A DATA DICTIONARY/DIRECTORY DRIVEN CLINICAL DATA MANAGEMENT SYSTEM

ERIC S. HERBEL
HOECHST-ROUSSEL PHARMACEUTICALS, INC.

ABSTRACT

A clinical research data management system was developed which is driven by a hierarchical data base form of a data dictionary using variable length record KSAM files. The actual structure of the data dictionary (the hierarchical data base approach, and variable length lists implemented in KSAM variable length record files) is emphasized. The system includes data entry, editing, on-line corrections, a variety of IBM/370 resident interfaces, and a generalized IMAGE data base schema write/loader. All of the system modules are driven by the data dictionary. Details of the IMAGE interface are also emphasized.

OUTLINE

- The Data Dictionary/Directory concept in general
    - and the HP implementation in detail

- Overview of the clinical data management system
    - Data Entry - key to disk emulation
    - Data Preparation - editing, corrections, listing, etc.
    - Data Dictionary - definition, modification, reporting
    - Data Management - Image schema writer/loader, interfaces to IBM/370 DBMSs.

- IMAGE/3000 schema writer/loader in depth.

DESCRIPTION OF DATA DICTIONARY/DIRECTORY

Commercially available Data Dictionary/Directory systems are typically characterized by:

- identification of each element or field (name, aliases, description)
- type specifications for each element (numeric/character, etc.), field size and other physical descriptions

- reporting specifications for the field (output format, and default label)
- data editing criteria - a range or table of acceptable values
- administrative information such as who can access various elements, and information on which applications use specific elements
- and maintenance responsibility for the element.

A data directory is an extension of the data dictionary concept intended primarily for machine processing of the meta data(data which defines data). On the other hand, a data dictionary is intended to aid in human understanding of the data. Applications of the directory concept have included automatic generation of data definitions (schema) for input to various DBMS.

IMPLEMENTATION OF A DATA DICTIONARY/DIRECTORY ON THE HP3000

Most data dictionary systems currently in use appear primarily concerned with aiding the DATA BASE ADMINISTRATOR and other HUMAN data dictionary users. We were certainly concerned with this aspect of the use of the data dictionary, but placed more emphasis on the DIRECTORY side of the concept. The net result of this emphasis shift is that the majority of application programs in the system are DRIVEN by the central Data Dictionary/ Directory.

The Data Dictionary/Directory implemented on the HP contains most of the traditional elements mentioned above, but it does not presently contain security information or pointers to application programs. However some significant additions were made.

First at the element or field level a number of modifications were made.

- KEY fields are noted; this will cause an IMAGE automatic master to be created for it
- On-line as well as background edit specificatiOns were added facilitating editing during data entry or later in background
- A link to a decode table for the field was included,e.g. the decode file table entry name(used as a prefix for its KSAM key).
- Should the field be verified during data entry?
- Should the field be automatically duplicated from a previous record in the data entry form(set of defined records)?

The second and most important set of additions center of the masterfile's structure. This structuring information includes physical RECORD definitions, logical UNIT definition, and when these UNITs occur, or the ENCOUNTER definition.

RECORD content, i.e. which elements make up each 80 byte masterfile record (this is the data entry medium); also where each field starts on the record, and whether it's repeated as a contiguous block or array.

Logical UNIT structure, i.e. which 80 byte records make up a logical UNIT of information (later used as the entity equivalent to an IMAGE data set). These logical units are generally entered together during data entry (called a FORM at data entry). Various key fields appearing on every record aid in the logical connection of records in a UNIT.

These structuring entities are sufficient in most cases to process the data, i.e. define the data entry form, define specifications for the batch editor, and define the schema for an IMAGE data base. However, we've gone one step further in the structure progression - adding the element of time, i.e. when a group of logical UNITs are to occur. This is called an ENCOUNTER definition, corresponding to a clinical visit encounter between a subject and physician. During each visit, various UNITs of logically related data are to be collected, so an encounter consists of a list of logical UNITs. To facilitate automatic assignment of UNITs to a given encounter, an ENCOUNTER CLASSIFICATION EQUATION is included - which defines membership criteria, e.g. VISIT NUMBER=5 or Days into study is between 4 and 7. This structure actually captures the final dimension necessary to fully define a clinical study to application programs which will categorize, inventory and even analyze the body of data.

ACTUAL DATA DICTIONARY/DIRECTORY FILE STRUCTURE

To facilitate most efficient use of the entire system - a true network structure was chosen. As a result, individuals defining a new data dictionary/directory can "point" to any entity already defined, at either the UNIT, RECORD or FIELD level. This automatic GLOBAL capability has greatly enhanced the use of the system. This has also encouraged use of STANDARD UNITs or RECORDs across studies.

The data dictionary/directory system is implemented in a set of variable length record binary KSAM files(except the field level file which is fixed length). Each KSAM file in the set corresponds to a level in the structural hierarchy.

The linkage between the various files in the hierarchy is maintained by a variable length array of pointers with associated information pointing to the next lower level file. The pointers are in the form of KSAM keys into the appropriate subordinate file. Selected information belonging to a subordinate file is often stored along with the pointer to the record(i.e., the Field start column is stored at the RECORD file level). This convention was adopted to facilitate use of global information, i.e. information used by many studies or records, and information thought to change most frequently was stored at a higher level in the hierarchy. As an example, the length, edit specs, label, etc. would probably not change for the field blood pressure between studies. The starting position on the 80 byte record probably would change from study to study, so the starting column was stored at the RECORD level in the

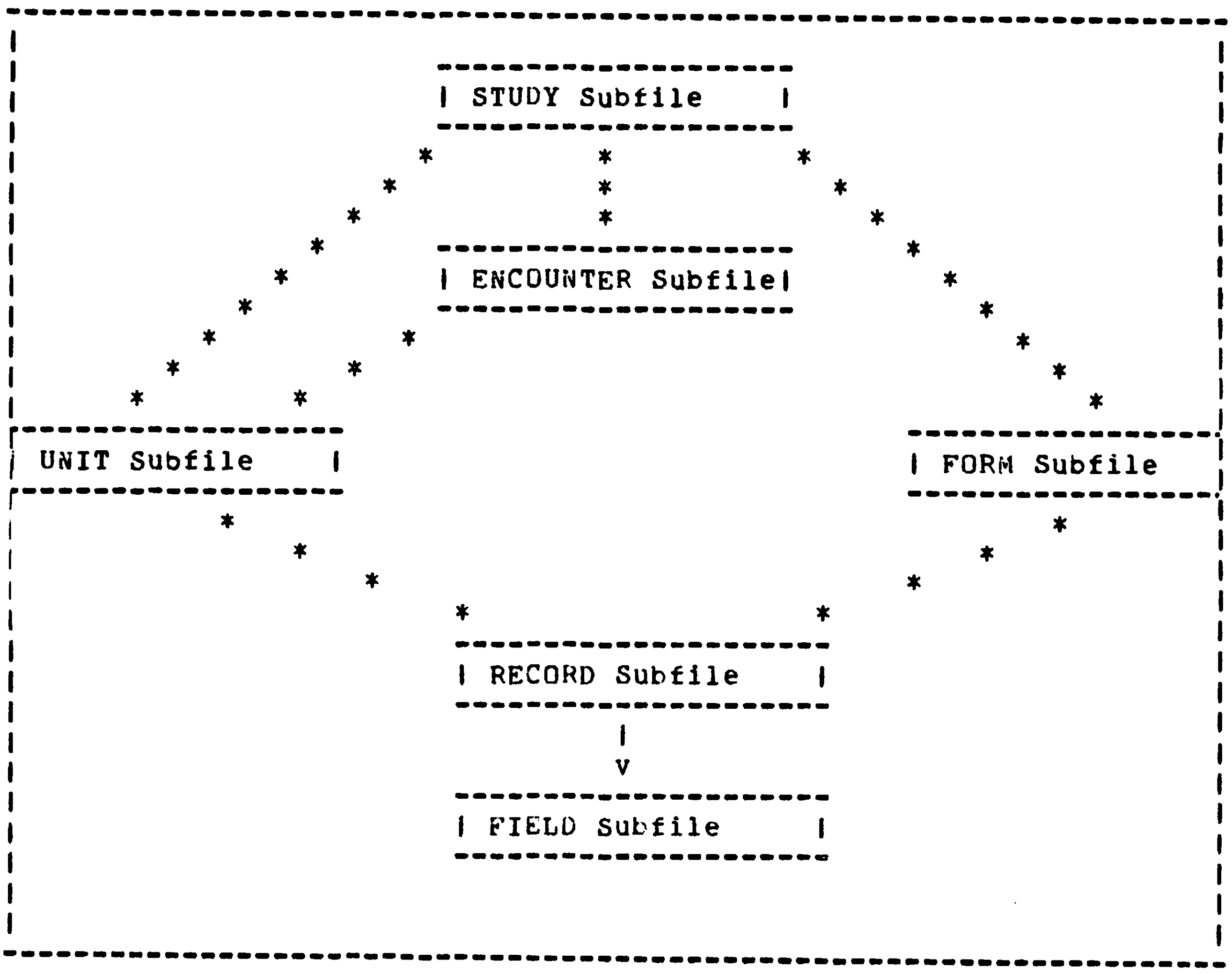

Figure 1. Illustration of the data dictionary hierarchy.

hierarchy, so many RECORDs can "point" to the same common blood pressure definition.

The STUDY level file records contain descriptive fixed length information about a clinical study(masterfile), as well as three variable length pointer stacks. These three stacks contain character type keys to the attached UNITs, the ENCOUNTERs and the data entry FORMs. The three stacks share contiguous space in the record - with the base of the FORM pointer stack just after the UNIT stack and so on. The Data Dictionary/Directory is generally accessed from FORTRAN, so, the fixed length elements of the record can be accessed after a record is read(using FREADBYKEY) simply by an EQUIVALENCE of the items to the LOGICAL type buffer. The stacks are accessed by actually moving a LOGICAL word at a time into a LOGICAL typed TRANSFER buffer which has been equivalenced to a character variable.

The ENCOUNTER file records contain a fixed length portion which includes the primary key, the ENCOUNTER CLASSIFICATION EQUATION and the top of stack counter, followed by a variable portion containing the member UNIT stack. Each entry in the member UNIT stack contains a PL/I like structure (heterogeneous typed structure) containing the UNIT key (character type) and the number of that type UNIT required in the ENCOUNTER(integer type). Each entry in the stack is accessed by moving a word at a time from the appropriate address in the record buffer to a logical type transfer buffer. The character variable followed by an integer variable are equivalenced to the transfer buffer. Note that the encounter classification equation is later parsed then interpreted by various application programs.

The UNIT file records are very similar to the previous level records, i.e. a fixed length portion for UNIT description followed by a variable portion for the attached RECORD stack (also containing the number of each RECORD type required in UNIT). A new value is added to the fixed length portion of the record - the global counter - to aid in controlling use of the global facility. This is present also at the RECORD and FIELD level. The counter is set to 1 if the UNIT record is local, i.e. not pointed to by any other study, but it's incremented every time it's pointed to. This allows detection of its global status(to provide for a warning on the effects of modifying it), as well as providing for local deletes. If an entity is global, and it's to be "deleted", the global counter is decremented, and the pointer to it is removed(the actual record is not removed). If the counter is 1 and it's being deleted - then actually FREMOVE it.

The data entry FORM file is almost identical.

The RECORD file record is basically the same, except that each entry in its member FIELD stack contains a larger heterogeneous typed structure. The structure includes the FIELD's key(character), its starting column (integer), the member of contiguous repeats of the field on the record(integer) and whether it's required (for editing purposes).

The FIELD file record is the simplest - fixed length binary - containing all the basic information such as type, length, description(label), etc.

REASONS FOR USING KSAM/3000

The Data Dictionary/Directory could have been implemented using a set of direct access files, with an index into the top most level (study level) providing a table of contents. Pointers to subordinate entities, e.g. pointers to all attached RECORDs in any UNIT, could have been the actual record address in the subordinate file. This approach would have been very efficient in tree traversal time(data dictionary hierarchy traversal). Problems associated with this approach are:

- MPE presently doesn't support direct access variable length records (variable length records being very important for efficient implementation of the overall structure)
- broken pointers, i.e. bad record addresses are very difficult to fix - probably requiring a dual pointer structure (pointer to son from father and vice versa)
- a garbage collection mechanism was needed - would have been necessary to explicitly set this up
- the global use of UNITs, RECORDs or FIELDs would have become extremely difficult to implement - one would need a separate list of availabe entities with their record addresses.

IMAGE/3000 itself could have been used to implement the data dictionary (something occassionally done in commercial systems). The straight hierarchical structure of any single LOCAL data dictionary would have been implemented satisfactorily as illustrated in Figure 3.

The global entity concept would have been very difficult(impossible) to implement using this structure, though. Also, information such as Field start column couldn't have easily or efficiently been stored at the RECORD level (no variable length records in IMAGE, or compound items as chain heads).

The Data Dictionary/Directory was implemented in KSAM/3000 because:

- variable length keyed-access records are supported, so variable length pointer lists with associated information is possible.
- using keys as pointers rather than actual record addresses greatly simplifies implementation of global entities and security/accuracy of pointers.
- KSAM takes care of garbage collection automatically.

Traversing the tree structure using keys into subordinate files is not as fast as direct access using a record address - but it's close (KSAM takes care of that). More importantly though, this small disadvantage in speed is considerably offset by all other capabilities gained (as stated above).

OVERVIEW OF THE SYSTEM DRIVEN BY THE DATA DICTIONARY/DIRECTORY

The clinical data management system implemented on the HP3000 includes:

- a key-to-disk emulating DATA ENTRY system
- a set of DATA PREPARATION modules, including a batch editor, on-line error corrections, a masterfile inventory procedure and various listing/reporting utilities
- the DATA DICTIONARY/DIRECTORY subsystem itself - containing an interactive formatted screen definition/modification procedure, along with many reporting and auxiliary use

utilities

- and the DATA MANAGEMENT subsystem which contains segments for masterfile tracking(primary residence(IBM vs HP), statistics, etc.), masterfile movement, and an IMAGE schema writer/loader. (Interfaces to IBM resident DBMSs also exist).

All elements of the system are generalized and function for any study based on information gained primarily from the Data Dictionary/ Directory(some modules are implemented using other similarly constructed central files, such as the masterfile tracking file).

DATA ENTRY SUBSYSTEM

This is a key to disk emulating "keypunch" operation which supports an input/verify data entry cycle. Data is keyed on a micro-processor controlled data entry terminal(the standard HP2645A) utilizing a microprogram downloaded from the HP3000. The data is keyed in 80 column card images using card masks or FORMS which are constructed from information contained in the Data Dictionary/Directory. The microprogram allows for automatic tabbing from field to field , automatic as well as manual duplication of fields (from the previous record in the FORM) and automatic insertion of RECORD identification information. The microprogram communicates with the HP3000 in a background mode with buffering and accepts keying in foreground mode. This results in instantaneous response at the keyboard independent of load on the HP3000 - and also reduces the processing drain data entry would normally place on the HP. During on-line verification , the record being keyed is matched against the corresponding record previously keyed during input. Any discrepancies are resolved then.

Completed data entry batches are automatically routed to the appropriate masterfile based on information contained in the Masterfile Tracking file (this includes automatic submission of IBM bound jobs for those masterfiles resident on the IBM).

DATA PREPARATION SUBSYSTEM

The Data Preparation subsystem supports background editing and inventory of masterfiles, interactive error corrections and various listing utilities. All modules of the subsystem gain information from the Data Dictionary/Directory and the Masterfile Tracking Control file.

The batch EDITOR works on a complete masterfile or a selected subset, sorts the file by record id and other key fields, then produces a formatted report. The report is constructed using field names as column headers and edit specifications (acceptable data range) for flagging of data errors - both from the data dictionary. The EDITOR knows where to pick up each field on a record based on the field start column and length information, also in the Data

Dictionary. A similar version of the EDITOR exists on the IBM/370 , written in PL/I, also running off of the Data Dictionary. The IBM resident version is run by jobs automatically submitted from the HP if the Masterfile Tracking file says the file's primary residence is the IBM.

The interactive ERROR CORRECTION procedure is a formatted screen oriented program which directly updates an HP resident masterfile. Masterfiles are maintained as KSAM files on the HP, so a particular record to be corrected can be located very rapidly using FREADBYKEY or FFINDBYKEY. Once desired modifications are made to the record, it's FUPDATEd on the KSAM file. A log of corrections is maintained for backing them out, or reapplying them in the case of system fails.

The batch INVENTORY procedure takes a masterfile, sorts the file by patient number, then performs an inventory of data records comparing the data present to the expected data structure. This expected data structure - which records comprise a unit, and when/how many units are to occur - is constructed based on the Data Dictionary ENCOUNTER and UNIT structuring records. The INVENTORY procedure runs in either a total inventory listing mode or in an exception mode - listing only patients with missing records.

This subsystem also includes maintenance and reporting facilities for a Decode file - a KSAM/3000 based data decoding table. An ISAM version of this decode file is maintained on the IBM/370 through the HP resident maintenance system - any updates made to the HP file cause a job to be automatically submitted to the IBM for identical updates to be made there.

DATA DICTIONARY/DIRECTORY SUBSYSTEM

The Data Dictionary/Directory subsystem includes an interactive formatted screen oriented DD/Dir definition and modification program, automatic coding manual generation, a data entry FORM definition interface, various reporting utilities and a translate/ export module to translate the Data Dictionary into its IBM/370 version and export a copy of it automatically.

The definition/modification program uses a formatted screen (not in block mode - but under program control) which is set up with an indented hierarchy corresponding to that of the data dictionary. The formatted screen is traversed in a manner corresponding to the tree structure of the data dictionary. When changes are to be made, the user simply defines the path down the tree to the item to be changed, then enters the information to be modified.

Reporting utilities exist which produce a comprehensive list of any Data Dictionary's content, study encounter structure, as well as a coding manual to aid data coding (prior to data entry).

The translate/export module traverses any data dictionary, produces a sequential version of it and ships this to the IBM/370 where it's used by many IBM resident DBMS interfaces (as well as the batch editor).

DATA MANAGEMENT SUBSYSTEM

The Data Management subsystem includes the Masterfile Tracking control file maintenance/report utilities, masterfile movement utilities and an IMAGE schema writer/loader. The entire system is controlled by a top level program in the Data Management system - called the Utility controller - which runs as a very high level language interpreter.

The Utility controller program of the subsystem accepts English like commands, parses them, and takes action on them by calling one of many programs in the system (using the MAIL intrinsics to pass information to the called programs).

The Masterfile Tracking control file contains information on the primary residence of all masterfiles, predicted number of records in the masterfile, and a variety of other accounting/security types of information. In addition, information on the existence of an IMAGE data base, or any other IBM resident DBMS, and the date that it was last loaded, is contained in this file. This information is very useful, not only to tell how current a data base is, but also, whether it already exists, and should be purged before being created(recreated).

Masterfile movement utilities are also available as a part of the Utility controller. The command MOVE HPD<masterfile> TO OS<dsname> is sufficient to prompt the system to automatically submit a job to move the masterfile named to an IBM/370 resident data set(building it if necessary). The reverse command is possible, and in fact, movement between any of 4 logical origins/destinations is possible. The jobs to move the masterfiles in any direction are submitted automatically to MRJE/3000.

The system includes interfaces to many IBM/370 resident DBMS packages, and to the IMAGE/3000 system. The IBM resident DBMS interfaces all use the Data Dictionary to create the equivalent of an IMAGE schema, then later in the same job, actually load the masterfile desired into the DBMS. The IMAGE/3000 interface submits a background job to run 2 programs (with standalone sorts between) which write a Schema, then actually load the data into the created database. The details of this set of programs are covered below.

IMAGE/3000 SCHEMA WRITER/LOADER

The IMAGE Schema Writer is an interface program which reads the data dictionary, and primarily constructs appropriate schema definition statements from the data FIELD types. The procedure will create a schema containing all UNITs of a study or only selected UNITs depending the request made to the front end program (run by Utility controller).

Total output from the program includes the completed schema ready for processing by DBSCHEMA.PUB.SYS, a detail load map, and a summary or data set load map. The schema is produced by building two intermediate files - one for the SET part and one for the ITEMS part of the schema. As the data dictionary is traversed - accumulating all FIELDs belonging to each UNIT, the schema ITEM part for each field is written (knowing what type the FIELD is, its size, etc.). At the same time, a posting to the SET part is made for the FIELD. Each Data Dictionary UNIT is considered to be equivalent to an IMAGE Data Set, so whenever a new UNIT is started in the traversal operation, the appropriate SET definition is written, followed by its FIELD or item entries. The buffer position for the Loader procedure is also calculated for each field in each data set. The detail load map file generated by the schema writer program consists of a record for each Field in the data dictionary containing the field's input buffer address, the IMAGE data type, the field's parent Record Id, starting column, length, and decoding information. This information is sufficient for later posting of the data to the data base. The summary load map produced contains an entry for each data set being created, with the number of fields in the data set, the total buffer size (used for acquiring the proper size extra data segment in the Loader procedure), and the name and sequence number of the data set.

After the Schema writer produces the Schema, Detail load map and summary or data set load map, the DBSCHEMA.PUB.SYS program is invoked to process the new schema(FILE equation being issued by the schema writer). After the root file is created, the DBUTIL.PUB.SYS program is executed, to create the new data base. Finally the detail load map is sorted by Record id, then start column and the masterfile is sorted by patient id, then record date, then record id. The Loader procedure is now ready to run.

The Loader program reads the sorted Detail Load map, and constructs a tree structure containing all information to convert/decode from the masterfile records into a logical buffer to be passed to the IMAGE data base. The procedure also reads the Data Set load map, and creates an extra data segment of proper size to act as data accumulation areas for each of the Data Sets. The program then begins reading the masterfile and converting/moving the data to the appropriate extra data segment location. When a logical breakpoint in the data occurs(change of patient, or change of date), all extra data segments which have been posted, are accessed, posting their contents to the IMAGE data base. Various safety checks are built in to prevent overwriting of the extra data segment buffers, i.e., if a record member of a buffer has already been posted, and it's about to be posted again, the buffer is cleared first (this should not happen normally due to the UNIT structure of the Data Dictionary).

During the loading process, blank or missing data is noted, by posting a bit in a MISSING field(a field which is automatically generated for each data set) which corresponds to the sequence number of the field in the data set. This MISSING bit is to be used by various application programs accessing the data base to ensure proper statistical treatment of missing data. Each UNIT occurrence is also categorized into one of the ENCOUNTERs of the Data Dictionary, with the categorization being also posted to an ENCOUNTER field in each data set(this aids in data retrieval later). Data which is to be decoded for reporting purposes, is decoded during the load operation - the Decode table field name and field value (or encode) are concatenated, then used as a KSAM key into the decode file - retrieving the text to be inserted into the data base.