THE IMPLEMENTATION OF A SUBSET DATA DICTIONARY VERIFIER

by

JACQUELYN FERN CLINE

B.S., Central State University, Edmond, Oklahoma, 1974

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

Dr. David A. Gustafson

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

CHAPTER 1

THE ROLE OF DATA DICTIONARIES AND A VERIFICATION IMPLEMENTATION

1. Introduction

This paper introduces the concepts of data dictionaries and a verification process called a Subset Data Dictionary Verifier. The Subset Data Dictionary Verifier is developed to verify the frames in a data dictionary in relation to a master data dictionary. Both dictionaries used in the verification process are derived from the Entity Relationship Attribute (ERA) Specification. The verifier will show the discrepancies between the subset data dictionary and the master data dictionary.

1.1 Software Design Crisis

It has been suggested that we are in a software design crisis resulting from cheaper hardware technology and increasing software design costs. "The crisis is created by the labor-intensive nature of design methods..." (1) The hardware costs are decreasing rapidly; in fact, those costs are being cut to the bone. However, hardware only constitutes approximately 30% of the total cost of a system. The software costs, likewise, need to be cut to a minimal figure. No longer can we continue to support "made to order" applications. Technology is continuing to strive towards better firmware and more generalized hardware. Software, on the other hand, is continuing to fall farther and farther behind.

This crisis is ameliorated if the computer technology implements the philosophies and structures of: automated programming and design aids, such as data dictionaries and data base management systems; direct access to computers by end users, such as user-friendly and user-maintainable query systems; and project management techniques and tools for software development projects.

We are quickly approaching the need for knowledge information processing systems (KIPS) using data inference and artificial intelligence. The Japanese are in a fast paced race to complete such a task by 1991. "That means all the vigorous hand cranking, throttling, and wrenching a pioneer now accepts as the inevitable price of using the machine -- the difficult programming languages, the struggle to make different programs compatible, the problems of putting human knowledge into machine form -- are to disappear, eliminated in the new Japanese Fifth Generation of computers." (2)

Each analyst designs systems individually with little regard to the way other analysts design systems; programmers code programs as individually as their personalities and background allow. Technology has been rapidly changing and now we must draw these concepts together into a unified assembly. Company guidelines are issued to analysts and designers to make sure a development effort is flexible, user-friendly, and user-maintainable. Programmers are being given strict guidelines to insure programming products are cohesive, consistent, and understandable. "We have made great gains in the consistency and rigorousness with which we apply these

tools to the development of data processing systems." (3) The age of data processing is truly evolving into data automation. The needs are there to provide data support and automation in line with the fast paced advances of technology.

1.2 Data Definition and Management

Information is simply data with meaning applied. The amount of data stored on the computer is growing tremendously. Storing, manipulating, and retrieving data is increasing significantly due to technological advances. "The sheer volume of the information that is stored on computer equipment has fostered an atmosphere that has encouraged technological experimentation to bring about ways of storing and retrieving that information more efficiently." (4)

The process of defining data and the management of such definitions is a primary and necessary step. Being able to cohesively control the data definition process allows greater flexibility, integrity, and understanding. In order to manage the control and definition of the data, a data management system or tool is helpful. "A data management system is a collection of separable operations that define, store, update and retrieve data and also provide security, integrity, backup and recovery of information of various kinds..." (5).

Data is an expensive and time consuming corporate resource. The corporate dollars spent on salaries and benefits of those who manipulate data continue to increase. The cost of storage is

decreasing. Advancing hardware technology has provided solutions to storage capacity in such a fashion to virtually eliminate the high cost of data storage. Now we look to the area of software development to find solutions to the rising costs in that area.

It is increasingly necessary for data to be readily available for all persons in the corporation, including the managers, terminal operators, systems programmers, users, and application programmers. And in order for those persons to do their jobs effectively and as efficiently as possible, we look to improved communications with data and its definition. "It is desirable that there should be centralized control over the data and their structuring but nevertheless, that usage of the data should develop freely by the using departments." (6)

Disorder is the by-product of lack of data definition and management tools. We are slowly bringing discipline to this disorder. Discipline needs to continue to be applied to this disorder to keep communication flowing and understanding clear at all levels. The level of frustration rises when communication and understanding ceases between organizations or groups who are "responsible" for data integrity. How can we build structures without knowing the properties involved? How can we control information without knowing the data involved? "Although we have become more efficient in the methods we use to collect, compute, and disseminate data, we are still relative novices in our understanding of the characteristics and relationships of the data itself." (6)

1.3 *The Concept of Data Dictionaries*

The concept of data dictionaries as a central collection point for data creates a more efficient system of data management. "A data dictionary, in concept, is somewhat like a word dictionary. Within a data dictionary reside the terms used to identify the data elements used in computer applications, as well as terms used to define the components found in computer networks." (8)

Data dictionaries are tools defined to maintain cohesive understanding of each piece of data. This allows the data to be examined and dissected into its elementary components so that they are viewable by all parties. This allows the collection process to be complete and the dictionary to provide complete data definition and control.

1.4 **The Role of Data Dictionary Systems**

The role of data dictionaries is for collection of data as well as implementation of data bases. The initial cost of generating a data dictionary is easily justified when compared with the cost of determining maintenance changes to an existing system. When time is applied to the development of the data dictionary, success is more easily achieved. "There is a direct proportion between the effort and thoroughness invested in developing a dictionary and the success of the proposed database." (9)

There are three functional areas that most likely are to be addressed by a data dictionary system. The first area is the design

of the data base system. The second functional area is the implementation of the data base system. And the third functional area is the "in-line" operational aids.

In the first area, the data dictionary is called a design aid. "...the data dictionary serves as a repository of information collected during the analysis that leads to the creation of the data base." (10) This data dictionary when used as a design aid stores what is called structural knowledge. This information includes the data element definition, approximation of usage frequencies, security by user as well as program, and so forth. Before the implementation occurs, this information serves as documentation and also a monitoring base. These components give us structural knowledge of the database. "We define structural knowledge to be the knowledge we have about dependencies and constraints among the data, restricting ourselves to general and intensional information." (11) This structural knowledge directly feeds the database design phase.

The second functional area is the implementation of the data base system. "A data dictionary can provide a number of services, which collectively improve the productivity of data base administration and programming." (10) Areas serviced here include data base description, sub-schema generation, and data description for application programs. IBM's IMS system uses this technique to create such things as data base definitions, segment search arguments, control blocks, and support blocks. These are called administrative aids.

The third functional area is "in-line" operational aids. In this sense, the data dictionary becomes an integral part of the executing data base management system. The data dictionary contains information which assists in scheduling jobs, verification of security, and validation of transactions. Also, statistics of data base usage can be collected at this time. "A data dictionary is a repository of information about the data base system." (10) A data dictionary system uses this information to one or more of the functions discussed here.

1.5 The Role of Data Dictionaries in Software Development

A data dictionary is extremely valuable to designers and programmers in many phases of software development among which are: analysis, design, implementation, requirements, definition, coding, testing, and maintenance.

The analysis phase is critical to establishing a firm foundation for the proposed new or modified system. The process of analyzing a problem and developing a solution is aided by the data dictionary in several ways. "The dictionary can identify the use of data as it relates to the functions (processes) of the enterprise, and it can produce a description of data requirements in terms of an entity/attribute/relationship data (conceptual) model." (12)

This analysis phase is generally used to investigate and define the problems as well as the solutions. The designer benefits from elementary data stored in the dictionary. This data

enables the designer to present an accurate and comprehensive statement of the problem along with an orderly solution.

Another phase, the design phase, is generally used to analyze user requirements and ultimately determine a proposed system to meet such demands. At this time, the designer determines any secondary links that have been made into other existing systems. The data dictionary which shows "where used" information is extremely helpful in solving this problem quickly. The designer adds to the dictionary details of the data structures which will lead to the construction of the implementation model.

The implementation phase needs the dictionary to contain all possible information about data elements, programs using the elements, validation rules, frequency of use, any access paths to be followed, ranges of values, and detailed description of the data structure. During the implementation phase, the designer or programmer uses the dictionary as a consistency and validation tool.

The coding, testing, and maintenance phases all are benefitted by the inclusion of the data dictionary. This inclusion allows each area a check and balance routine for up-to-date information. The verification process allows these phases the luxury of knowing they are using the latest information and structures.

1.6 The Benefits of Data Dictionaries

The benefits of implementing a data dictionary exceeds the effort expended. The data dictionary requires the technical expertise to implement a vendor purchased system or the expertise to write an individual system. However, once in place and once the training has been accomplished then the continued use of the system allows for open communication and understanding.

The data dictionary eliminates the fear that the information is lost when the person who "knows about the data" takes vacation or quits. No longer is the person with the most senority on a system tied to the technology transfer or education of "new hires". The answers are derived from the data dictionary system.

Additional benefits are derived from the data dictionary system. These include: (1) Data about data is efficiently stored and retrieved for later use. (2) All levels of expertise (i.e. managers to terminal users) become involved in capture and maintenance of data. (3) Documentation effort is a product of the organization structure applied to centrally control the data information. (4) Data security can be incorporated into the storage attributes allowing the data dictionary or the data base management system to be responsible.

1.7 The Role of the Subset Data Dictionary Verifier

This project involves the maintenance of data integrity with a master data dictionary and a subset data dictionary. The Subset

Data Dictionary Verifier will examine the subset dictionary in relation to a master data dictionary.

The master data dictionary is one that has been developed from the Entity Relationship Attribute (ERA) specification. The ERA specification defines the data entities in keyworded elements called frames. All elements should be found in the master dictionary. New elements would not be generated in the creation process of developing the subset data dictionary. The master data dictionary may, on the other hand, contain elements not defined in the subset data dictionary.

The verification process is to insure that the entities in the subset data dictionary are consistent with those contained in the master data dictionary. The verification process also increases the confidence level in the data contained in the subset data dictionary.

The result of the project shows the missing frames or discrepancies encountered between the subset data dictionary and the master data dictionary. The verification process is keyword oriented, therefore it can show that some pieces of data included in the frame may in actuality be valid while other pieces of data in the same frame are not.

1.8 **The Benefits of the Subset Data Dictionary Verifier**

Automated verification is a benefit that is derived from the Subset Data Dictionary Verifier. The automated process of verification

gives notification of discrepancies based on the concept that the master dictionary is all-inclusive. Therefore, if all keyworded elements are precisely derived from the master dictionary, data integrity can be assumed. Also 100% confidence levels can exist when the subset data dictionary has been verified with the frames involved.

Some data definition systems allows a security level to be identified on each unit of data. This security definition is often the responsibility of the data base administrator. The organization or persons viewing these definitions sometimes constitute the security level necessary. A properly applied security level allows information and data to be applied to the responsible organizations.

The integrity derived from data definition control is beneficial. When there are multiple persons or organizations depending on the same data, it is extremely critical that all persons involved can depend on data definition integrity. Information is of no subsequent value to the dependent organization if allowed to "age" or "grow obsolete". Furthermore, we can assure ourselves that the data definitions are consistent among all organizations because there is only one version of the data definition, not multiple versions.

Verification process checking keeps those concerned aware of changes or updates. Frustration levels can be reduced if the data definitions stored in the dictionary are kept current; this will

ease the pains of the maintenance process. Once we allow our data definitions to become archaic then our entire system of programs, documentation, and communication have also become archaic.

The verification process insures discrepancies can be highlighted and integrity maintained when properly applied. Once the subset dictionary has been extracted, the verification process also assures the user he has up-to-date information. The capability to invoke the verification routine allows the subset data dictionary to continue independent of the master dictionary.

CHAPTER 2

THE SUBSET DATA DICTIONARY VERIFIER REQUIREMENTS

This project is a Subset Data Dictionary Verifier. The software verifies the frames in a subset data dictionary as consistent with a master data dictionary. This Subset Data Dictionary Verifier provides the user with a beneficial package that enhances the application of data dictionaries and their relationships and independencies.

The Subset Data Dictionary Verifier reads two data dictionaries as input (figure 2.a). One dictionary is called the subset data dictionary and the other is called the master data dictionary. The data dictionaries consist of a set of frames (figure 2.b). The frame is not in a predefined fixed order. Each frame contains information about one entity and each frame starts on a newline. The first line of the frame contains the keyword that describes the type of the entity and the frame of the entity. The type and the name are separated by a colon. At least one blank line separates each frame. The BNF for the data dictionaries is found in Appendix A.

The subset data dictionary may have been developed from the master data dictionary; however, this is not a requirement. It is assumed that the master data dictionary is all-inclusive and no elements have been developed in the extract process. The verification process highlights any discrepancies found between the subset data dictionary and its master data dictionary. The output

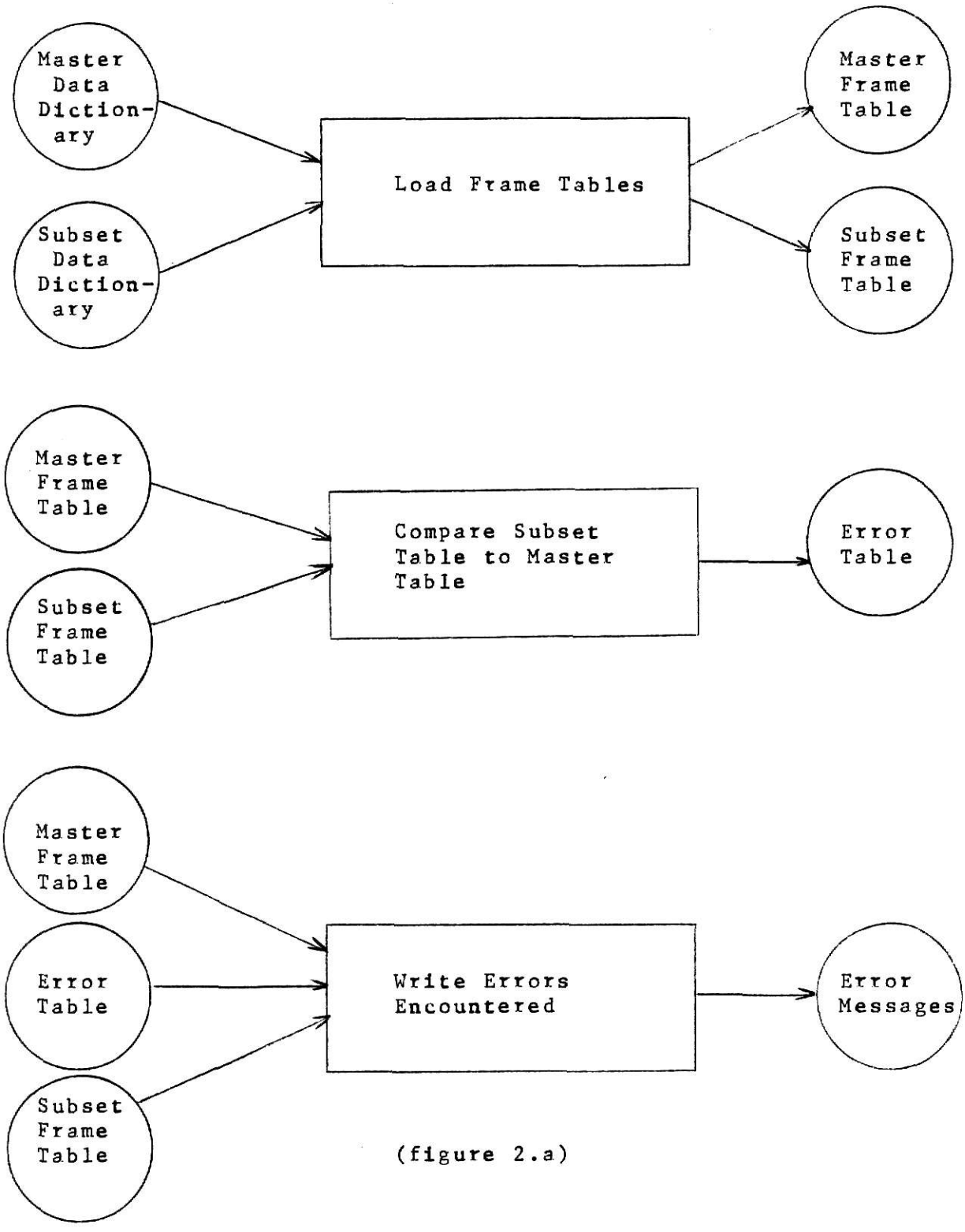
shows the frames from the subset data dictionary which can not be located on the master data dictionary as well as discrepancies between elements of the frames on both dictionaries. The comparison is performed at the keyword level insuring greater flexibility in the system.

The output error messages from the verification process is routed to standard output. If the subset frame can not be located on the master data dictionary, then an appropriate error message is written and the subset frame found in error. If the subset frame has entities which does not match the master frame entities, then an appropriate error message is generated showing which lines are in error and the subset frame and the master frame.

Many requirements were expected and received from the implementation of the Subset Data Dictionary Verifier. Among these requirements are order independence of the keywords within each frame, order independence of the data within each data dictionary, modularity of the code to provide ease in maintenance, and no limitations to the number of keywords. The flexibility and maintainability requirements is achieved through the modularity of the code. The useage of the programming language C allows the additional benefits of portability on various Unix based systems.

This project allows increasing control among "using" organizations for data that is relevant to their needs. However, absolute control is maintained on the master data dictionary by the responsible support organization.

Subset Data Dictionary Data Flow Diagram



(figure 2.a)

Sample Data Dictionary Format

```
NAME : $piece$
COMPOSITION : a string from the set {Kr,Kk,Kb,K,Q,Qb,Qk,Qr,p}

NAME : $rank$
COMPOSITION : a string from the set {1,2,...8}

NAME : $board_description$
SOURCE : keyboard
COMPOSITION : 'white' set of $piece_position$
            : 'black' set of $piece_position$
            : 'end'

NAME : $create$
SOURCE : keyboard
COMPOSITION : 'create'
```

(figure 2.b)

CHAPTER 3

THE SUBSET DATA DICTIONARY VERIFIER DESIGN

The Subset Data Dictionary Verifier is a software package designed to enhance data integrity and data independence among interrelated data dictionaries. It allows the user the flexibility of having a subset data dictionary pertinent to his needs yet it can be verified against its master data dictionary for possible updates. The user then only needs to become concerned with his "need to know" data or information. The results show any discrepancies found between the master data dictionary and the subset data dictionary.

The main processing function of the software is divided into 3 sub-functions; load entity tables, compare subset table to master table, and error condition encountered (figure 3.a). Each sub-function works with the frames in the dictionaries as keyword oriented data. Detailed specifications are found in Appendix D.

The first sub-function (figure 3.b), load frame tables, performs the functions of loading the subset frame into an internal subset table, finding the matching master frame, and then loading the master frame into an internal master table. During the load of the internal subset table and the internal master table, the keyword is identified and placed into the table separate from its data. If the subset frame can not be found on the master dictionary then an error flag is set.

The second sub-function (figure 3.c) compares the internal subset table with the internal master table. The keyword in the subset table is found in the master table, else an error flag is set. The data of the subset keyword is compared to the data of the master keyword. If discrepancies exist an error flag is set.

The errors flagged in the first and second sub-functions, are reported in the third sub-function (figure 3.d) called "error condition encountered". If a subset frame is not found on the master data dictionary, then an error condition is reported. If an element in the subset frame does not agree with the master frame, then an error condition is reported.

Internally the Subset Data Dictionary Verifier uses an error table. The error table's function is to hold the line number of the error detected during the keyword by keyword comparison process.

The error table has the subset element subscript and the master element subscript matched in the same error table entry. This matching shows which subset line disagrees with which master line. This matching shows the keyword can be matched but the values are different. If the subset frame element keyword can not be matched to the master frame element, then only the subset line number is in the error table entry; and the master line number entry is null.

The output from the Subset Data Dictionary Verifier is a printed report showing discrepancies found during the verification

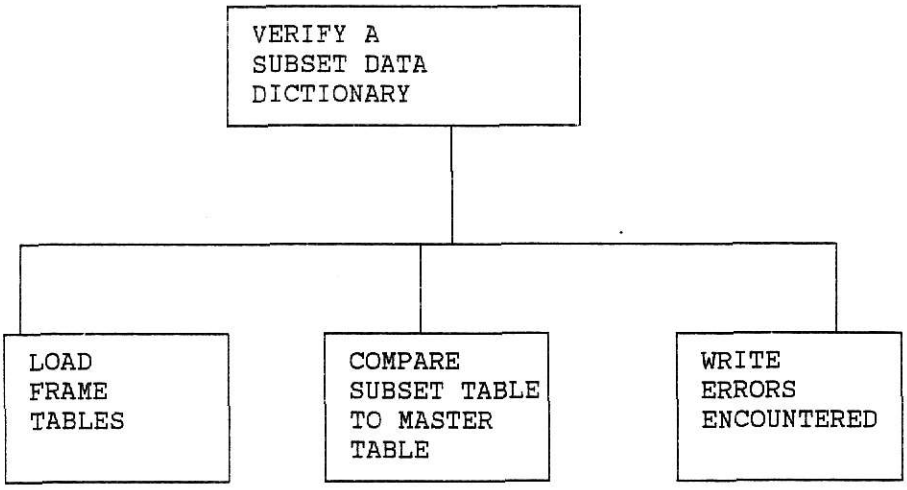
process. When a "not found" condition exists, the error message and subset frame is printed. When a "mismatch" error occurs, the error message denoting the line(s) in error, the subset data dictionary frame, and the master data dictionary frame is printed. The error table which contains the master and subset line subscripts in error is printed at this time as the line(s) in error.

The Subset Data Dictionary Verifier is designed to be modular in concept. Each function in the hierarchy diagram (figures 3.a - 3.d) is intended to have a single task to perform. The complete verification process is designed to manipulate keyworded elements within a frame regardless of the extent of the data associated with the element.

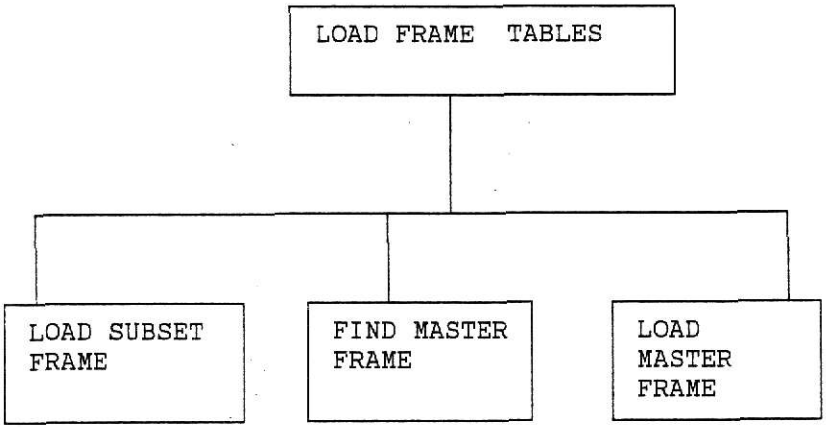
The Subset Data Dictionary Verifier is also designed to be as flexible as possible. This process allows greater flexibility than the BNF format as outlined in Appendix A. This allows the flexibility of single data items in a keyword or multiple data items in a keyword. This concept allows greater flexibility and integrity among data dictionaries using the master/subset concept.

The Subset Data Dictionary Verifier also provides an easily readable form of any discrepancies found in the input dictionaries which are using the ERA specification format as their basis. Meaningful error messages are used to help clarify any discrepancies.

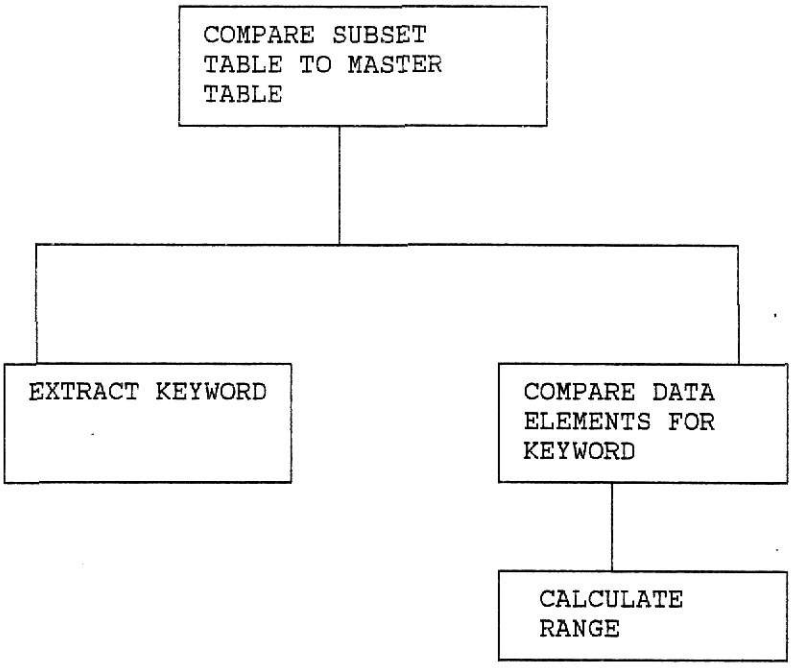
Hierarchy diagram for the
Subset Data Dictionary Verifier



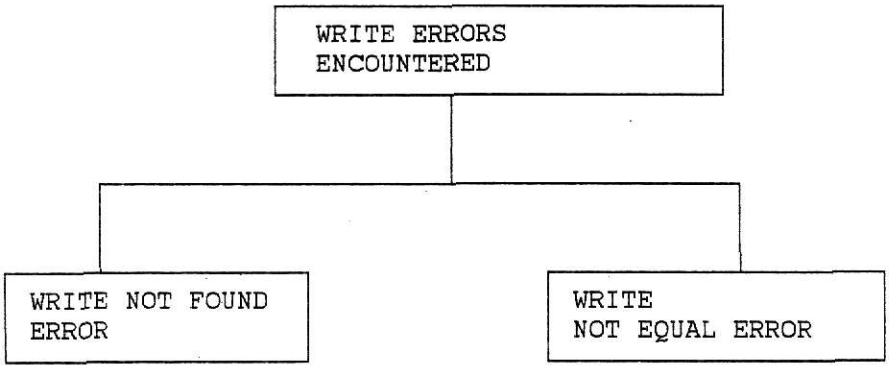
(figure 3.a)



(figure 3.b)



(figure 3.c)



(figure 3.d)

CHAPTER 4

THE SUBSET DATA DICTIONARY VERIFIER IMPLEMENTATION

The Subset Data Dictionary Verifier is implemented on the 8/32 at Kansas State University. This software is written in C and has approximately 500 lines of code. The inputs to the Subset Data Dictionary Verifier are two data dictionaries (figure 4.a); one is named "subset" and the other is named "master". If the program is invoked with less than 2 input files, an error message is printed that reads "need 2 files for input/subset & master".

The two dictionaries are compared using internal tables (figure 4.b) to store the frames to be compared. The logic uses an error table (figure 4.c) to store the line numbers in error.

The output consists of two possible error conditions. If the frame from the subset data dictionary can not be found on the master data dictionary then an error is printed along with the subset frame (figure 4.d). If any element in the subset disagrees with the master, then an error is printed as well as the subset frame and the master frame (figure 4.e).

When the verification process terminates successfully, i.e. no errors, then the program completes without listing any errors. A completion message is issued at termination.

Example of a data dictionary from the
ERA Specification

```
NAME : $piece_position$
SOURCE : crt
COMPOSITION : $piece$',$position$

NAME : $store$
SOURCE : crt
COMPOSITION : 'store'$name_of_game$

NAME : $time_out$
SOURCE : crt
COMPOSITION : 'too much time - game over'

NAME : $computer_move_message$
SOURCE : crt
COMPOSITION : 'computer's move is $position$ 'to' $position$
```

(figure 4.a)

Subset and Master Tables

```
struct s_array    /* subset frame table  */
 { char skey[MAXKEY];
   char sdata[MAXDAT];
 } subtab[MAXFRM];


struct m_array     /*  master frame table  */
 { char mkey[MAXKEY];
   char mdata[MAXDAT];
 } mastab[MAXFRM];


#define MAXKEY 15    /*  maximum number of chars in the key */
#define MAXDAT 80    /*  maximum number of chars in the data */
#define MAXFRM 100 /* maximum number of entries in a frame  */
```

(figure 4.b)

Internal Error Table

```
struct err_array;
 { int serr;
   int merr;
 } errtab[MAXFRM];

#define MAXFRM 100  /* maximum entries in a frame */
```

(figure 4.c)

```
THE FOLLOWING SUBSET ELEMENT COULD NOT BE FOUND IN
THE MAJOR DICTIONARY


Input : $piece_position$
  media:crt
  structure : $piece$',$position$
```

(figure 4.d)

```
AN ERROR HAS OCCURRED ON LINE(S): M2/S2; M4/S4; M5/S5; S8
```

Major Dictionary

```
M1    Input : $board_description$
M2      media : crt
M3      structure : 'white'
M4                  set of $piece_position$
M5                  'black'
M6                  set of $piece_position$
M7                  'end'
```

Subset Dictionary

```
S1    Input : $board_description$
S2      media : crd
S3      structure : 'white'
S4                  set of $pieces$
S5                  'blk'
S6                  set of $piece_position$
S7                  'end'
S8                  'finished'
```

(figure 4.e)

CHAPTER 5

CONCLUSIONS AND EXTENSIONS

The Subset Data Dictionary Verifier design provides a software tool to be used to validate a subset data dictionary and a master data dictionary for consistency between frame entities. The verification process is keyword oriented achieving great flexibility. This flexibility allows the capability of verifying the Entity Relationship Attribute specification directly. The software package has greater flexibility with the input dictionaries because the verification process is not dependent on any given sequence or order among the entities or the frames. The subset dictionary is verified directly against the master dictionary.

An extension to this design package would be to incorporate more user interface with the program directly. This system is currently a batch system; the user executes the program and sees the errors printed. If the user could interact with the software directly at a terminal, benefits such as verifying only a particular portion of the dictionary at a time could be achieved.

Modifying the program to allow the user to input a subset frame for direct verification with an online data dictionary would be an additional enhancement to the current system. This benefit would be extremely helpful for the design and modification phases when elementary data is being captured to provide a foundation for

the acceptance or rejection of a project. Also a programmer could verify a given subset frame in relation to an online master dictionary frame when he is in the process of program coding or maintenance.

Also, direct feedback from the program to the user could allow the user the option of selecting the "correct" master frame to replace the "incorrect" subset frame. This selection would generate a new up-to-date subset data dictionary. This benefit would minimize the process of creating a subset dictionary by another mechanism.

The Subset Data Dictionary Verifier has been designed to provide flexibility to the verification process. Also, the modularity of the coding has enabled the implementation to be a smooth transition. The modularity will provide greater ease should any of the above enhancements be applied.

REFERENCES

(1) Graham, Alan K., "Software Design: Breaking the Bottleneck", *IEEE Spectrum*, Vol 19, No 3, Mar 1982, p 43

(2) Edward A. Feigenbaum and Pamela McCorduck, *The Fifth Generation*, Reading, Massachusetts:Addison-Wesley Publishing Co., 1983, p.11

(3) William Durell, "Disorder to Discipline Via the Data Dictionary", *Journal of Systems Management*, May 1983, p 12

(4) Myers E. Walsh, *Database & Data Communications Systems: A Guide for Managers*, Reston Publishing Co. Inc., Reston, Virginia, 1983, p. 129

(5) Elizabeth Unger, Paul Fisher, and Jacob Slonim, "Evolving to distributed database environments", *Computer Communications*, Vol. 5, No. 1, Feb 1982, p 17

(6) James Martin, *Principles of Data-Base Management*, Englewood Cliffs, N.J.:Prentice-Hall, Inc, 1976, p 221

(7) William Durell, "Disorder to Discipline Via the Data Dictionary", *Journal of Systems Management*, Vol 34, May 1983, p 12

(8) Ibid, p.128

(9) Roy Graham, "How to Develop Databases and Data Dictionaries", *Management World*, Vol 10, Oct 1981, p. 33

(10) Leo J. Cohen, "Data Dictionary Systems, What Are They?", Data Base Newsletter, Performance Development Corp, Sept. 1977, pp 2-3

(11) Wiederhold, Gio, "Knowledge and Database Management", *IEEE Software*, Vol 1. No 1., Jan 1984, p 65

(12) J. Van Duyn, *Developing a Data Dictionary System*, Englewood Cliffs, N.J.:Prentice-Hall, Inc., 1982, p 106

Appendix A

Detailed Specifications in BNF Format

General Description

The era specification will consist of a set of frames. The order of the frame is not fixed. Each frame will contain information about one entity. Each frame will start on a newline. The first line in the frame will contain the keyword that describes the type of the entity and the name of the entity. The first letter in the type is capitalized. The type and the name are separated by a colon. At least one blank line will separate each frame.

The information in a frame is generally in the form of relations between this entity and other entities. Some of the information is in the form of attributes. An attribute gives information about this entity without referring to other entities. The order of these relations/attributes is not fixed.

Each relation/attribute is specified by a keyword that specifies the relation/attribute and its value. The value is either the name of the entity that has that relation or a text description of the attribute value. A colon separates the keyword and its value. Each relation/attribute starts on a new line. If a relation/attribute continues on to another line, the continuation line starts with a blank field followed by a colon. Multiple occurences of a relation/attribute is represented by multiple occurences of the keyword.

Entity Types

These entity types are not fixed. Additional entity types may be defined in the future. All entity types will start with a capital letter.

Activity
Type
Input
Output
Periodic_function
Input_output
Data
Constant
Comment
* additional entity types may be added at any time

Relations/Attributes

keywords
input
output
required_mode
necessary_condition
occurence
assertion
action
comment
media
structure
type
units
subpart_is
subpart_of
uses

* additional entity types may be added at any time

Syntax Description

```
<era_spec> ::=
        <era_title> <era_body> <mode_table>

<era_title> ::=
        PROCESS : <text>

<era_body> ::=
        <frame> | <frame> <era_body>

<frame> ::=
        <NL> <NL> <frame_header> <frame_body>
      | <NL> <NL> Comment : <text_lines>

<frame_header> ::=
        <i_o_data_header> : <i_o_data_name>
      | <function_header> : <CAPITAL_WORD>

<i_o_data_header> ::=
        Type | Input | Output | Input_output | Data
      | Constant | <CAPITAL_WORD>

<function_header> ::=
        Activity | Periodic_function | <CAPITAL_WORD>

<frame_body> ::=
        <relation> | <relation> <frame_body>
```

```
<relation> ::=
        <NL_B> <relation_type> : <relation_value>

<relation_type> ::=
        keywords | input | output | required_mode
      | necessary_condition | occurence | assertion
      | action | comment | media | structure | type
      | units | subpart_is | subpart_of | uses | <WORD>

<relation_value> ::=
        <text_lines> | <structure>

<structure> ::=
        <struct> | <struct> <NL_B> : <structure>

<struct> ::=
        <name> | <text> | <name> <structure> | <text> <structure>

<name> ::=
        <mode_name> | <i_o_data_name>

<i_o_data_name> ::=
        $ <WORD> $

<mode_name> ::=
        * <WORD> *

<mode_table> ::=
        <NL> <NL> MODE_TABLE <mode_list> <inital_mode> <transition_body>

<mode_list> ::=
        <mode> | <mode> <mode_list>

<mode> ::=
        <NL_B> Mode : <mode_name>

<initial_mode> ::=
        <NL> <NL_B> Initial_Mode : <mode_name>

<transition_body> ::=
        <NL> <NL_B> Allowed_Mode_Transitions : <transition_list>

<transition_list> ::=
        <transition> | <transition> <transition_list>

<transition> ::=
        <NL_B> <event> : <mode_name> -> <mode_name>

<event> ::=
        <i_o_data_name>
```

```
      | <i_o_data_name> = ' <text> '
      | <function_header>


<text_lines> ::=
      <text> | <text> <text_cont>

<text> ::=
      <WORD> | <WORD> <text>

<text_cont> ::=
      <NL_B> : <text> | <NL> : <text> <text_cont>

<NL> ::=
      '0 | '0 <NL>

<NL_B> ::=
      <NL> '   '
```

Lexical Scanner Information

Tokens used in the productions above may begin with <char> or one of the following characters: *$,':-={} Blanks can delimit tokens as well.

The following tokens are important above:

```
      <WORD>          ::= <char> | <char> <WORD>
      <CAPITAL_WORD>  ::= <capital_letter> <WORD>

<char> ::=
      <lower_case_char> | <symbol>

<lower_case_char> ::=
      a | b | ...| z | 0 | 1 | ... | 9

<symbol> ::=
      # | % | & | ( | ) | ? | _

<capital_letter> ::=
      A | B | ... | Z
```

There exists a set of "reserved word" tokens which includes:
{keyboard,crt,internal,secondary_storage,NONE,every,mode}

Appendix B

USER'S MANUAL

This program is a non-interractive program which was written to be executed in background mode. To invoke the program, enter the following:

```
cc sddv
a.out subfilename masterfilename > sub.out 2>errors.out
```

where: sddv = program code name
subfilename = subset data dictionary to be used
masterfilename = master data dictionary to be used
sub.out = file to contain new compared equal file
errors.out = file to contain any error messages

Appendix C

SOURCE CODE LISTING

```
#include <stdio.h>
#include <strings.h>
#define MAXFRM 100 /* max entries per frame   */
#define MAXLNE 80  /* max characters per line */
#define MAXKEY 15  /* max characters per key  */
#define MAXDAT 80  /* max characters in data  */

/*  global variables follow    */

  int err_sub;
  int sub_sub;
  int mas_sub;
  int endall;
  int error;
  int sub_stop;
  int mas_stop;
  int max_sub;
  int max_mas;
  int endit;
  int line_sub;
  int err_code;
  int *masptr;
  int *subptr;
  int error_ct;
  char line[MAXLNE];
  char save[MAXLNE];

  struct m_array
   { char mkey[MAXKEY];
     char mdata[MAXDAT];
   } mastab[MAXFRM];

  struct s_array
   {  char skey[MAXKEY];
      char sdata[MAXDAT];
   } subtab[MAXFRM];

  struct err_array
   { int serr;
     int merr;
   } errtab[MAXFRM];

  FILE *insub, *inmas;
```

```
/*    this is the main function for the     */
/*    Subset Data Dictionary Verifier       */
/*                                          */

main(argc,argv)

  int argc;
  char *argv[];

 {
  endit = 0;
  max_sub = 0;
  max_mas = 0;
  err_code = 0;

   if (argc == 3)
     ;
   else
  { printf("need 2 files for input/subset & master0);
       return;  }

  insub = fopen(*++argv,"r");
  inmas = fopen(*++argv,"r");

  printf("begin sddv main0);

  cleanout();

  loadem();


  while (endit == 0)
   {
     if (sub_sub == 0) break; /* no entries  */

     if (err_code == 0)
      { cmpar();
        if (err_code != 0)
         err();
      }
     else
      { err();  }
    cleanout();
    loadem();
    }
    printf("end sddv main0);

  }
/*  end of the main function code */
```

```
/*  this is the cleanout function.  this function will     */
/*  set the subset keyword & data table to null lines      */
/*  as well as the master keyword & data table.            */
/*  this function is called f
rom     main   function      */
/*                                                         */

   cleanout()

    {

    int cl_sub;
    int subit;

    for (subit = 0;
         subit <= MAXFRM;
         subit++)
            for (cl_sub = 0;
                 cl_sub <= MAXKEY;
                 cl_sub ++)
       { mastab[subit].mkey[cl_sub] = ' ';
         subtab[subit].skey[cl_sub] = ' ';   }


    for (subit = 0;
         subit <= MAXFRM;
         subit++)
            for (cl_sub = 0;
                 cl_sub <= MAXDAT;
                 cl_sub++)
        {  mastab[subit].mdata[cl_sub] = ' ';
           subtab[subit].sdata[cl_sub] = ' ';  }

    }
     /*    end cleanout   function         */
```

```
/*  this is the loadem function.    */
/*  this function builds the        */
/* subset array with the frame to   */
/*  be verified, and                */
/* it also builds the master array  */
/*  from the matching               */
/*  frame entity.                   */
/*  this function is called from     */
/*  main                             */
/*                                   */

loadem()

  {

    int subit;
    int maximum;
    int next_sub;
    int chk_strg;


    subit = 0;
    next_sub = 0;
    chk_strg = 0;

    loadsub();

    if (endit != 0)
      return;

    max_sub = sub_sub;

    for (sub_sub = 1;
         sub_sub < max_sub;
         sub_sub ++)
      { if (subtab[sub_sub].skey[0] == ' ')
            ;
        else
         {
    for (next_sub= sub_sub+1;
         next_sub < max_sub;
         next_sub++)
         {  if (subtab[next_sub].skey[0] == ' ')
                ;
            else
         {  chk_strg=(strcmp(subtab[sub_sub].skey,
                             subtab[next_sub].skey));
```

```
            if (chk_strg == 0)
              {  subtab[next_sub].skey[0] = ' ';
                 subtab[next_sub].skey[1] = ' ';
                 }
            }
          }
          }
        }

    findmas();

    if (err_code == 1)
      return;

    loadmas();
    max_mas = mas_sub + 1;

     for (mas_sub = 1;
          mas_sub < max_mas;
          mas_sub ++)
      { if (mastab[mas_sub].mkey[0] == ' ')
            ;
        else
         {
    for (next_sub= mas_sub+1;
         next_sub < max_mas;
         next_sub++)
         {  if (mastab[next_sub].mkey[0] == ' ')
                ;
            else
         {  chk_strg=(strcmp(mastab[mas_sub].mkey,
                             mastab[next_sub].mkey));
            if (chk_strg == 0)
              {  mastab[next_sub].mkey[0] = ' ';
                 mastab[next_sub].mkey[1] = ' ';
               }
            }
          }
          }
        }

  }
/*     end of the loadem function                */
/*                                               */
```

```
/*  this function loads the subset dictionary  */
/*  table with one complete frame.             */
/*    Endit becomes "one" when null            */
/*  character is found.                        */
/*  this function is called by    loadem       */
/*                                             */

   loadsub()

    {


      sub_sub = 0;
      line_sub = 0;
      endit = 1;

   another:

     if ((subptr =fgets(line,MAXLNE,insub)) == NULL)
               {  return; }

   if (line[0] == '0)
       return;

        endit = 0;

             if (line[line_sub] == ' ')
              { subtw3ab[sub_sub].skey[0] = ' ';
                 subdata(); }
             else
               { subkey();
                 line_sub++;
                 subdata(); }

             sub_sub++;
             line_sub = 0;
             goto another;

     }

/*   end    loadsub    function    */
```

```
/*  this function loads the key position of the  */
/*  frame into the subset table.                 */
/*  this function is called by     loadsub       */
/*                                               */

   subkey()

   {

     int key_sub;

     key_sub = 0;

     while (line[line_sub] != ':')
       {  subtab[sub_sub].skey[key_sub] = line[line_sub];
          key_sub++;
          line_sub++;  }

   }

/*      end of     subkey    function     */
```

```
/*  this function loads the data portion of the  */
/*  frame into the subset table.                 */
/*  this function is called  by     loadsub      */
/*                                               */

   subdata()


   {

     int data_sub;


     data_sub = 0;

     while (line[line_sub] != NULL)
       { subtab[sub_sub].sdata[data_sub] = line[line_sub];
         data_sub++;
         line_sub++;  }

   }

/*   end of     subdata     function    */
```

```
/*   this function tries to find a matching frame on the */
/*   master dictionary.  this function expects the master*/
/*   files needs to be rewound to begin processing at the*/
/*   beginning of the file.                               */
/*   this function is called by       loadem   function. */
/*                                                        */

   findmas()

     {

       int chk_strg;

       mas_sub = 0;
       line_sub = 0;
       err_code = 1;

       rewind(inmas);

   again:
     if ((masptr= fgets(line,MAXLNE,inmas)) == NULL)
                 return;

           if (line[0] == '0)
                goto again;
          if (line[0] == ' ')
            goto again;

          line_sub = 0;
          maskey();
          line_sub++;
          masdata();

          chk_strg = (strcmp(mastab[0].mkey,subtab[0].skey));
          if (chk_strg == 0)
       { chk_strg = (strcmp(mastab[0].mdata,subtab[0].sdata));
              if (chk_strg == 0)
               {  err_code = 0;
                  return;  }
              else
                  goto again;  }
           else
              goto again;
   }

 /*  end    findmas    function      */
```

```
/*  this function builds the keyword     */
/*  in the master table.                 */
/*  this function is called by           */
/*   findmas   &  loadmas                */
/*                                        */

   maskey()


    {

     int key_sub;


     for (key_sub = 0;
          key_sub <= MAXKEY;
          key_sub++)
       mastab[mas_sub].mkey[key_sub] = ' ';

     key_sub = 0;

     while (line[line_sub] != ':')
       {  mastab[mas_sub].mkey[key_sub] = line[line_sub];
          key_sub++;
          line_sub++;  }


    }

 /*   end      maskey       function        */
```

```
/*  this function loads the data portion  */
/*  of the master frame                   */
/*  this function is called by            */
/*    loadmas   &   findmas               */
/*                                        */

   masdata()


    {

     int  data_sub;

     for (data_sub = 0;
          data_sub <= MAXDAT;
          data_sub++)
       mastab[mas_sub].mdata[data_sub] = ' ';


     data_sub = 0;

     while (line[line_sub] != NULL)
       { mastab[mas_sub].mdata[data_sub] = line[line_sub];
         data_sub++;
         line_sub++;  }


    }

  /*   end of    masdata     function   */
```

```
/*  this function loads the frame that was located in     */
/*  findmas function in the master array.                 */
/*  this function is called by   loadem   function        */
/*                                                        */

   loadmas()

     {



         line_sub = 0;

   readm:
    if ((masptr = fgets(line,MAXLNE,inmas)) == NULL)
            return;

          if (line[0] == '0)
            return;

          mas_sub++;

         if (line[0] == ' ')
              { mastab[mas_sub].mkey[0] = ' ';
                masdata();   }
         else
           {  maskey();
              line_sub++;
              masdata();  }

         line_sub = 0;

         goto readm;

     }

 /*  end        loadmas       function     */
```

```
/*  this function calls the function to     */
/*   locate the keywords                    */
/*  in the subset and master tables;  and   */
/*  calls the function                      */
/*  that compares the data and prepares     */
/*  the error table                         */
/*  this function is called by   main       */
/*                                          */

  cmpar()

    {
    endall = 0;
    sub_sub = 0;
    err_sub = 0;


    while (endall == 0)
      {  mas_sub = 0;
         extract();
         match();
         }

     }

/*  the end of the cmpar function           */
/*                                          */
```

```
/*  this function finds the next keyword in the subset  */
/*  and master tables.                                  */
/*  this function is called by   cmpar  function        */
/*                                                      */

 extract()
   {

     int chk_strg;


   while (subtab[sub_sub].skey[0] == ' ' &&
          sub_sub < max_sub)
      sub_sub++;

   if (sub_sub >= max_sub)
     { endall = 1;
       return;  }

  keyword:
     chk_strg =
     (strcmp(mastab[mas_sub].mkey,subtab[sub_sub].skey));
    if (chk_strg != 0)
         if (mas_sub < max_mas)
             { mas_sub++;
               goto keyword;  }

   error = 0;

   if (mas_sub >= max_mas)
 {  err_code = 2;              /* error mismatch found */
         error    = 3;         /* keyword missing      */
         return;  }

   }

/*  extract function end          */
```

```
/*  this function compares the data lines in the */
/*  subset table with the master table.          */
/*  this function is called by   cmpar  function */

 match()

   {
   int ok;
   int next_sub;
   int mas_hold;
   int mas_err;



   if (endall == 1)       /* no more keywords, get out*/
       return;

   if (error == 3)          /* keyword missing  */
     {  errtab[err_sub].merr = ' ';
        errtab[err_sub].serr = sub_sub;
        err_sub++;
        sub_sub++;
        while (subtab[sub_sub].skey[0] == ' ' &&
                  sub_sub < max_sub)
           { errtab[err_sub].merr = ' ';
             errtab[err_sub].serr = sub_sub;
             err_sub++;
             sub_sub++;  }
           return;
     }

   range();
   mas_hold = mas_sub;


 loop: ok = 0;

   check_next:
      while (mas_sub < mas_stop)
        {
         if (strcmp(subtab[sub_sub].sdata,
                    mastab[mas_sub].mdata) != 0)
             {  ok = 2;
                mas_err = mas_sub;
                mas_sub++;
              }
          else
            { ok = 1;
```

```
                mas_sub = mas_stop; }
            }

          if (ok == 0)      /* no match encountered */
           { errtab[err_sub].merr = ' ';
             errtab[err_sub].serr = sub_sub;
             err_sub++;
             err_code = 2;
           }

          if (ok == 2)        /* mismatch encountered */
           { errtab[err_sub].merr = mas_err;
             errtab[err_sub].serr = sub_sub;
             err_sub++;
             err_code = 2;
           }

           sub_sub++;
           mas_sub = mas_hold;

           if (sub_sub < sub_stop)
             goto loop;

    }
/*      end  match  function          */
```

```
/* this function calculates the end of the */
/* data associated with the keyword.       */
/* this function is called by   match  function */

  range()

  {

  int begin;


  begin = 1;
  sub_stop = sub_sub + 1;
  mas_stop = mas_sub + 1;

  while (begin == 1)
    {
     if (subtab[sub_stop].skey[0] == ' ')
        sub_stop++;
     else
        begin = 0;
    }

   begin = 1;

   while (begin == 1)
    {
     if (mastab[mas_stop].mkey[0] == ' ')
       mas_stop++;
     else
       begin = 0;

    }


  }

/*  end   range  function          */
```

```
/*  this function is called when an error has occurred. */
/*  it determines which error message function needs    */
/*  to be invoked.                                     */
/*  this function is called from the main function.    */
/*                                                     */


  err()

  {


    if (err_code == 1)
         notfnd();
    else
         notequal();

    err_code = 0;


  }

/*  end of the err function                            */
/*                                                     */
```

```
/*  this is the function to print an error message for  */
/*  a subset frame not found in the master frames.      */
/*  the printout contains an error message and the      */
/*  subset frame in error.                              */
/*  this function is called by     err  function        */
/*                                                      */

  notfnd()

   {

     int sub;

     putchar('0);
     printf("The following subset frame could not ");
     printf("be found in the master dictionary0);
     putchar('0);

     for (sub = 0;
          sub < max_sub;
          sub++)
       printf("%s %s",subtab[sub].skey, subtab[sub].sdata);

   }

 /*    end of    notfnd    function       */
```

```
/*   this is the function to print an error message  */
/*   for the subset frame not equal the master frame.*/
/*   the printout has: error message, lines in error, */
/*   the subset frame, and the master frame.          */
/*   this function is called byerr    function        */

  notequal()

    {
      int line_ct;
      int hold_error;
      int sub;

      line_ct = 34;
      sub = 0;
      hold_error = error_ct;


     sub = 0;
     putchar('0);
     printf("An error has occurred on line(s): ");

     while (sub < err_sub)
       {
       if (errtab[sub].serr != ' ' &&
           errtab[sub].merr != ' ')
   printf("S%3d / M%3d0,errtab[sub].serr,errtab[sub].merr);
       else
         if (errtab[sub].serr != ' ' )
           printf("S%3d0,errtab[sub].serr);
         else
            ;
        sub++;
       }                    /*  end while statement  */

   printf("10ubset frame0);

   for (sub = 0;
        sub < max_sub;
        sub ++)
  printf("S%3d %s %s",sub,subtab[sub].skey,subtab[sub].sdata);



     printf("0aster frame0);

     for (sub = 0;
          sub < max_mas;
          sub++)
```

```
 printf("M%3d %s %s",sub,mastab[sub].mkey,mastab[sub].mdata);
 }
/*   end    notequal   function     */
```

Appendix D

Detailed Design Specifications

Hierarchy diagram module name: Verify a Subset Data Dictionary

Executable name: sddv

Module name: Main

Input: Subset Data Dictionary
Master Data Dictionary

Output: Error Message for less than 2 input files
Begin processing message
End processing message

Global Variables established: err_sub, sub_sub, mas_sub, endall, error, sub_stop, mas_stop, max_sub, max_mas, endit, line_sub, err_code, error_ct, m_array, s_array, err_array

Global Variables modified: endit, max_sub, max_mas, err_code

This is the main function for the Subset Data Dictionary Verifier. This module modifies global variables: endit, max_sub, max_mas, and err_code.

The global variables function as follows. Endit is set at end of file to "other than zero". Max_sub is used to determine maximum subset entries in the frame that is loaded into the internal table. Max_mas is used to determine maximum master entries in the frame that is loaded into the internal table. And err_code is used to report to the error printing routine the error message to be printed.

The main function calls 4 sub-functions. These are cleanout, loadem, cmpar, and err.

Appendix D

Module name: Cleanout

Input: Master Frame Table
Subset Frame Table

Output: Master Frame Table
Subset Frame Table

Global Variables Accessed: MAXFRM, MAXDAT

Global Variables Modified: s_array, m_array

This is the cleanout function that is invoked from the main function. This function sets the subset frame table and master frame table keyword and data fields to null. This eliminates the possibility of garbage from a previous frame accidentally being left in either table.

Module name: Loadem

Input: Subset Data Dictionary Table
Master Data Dictionary Table

Output: None

Global Variables Accessed: endit, max_sub, sub_sub, max_mas, err_code, s_array, m_array

Global Variables Modified: max_sub, sub_sub, max_mas, mas_sub

This is the loadem function that is invoked from the main function. This function calls three sub-functions. They are: loadsub, findmas, and loadmas. If no matching master frame is located by findmas, then an error flag tells this function to return to the main function without invoking loadmas. Otherwise all three functions are invoked by loadem and then loadem returns to the main function.

Module name: Loadsub

Input: Subset Data Dictionary

Output: None

Global Variables Accessed: endit, sub_sub

Global Variables Modified: endit, sub_sub

This function is invoked by the function loadem. This functions purpose is to load into the subset frame table the next subset frame found in the subset data dictionary file. If there are no more frames in the input file, then this function returns to the calling function, loadem.

This functions calls two functions. The first function is subkey and the second is subdata.

Module name: Subkey

Input: Subset Data Dictionary

Output: Subset Frame Table

Global Variables Accessed: s_array, sub_sub

Global Variables Modified: s_array, sub_sub

This function is invoked by the function loadsub. This function loads into the keyword field in the subset frame table the keyword found in the subset data dictionary input file. This function looks for the colon that separates the keyword and data to determine it is at the end of the keyword before returning to function loadsub.

Appendix D

Module name: Subdata

Input: Subset Data Dictionary

Output: Subset Frame Table

Global Variables Accessed: s_array, sub_sub

Global Variables Modified: s_array, sub_sub

This function is invoked from the function loadsub. This function loads into the data field in the subset frame table the data found in the subset data dictionary input file. This function looks for a null character to determine it is at the end of the data before returning to the function loadsub.

Appendix D

Module name: Findmas

Input: Master Data Dictionary

Output: None

Global Variables Accessed: mas_sub, m_array, err_code, line_sub

Global Variables Modified: mas_sub, line_sub, err_code

This function is invoked by loadem. This function tries to find a frame on the master data dictionary file to match the frame previously loaded by loadsub.

This function calls two functions. The first function is maskey and the second function is masdata.

Findmas then performs a string compare through the subset frame and master frame for an equal condition on the keywords. If the keywords are found, then a string compare is performed on the data for that keyword. If an equal condition occurs, the error code is set to zero; else an error of '1' is returned to the calling function loadem. The error condition '1' indicates it cannot find a matching frame on the master data dictionary input file.

Findmas returns to function loadem on two conditions. The first is the condition of an error present which is denoted by an error code '1'. The other condition is when a null character is found which denotes an end of file condition on the master data dictionary input file.

Appendix D

Module name: Maskey

Input: Master Data Dictionary

Output: Master frame table

Global Variables Accessed: MAXKEY, m_array, line_sub

Global Variables Modified: m_array, line_sub

This function is invoked by the functions loadmas and findmas. This functions loads into the keyword field in the master frame table the keyword found in the master data dictionary input file. This function looks for the colon that separates the keyword and data to determine it is at the end of the keyword before returning to the calling function.

Module name: Masdata

Input: Master data dictionary

Output: Master frame table

Global Variables Accessed: MAXDAT, m_array, line_sub

Global Variables Modified: m_array, line_sub

This function is invoked by the functions loadmas and findmas. This function loads into the data field in the master frame table the data found in the master data dictionary input file. This function looks for a null character to determine it is at the end of the data before returning to the calling function.

Appendix D

Module name: Loadmas

Input: Master Data Dictionary

Output: Master frame table

Global Variables Accessed: line_sub, m_array

Global Variables Modified: m_array, line_sub

This function is invoked by the loadem function. This function is invoked when a master frame has been located by findmas that is equal to the subset frame loaded in the subset table.

This function calls two functions. The first function is maskey and the second function is masdata.

This function returns to the loadem function when a null character is detected on the master data dictionary input file.

Appendix D

Module name: Cmpar

Input: none

Output: none

Global Variables Accessed: endall, sub_sub, mas_sub, err_sub

Global Variables Modified: mas_sub

This function is invoked by the main function. This function calls two functions. The first function is extract and the second function is match. There is a variable named endall that is set to zero. This variable is used by the extract function to tell cmpar when to return to the main function.

Appendix D

Module name: Extract

Input: Subset frame table
Master frame table

Output: Subset frame table
Master frame table

Global Variables Accessed: s_array, m_array, max_sub, sub_sub, max_mas, mas_sub, err_code, endall

Global Variables Modified: endall, mas_sub, sub_sub, err_code

This function is invoked by the function cmpar. This functions responsibility is to locate the next keyword in the subset frame table as well as the matching master frame keyword. If no match on the keyword is found, then an error code is returned to cmpar.

If the subset frame index exceeds the maximum subset index set as a result of the loadsub function, then this function returns to cmpar with the endall variable set on.

If the master frame index exceeds the maximum master index set as a result of the loadmas function while there is still subset data left to verify, then an error condition is set and this function returns to the cmpar function.

Module name: Match

Input: Subset frame table
Master frame table

Output: Error table

Global Variables Accessed: endall, err_array, mas_sub, mas_stop, err_sub, sub_sub, mas_err, err_code sub_stop

Global Variables Modified: err_array, err_sub, sub_sub, mas_err, mas_sub, err_code

This function is invoked by the function cmpar. This function returns to cmpar when no more keywords are available in the subset frame table. This function calls one subfunction. That subfunction is range.

If the extract function could not find a matching keyword on the subset frame and master frame, than an error was set by extract. This error code is passed by cmpar to match and match puts the error information into an error table and returns to cmpar.

If there is matching keywords in the subset frame and master frame then this function does a string compare on the data in the subset frame table and the master frame table. If the string compare locates a mismatch in the data portion of the subset frame and master frame then appropriate error information is set in the error table by this function.

Module name: Range

Input: Subset frame table
Master frame table

Output: Number of data lines

Global Variables Accessed: s_array, m_array, sub_sub, mas_sub, sub_stop, mas_stop

Global Variables Modified: sub_stop, mas_stop

This function is invoked by match to determine how many data lines are associated with a keyword within the subset frame table.

Module name: Err

Input: None

Output: None

Global Variable Accessed: err_code

Global Variable Modified: err_code

This function is called by the main function when an error condition occurs. The function calls one of two sub-functions. One sub-function is called notfnd. The other is called notequal. This function then returns to main with an error flag set to zero again.

Module name: Notfnd

Input: Subset frame table

Output: Error Message
Subset frame entry

Global Variables Accessed: max_sub, s_array

Global Variables Modified: None

This function is called by the function err. This function prints an error message for a subset frame not found in the master data dictionary input file. This function prints the error message "The following subset frame could not be found in the master dictionary" followed by the subset frame found.

Module name: Notequal

Input: Error Table Array
Subset frame table
Master frame table

Output: Error Message
Subset frame entry
Master frame entry

Global Variables Accessed: error_ct, err_array, s_array, m_array, max_sub, err_sub, max_mas

Global Variables Modified: None

This function is called by the function err. This function prints the error message "An error has occurred on line(s):" followed by the associated line numbers of the lines found to be in error. This function then prints an error message "Subset frame" followed by the subset frame. This function then prints an error message "Master frame" followed by the master frame.

THE IMPLEMENTATION OF A SUBSET DATA DICTIONARY VERIFIER

by

JACQUELYN FERN CLINE

B.S., Central State University, Edmond, Oklahoma, 1974

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Abstract

This implementation project is a Subset Data Dictionary Verifier. It is used to verify a subset data dictionary against a master data dictionary. Both input dictionaries have been developed from an Entity Relationship Attribute (ERA) specification. The master dictionary is considered to be complete; therefore all information found on the subset dictionary should be included in the master dictionary.

Both dictionaries have entities called frames. Each frame is denoted by a keyword and an entity name. The elements within a frame are not order sensitive. Each element is also keyworded, and each frame is separated by at least one newline character.

The output may consist of two error conditions. One is a "not found" condition; it is encountered if a frame on a subset dictionary can not be located on a master dictionary. The second is an "element mismatch" error; this is encountered when the data element keyword or its data in the subset frame does not agree with the master dictionary frame.

1430-60
CD-53