDS databases only, the reference code of the just-stored record is placed back into the TRIF within the current record instance. This facilitates the setting of currency for storage of detail record types in PHASE 2.

6.3.8 PHASE2 Link Program Logic

Only IDS databases have this component executed. The procedure is to select a record type whose masters have all been previously stored. This restriction is due to the IDS Store verb constraint which requires that if a record instance is going to be stored, all of its master instances must exist in the database. As in PHASE1, a record instance is built a data item at a time. However, one additional substep is performed, the currency setting of all master instances which were previously stored into

the TRIF after they were stored. Using the reference codes, PHASE2 places the values into the IDS Chain tables. The record is then stored via .QSTOR, linking being done automatically by IDS. And, as before, the stored record's reference code is placed back into the TRIF.

6.3.9 PHASE 3 Link

Some final wrapup steps are performed, listed below:

a) A new Between Runs Save File is written with the history of all incremental runs to date.

b) The target database is closed (.QCLOS for IDS, .GCLSE for seq/ISP).

7.0 FRONT END

7.1 Introduction

7.1.1 Purpose

The Front End module is an interactive program intended to increase the user-friendliness of the Data Translator. The Front End provides a means to automatically build control card files for the IDS Analyzer, TDL Analyzer, Reader, Restructurer, and Writer.

The control card files needed for an entire translation may be built in one terminal session or in a series of terminal sessions.

7.1.2 Terminology and Concepts

ADBMS - A Data Base Management System

The DBTG-like database management system used internally by the Front End.

MPCCF - Modified Prototype Control Card Files

The ADBMS database is partially populated with MPCCF. The lines of MPCCF have both lines of control cards that need substitutions and lines that are used purely for control purposes.

7.2 Functional Overview

7.2.1 Input/Output

The Front End receives input from two sources; the ADBMS database, and the user's responses at the terminal. The database contains both MPCCF and other information that is used either in creating files or providing data to the Control Card Drivers. The user is prompted only for information that is actually needed for the given translation.

The output of the Front End consists of one or more control card file(s). The control card files are ready to run without modification, except for conversion from BCD to ASCII. Diagnostics are also produced.

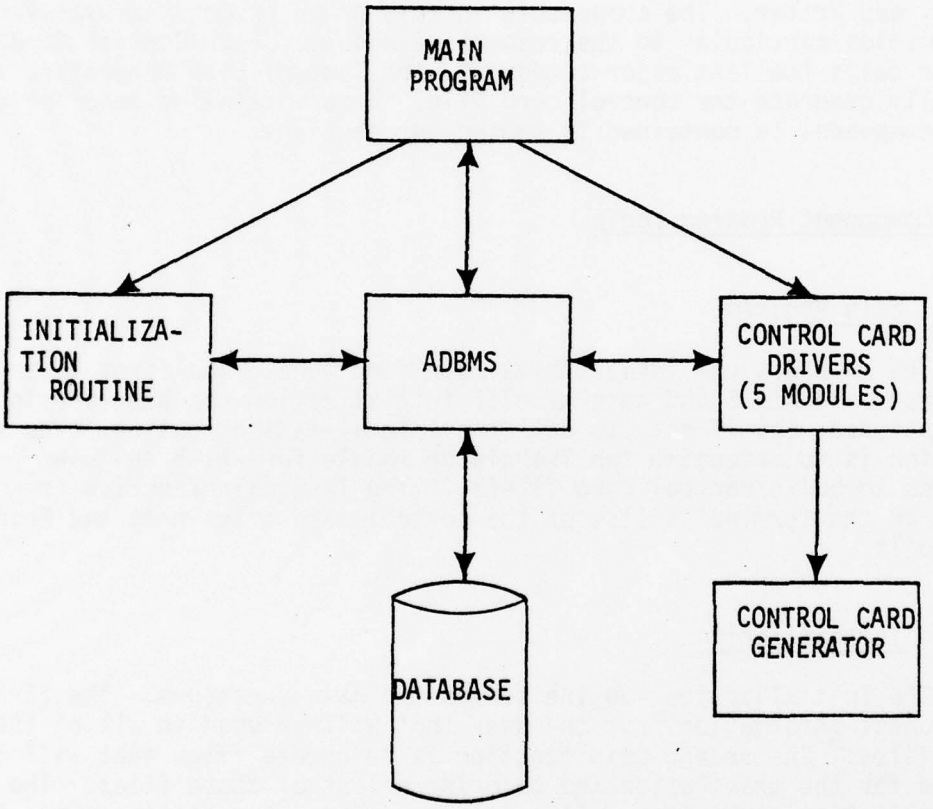


Figure 7-1
Overview of Front End Components

7.2.2 Module Components

The Front End consists of four major logical components; the Main Program, the Initialization routine, a group of Control Card Drivers, and a Control Card Generator.

The Main Program does some bookkeeping and controls the overall flow of the Front End. The Initialization routine is called only once for each Translation. There is a Control Card Driver component for each of the five Translator modules; IDS Analyzer, TDL Analyzer, Reader, Restructurer, and Writer. The components in this group prompt the user for information particular to the respective module. Each Control Card Driver calls the last major component, the Control Card Generator, to actually generate the control card file. A more detailed description of each component is contained in subsequent sections.

7.3 Component Program Logic

7.3.1 Main Program

The Main Program serves three main functions. The first is to interact with ADBMS and determine if initialization has been previously accomplished, and if not, to call the Initialization routine. The second function is to determine the Translator module for which the user desires to build control card file(s). The last main function is to print on the terminal a list of the control card files that the Front End has built.

7.3.2 Initialization

The Initialization routine serves two main functions. The first is to request information from the user that will be used in all of the control card files. The second main function is to create files that will be needed for the translation and to print a list of these files. The Initialization routine is called only once for a translation since the information is stored in the ADBMS database and is used in subsequent terminal sessions.

7.3.3 Control Card Drivers

There are five Control Card Drivers, one for each Translator module, but the functions they perform are basically the same. A Driver puts the information needed by the Control Card Generator into the form it can use, and then calls the Generator to put the control cards into a temporary file. The Driver then creates a permanent file and copies the control cards into it.

As input, the Drivers use both the ADBMS database and user responses to prompts. The output of a Driver is one or more control card file(s).

7.3.4 Control Card Generator

The input to the Generator is provided by the Driver. The MPCCF database contains all of the MPCCFs that can be called (there are approximately 20 different MPCCFs in the database). The call to the generator contains information about which MPCCF to use. After the proper MPCCF is selected, a card-image is read in, and the variables appearing in the line are substituted. Then the card is written to a temporary file.

8.0 ADBMS

8.1 Introduction

ADBMS is a database management system which facilitates the creation, maintenance and accessing of simple and complex data structures. It consists of a collection of FORTRAN-CALLable subroutines whose purpose is to create databases from a user's data structure description, and to serve as an interface between the user and these databases. The following components make up the environment in which ADBMS is used:

a) The database itself, containing the data which is to be accessed and a tabular representation of its schema.

b) The database control system, ADBMS.

c) The user's database access program, containing CALLs to ADBMS routines which access the database.

8.1.1 Purpose

ADBMS is used by the Data Translator as follows:

IDS Analyzer
- a) creates SDDL tables (ADBMS database)
- b) uses Internal Work Database (ADBMS database) internally

TDL Analyzer
- a) retrieves information from source and target SDDL tables (ADBMS databases)
- b) creates TDL tables (ADBMS database)

Reader
- a) uses source SDDL tables (ADBMS database) as input
- b) creates source RIF (ADBMS database)
- c) uses DDL writer work database (ADBMS database)

Restructurer
- a) retrieves data from source RIF (ADBMS database)
- b) stores data in relational RIF (ADBMS database)
- c) accesses TDL tables (ADBMS database)
- d) uses DDL writer work database (ADBMS database)

Writer
- a) uses target SDDL tables (ADBMS database) as input
- b) uses relational RIF (ADBMS hash database) and source RIF as input
- c) uses TDL tables (ADBMS database)

8.1.2 Terminology and Concepts

The terminology and concepts of ADBMS are described below:

Currency - ADBMS currency indicators are used as place markers to keep track of the state of the interface between the user program and the database.

Database - An initialized ADBMS database is a random file consisting of formatted physical pages on which all information in the database is stored. The database tables are stored on the first page(s) of the database. The remaining pages are initialized to a specific format.

Database Key - The unique identifier which distinguishes a record instance from all other record instances in the same database by specifying the page and displacement within that page where the record instance is stored.

DBT (Database Tables) - Tabular form of the logical description of a database according to its DDL. The database tables exist physically in two forms:
1) on the first page(s) of the corresponding initialized database random file, and
2) as a separate sequential database tables file.

DBTF (Database Tables File) - The intermediate sequential form of the database tables, used within the DDLA/DBINT.

DDL (Database Description Language) - A language used to describe the schema of an ADBMS database in terms of records, items, and sets. A specific database description written in this language is also called a DDL.

DDLA/DBINT (DDL Analyzer/Database Initializer) - The utility module used to create ADBMS databases. It analyzes the DDL and if there are no syntax errors or inconsistencies, initializes an ADBMS database according to the DDL description.

Hash Database - A database whose schema includes at least one hash record type.

Hash Input - A series of statements optionally input to the DDLA/DBINT which cause user-specified values to override default values of hashing function parameters.

Hash Record - The storage location of an instance of a hash record type is determined by randomizing its primary key items (defined in the DDL) via a hashing function.

Item - The elementary data unit in the database; used to represent specific data as a number, a string of characters, a logical truth value, etc.

Match-key set - The member-to-owner relationship for a match-key set is established when the user stores the primary key items of the owner of the set in the set significant items of the member. A match-key set instance is defined as an owner record instance and all the member record instances whose set significant items hash to the owner instance.

Multiple Databases - ADBMS has the capability to manage multiple databases simultaneously, i.e., operations can be performed first on one database and then another without closing the first database and opening the second database between operations.

Non-Hash Database - A database whose schema does not allow any hash record types.

Non-Hash Record - Non-hash record instances may be stored either in the next available space in the database, as determined by ADBMS, or in a region specified by the user when the record is created.

Ordered Set - The member-to-owner relationship for an ordered set is established when a member is "added" to the set, an operation requested by the user but implemented and maintained by ADBMS. The sequence of retrieval of member records and the linkage of new member records in the set are controlled by the set ordering criterion (specified in the DDL).

Primary Key - The set of items in a hash record which are hashed to determine the location of the record instance in the database.

Record - A named collection of data items used to represent the major entities of an application.

Schema - The description of the logical structure of a database.

Set - A named collection of record types which specifies an ordering or relation among the records within it.

SYSTEM record - A predefined non-hash record type which is implicitly included in every database schema description.

Work Database - A pseudodatabase used to hold images of individual physical records in core until they are moved back to a database page. Its purpose is to minimize time spent changing pages in and out of core when records on many different pages in the database are being accessed.

8.2 Functional Overview

8.2.1 Input/Output

The input/output relationships for running the DDLA/DBINT are given in Figure 8-1 . The DDL consists of statements which express the schema of an ADBMS database in a form recognizable to the DDLA/DBINT. The hash input consists of statements which cause user-specified values to override default values of hashing function parameters. The DDL and optional hash input are input to the DDLA/DBINT which produces an initialized database file and, optionally, the sequential database tables file.

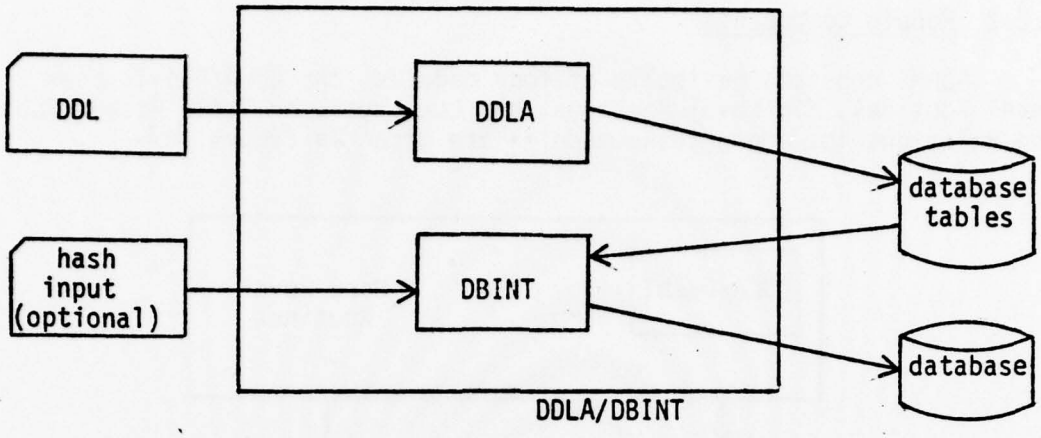


Figure 8-1 DDLA/DBINT

The ADBMS scenario for running a user's database access program is given in Figure 8-2. User database access programs contain CALLs to ADBMS routines which perform the actual database access functions requested.

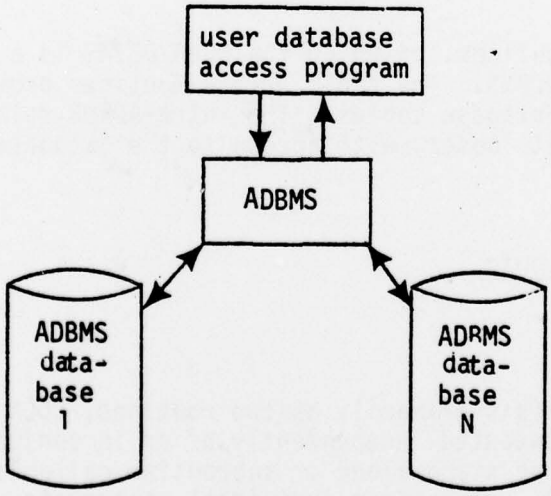


Figure 8-2 ADBMS Scenario

8.2.2 Module Components

ADBMS consists basically of four modules, the DDLA/DBINT, User Level Routines, Mid Level Routines, and Low Level, or Table Access Routines. The relationships among these modules are given in Figure 8-3.

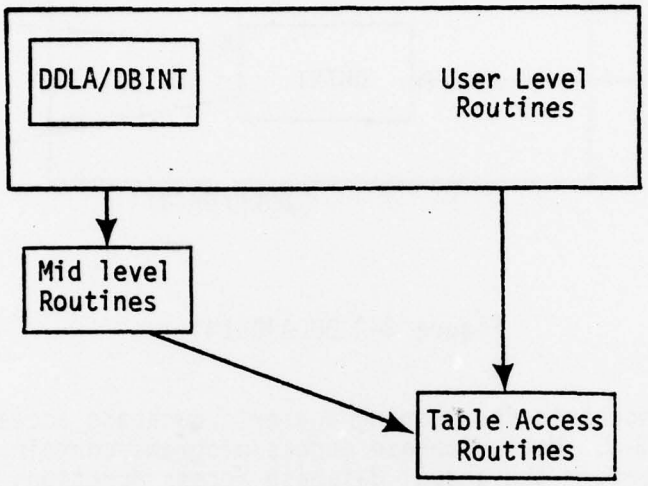


Figure 8-3

The User Level Routines, of which the DDLA/DBINT is a subset, provide the user interface to ADBMS. The table Access Routines provide the interface between ADBMS and the database tables. The intra-ADBMS relationships among the modules are somewhat looser, with access to the database being performed at all levels.

8.3 Component Module Logic

8.3.1 DDLA/DBINT

The DDLA/DBINT exists primarily as two routines, DDLA and DBINT. These routines may be executed independently of or in conjunction with each other, and in either stand-alone or subroutine-callable mode. The DDLA stage reads the DDL, recognizes individual statements, and builds the appropriate control blocks in the database tables to store and maintain the information specified in the DDL. The output of the DDLA module is the DBTF. The DBTF and the optional hash input are the inputs to the DBINT stage of the DDLA/DBINT. The DBINT produces an initialized database ready for the storage and accessing of data.

8.3.2 User Level Routines

The interface between the user and the database in ADBMS is accomplished via CALLS from the user program to a set of ADBMS routines designated as being "user level." Each user level routine corresponds to an accessing function which the user may request to be performed on the database. Each routine translates this high level request into physical input/output operations on the database, using information stored in the database tables and the database itself. In general, the actual interface with the database and database tables is accomplished via the routines in the mid-level and table access modules.

8.3.3 Mid-Level Routines

Mid-Level routines in ADBMS perform such functions as database storage allocation, managing database pages in and out of core as they are accessed, maintaining currency and set ordering as records are accessed and members added to and retrieved from sets, etc. While much of the interface with the actual data in the database is accomplished by the Mid-Level Routines, all access to the database tables is accomplished via Table Access module routines.

8.3.4 Table Access Routines

Each routine in the table access module of ADBMS is responsible for the retrieval or storage of data in one field of the database table control blocks. This isolates the higher level ADBMS routines from any design change in the physical structure of the database tables.

9.0 ASP

9.1 Introduction

The ASP is an extension of ADBMS which, when used in conjunction with ADBMS, allows the user to view an aggregation of subsets of one or more ADBMS databases as one Aggregate Schema (AS) database and thus access multiple databases as one database.

The ASP environment, then, consists of the following elements:

a) several ADBMS databases, each containing the data to be accessed and a tabular representation of its schema

b) ADBMS

c) the AS Database Tables, a tabular representation of the AS Database's logical description.

d) the Aggregate Schema Processor (ASP), consisting of a collection of FORTRAN CALLable subroutines whose purpose is to map AS database access requests into ADBMS database access requests, based on the information stored in the AS database tables.

e) the user's database access program, containing CALLs to ASP routines.

9.1.1 Purpose

The Data Translator uses the ASP in the writer module to facilitate accessing the source RIF and relational RIF ADBMS databases.

9.1.2 Terminology and Concepts

The terminology and concepts of the ASP are described below. Many ADBMS terms and concepts are also fundamental to the ASP, while other terms have different meanings in the two contexts; reference to Section 8.1.2 may be helpful.

AS Database (Aggregate Schema Database) - a logical "view" of one or more physical databases. The term is used in a physical sense to refer to the composite of a) several ADBMS databases, and b) the ASDBT

AS Database Key - the unique identifier which distinguishes an AS record instance from all other record instances in the same AS database by specifying the ADBMS database and the page and displacement within that page where the record instance is stored.

ASDBT (AS Database Tables) - tabular form of the logical description of an AS database according to its ASDDL and ASDNDDL.

ASDDL (AS Data Definition Language) - a language used to describe an AS database view in terms of sets, records, items, and their mappings to corresponding ADBMS database constructs. A specific AS database description written in this language is also called an ASDDL.

ASDDLA (ASDDL Analyzer) - a utility module used to create AS databases. It analyzes the ASDDL and if there are no syntax or semantic errors, produces the ASDBT according to the ASDDL description.

ASDNDDL (AS Database Name DDL) - a series of statements which identify the ADBMS databases involved in an AS database.

Assigned Record - a one-to-one correspondence between the AS record type and a schema (ADBMS) record type, such that the schema record is viewed directly through the AS assigned record type.

Assigned Set - a one-to-one correspondence between the AS set type and a schema (ADBMS) ordered set type.

Coupled Record - a coupling of two schema (ADBMS) record types such that the AS coupled record type is viewed as a merge of the two schema record types.

Currency - the ASP uses ADBMS currency indicators in addition to its own currency indicators to keep track of the state of the interface between the user program and the database.

Item - AS item types defined in an AS record type specify which schema (ADBMS) items in the corresponding schema (ADBMS) records are to be included in the AS database view of the record and how they are to be viewed.

Match-key set - AS match-key set types are parallel to ADBMS match-key set types, but allow the owner and member record types to reside in different ADBMS databases.

Record - AS record types specify which schema (ADBMS) record types are to be included in the AS database view of how they are to be viewed.

Set - AS set types define which schema (ADBMS) sets are to be included in the AS database view and how they are to be viewed. They may also define new match-key sets which are not defined at the schema (ADBMS) level.

Set Significant Items - similar to ADBMS set significant items.

9.2 Functional Overview

9.2.1 Input/Output

The input/output relationships for running the DDLA/DBINT are given in Figure 9-1. The ASDDL consists of statements which describe the Aggregate Schema database view of its underlying ADBMS databases. The ASDNDDL consists of statements which identify each ADBMS database involved in the AS database. The ASDDL inputs both the ASDDL and the ASDNDDL and references the database tables of each of the underlying ADBMS databases to produce the ASDBT as output.

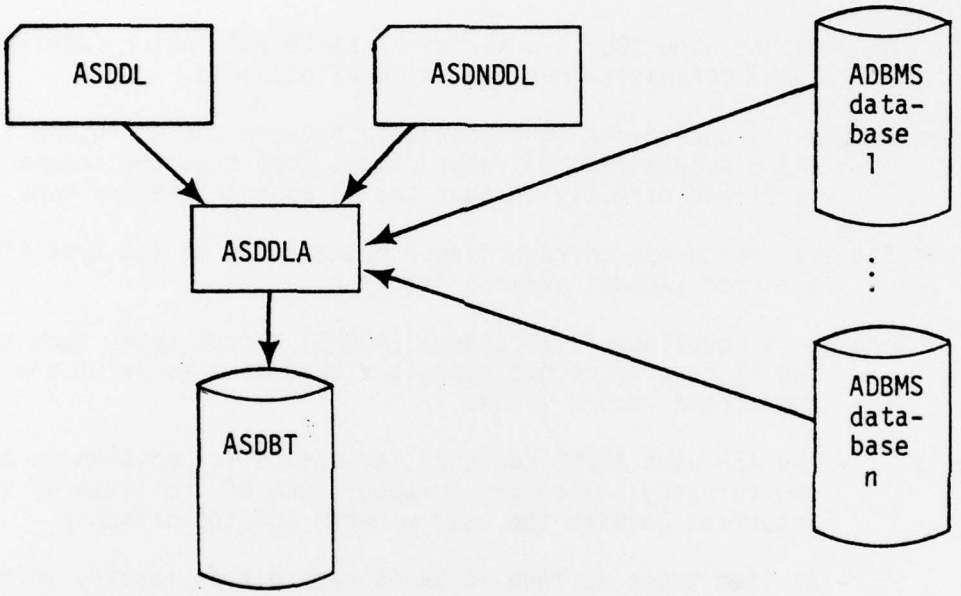


Figure 9-1
The ASDDLA

The scenario for running a user's database access program under the ASP is given in Figure 9-2. User database access programs contain CALLs to ASP routines. These routines use the information stored in the ASDBT, the ASDNDDL, and the individual ADBMS databases themselves to perform the actual database access functions requested.

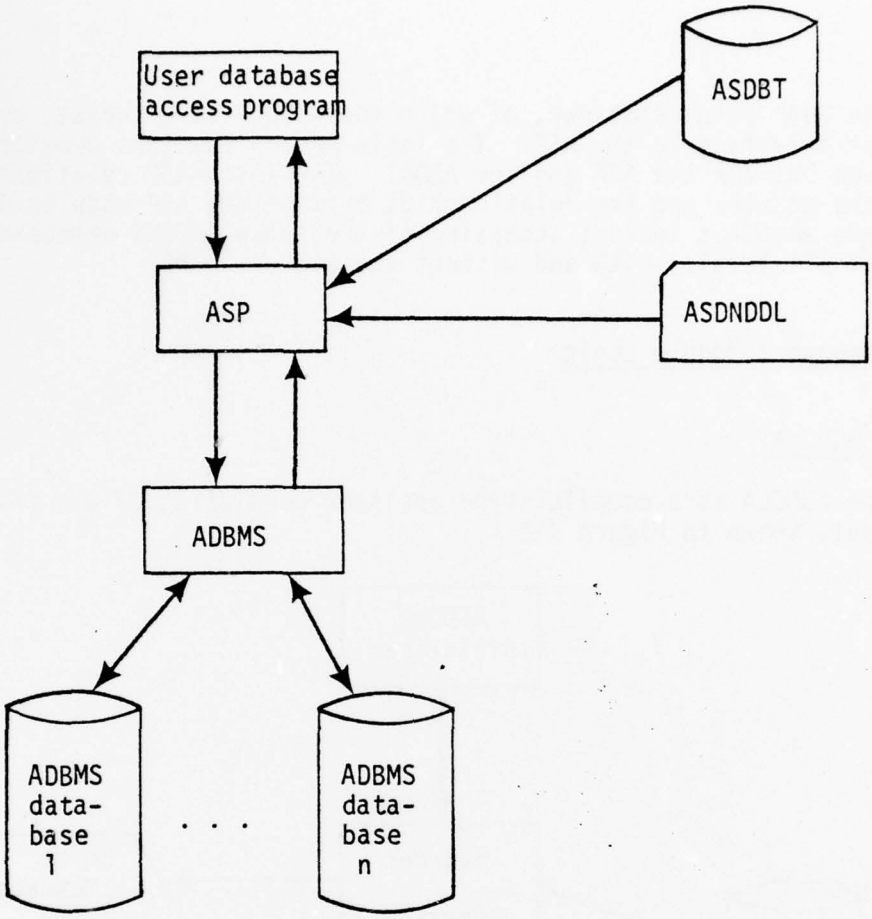


Figure 9-2
ASP Scenario

9.2.2 Module Components

The ASP consists basically of four modules, the ASDDLA, User Level Routines, Mid-Level Routines, and Low Level, or Table Access Routines. The relationships among these modules are given in Figure 9-3.

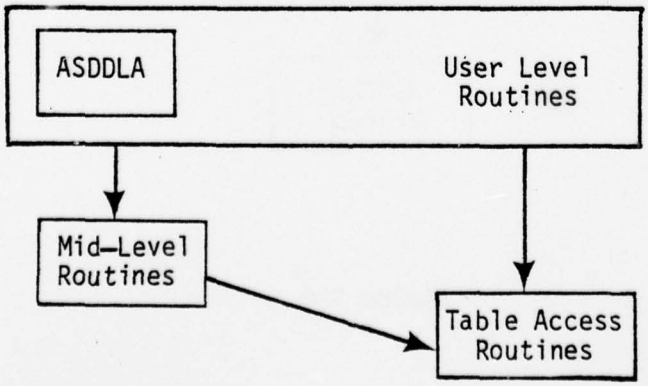


Figure 9-3

The User Level Routines, of which the ASDDLA is a subset, provide the user interface to the ASP. The Table Access Routines provide the interface between the ASP and the ASDBT. The intra-ASP relationships among the modules and the relationships between the ASP modules and ADBMS are somewhat looser; accessing of the actual ADBMS databases takes place at all levels, with and without the aid of ADBMS.

9.3 Component Module Logic

9.3.1 ASDDLA

The ASDDLA is a compiler type analyzer consisting of the five components shown in Figure 9-4.

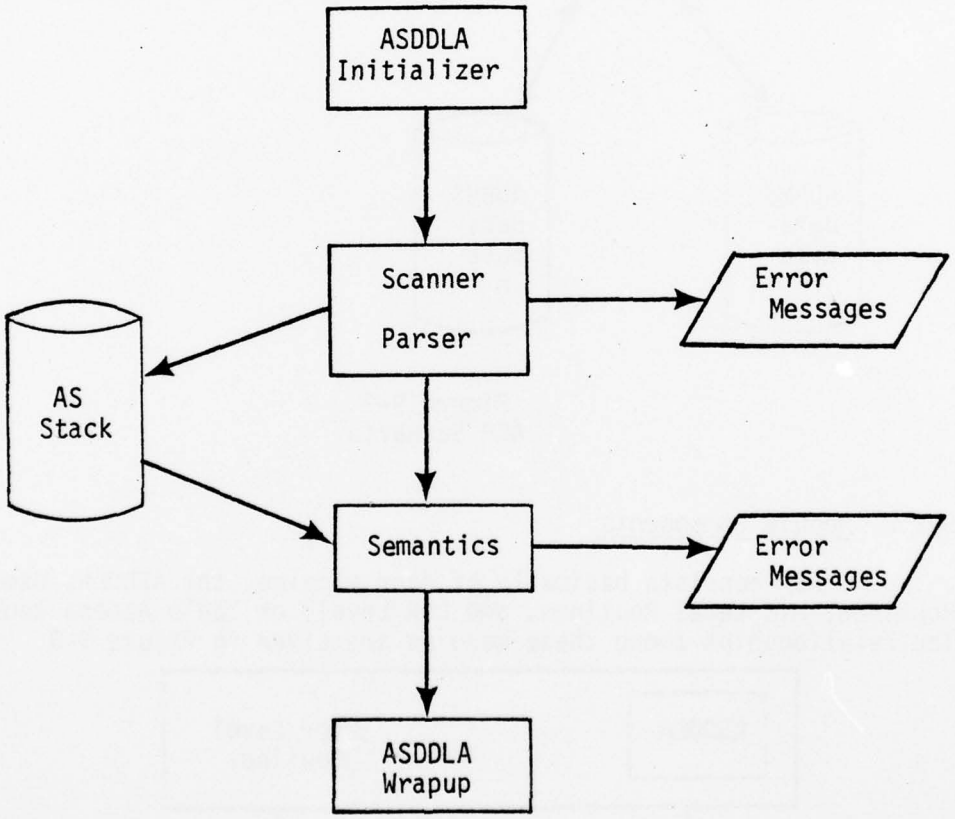


Figure 9-4

The ASDDLA Initializer is the controlling program for all of the ASDDLA modules. It initializes arrays and prepares the scanner for input. The scanner and parser identify ASDDL constructs and resolve the ASDDL into various productions. Once a production has been isolated the appropriate semantic routine is called. The semantic routines build the AS database tables and check for internal consistency. The ASDDLA wrap-up module prints out a review of all table blocks generated and statistics for the analyzer run.

9.3.2 User Level Routines

The interface between the user and the AS database in the ASP is accomplished via CALLs from the user program to a set of ASP routines designated as being "user level." Each user level routine corresponds to an accessing function which the user may request to be performed on the AS database. Each routine translates this high level request into an ADBMS-level accessing request using information stored in the AS and underlying databases themselves. This translated request is performed by ADBMS routines or routines in the mid-level and table access modules of the ASP.

9.3.3 Mid-Level Routines

Mid-Level Routines aid in translating requests from AS terms to ADBMS terms and in the actual execution of these requests.

9.3.4 Table Access Routines

Each routine in the table access module of the ASP is responsible for the retrieval or storage of data in one field of the AS database table control blocks. This isolates the higher level ADBMS routines from any design change in the physical structure of the AS database tables.