The role of Data Dictionaries in Application Development, with an Emphasis on System Dictionary.

Raymond Ouellette
Infocentre Corporation
3100 Cote Vertu
St. Laurent, Quebec
Canada H4R 2J8

The idea behind a data dictionary is easy to understand. You create a description of the data available on your computer system and store it on the computer itself. In this way, programmers (or programs) can find out exactly what information is supposed to be on the machine and where and how to find it. This results in less error caused by confusion over the format of the data and makes it possible for end users to create reports without prior knowledge of the structure of the data.

Unfortunately the realities of implementing a working data dictionary environment are much more complex than the basic idea would suggest. It is therefore advisable to understand the general principles of data dictionaries before closely looking at a particular product.

This discussion of data dictionaries is in three parts.

Firstly there is an analysis of the benefits possible when a dictionary is available and the type of information which needs to be stored in order to achieve these benefits.

This is followed by a description of the problems which must be considered before implementing a data dictionary in a realistic environment.

Finally the ways in which System Dictionary can address these subjects in the HP3000 environment are discussed.

Audience level of my abstract: 3+ years.

This paper would best fit in track 3.

Introduction

Data dictionaries provide a means by which we can manage information. Dictionaries are simply a tool that facilitate the management of data, and the conversion of data into corporate information.

An effective implementation of a data dictionary can help manage this critical corporate resource. An ineffective implementation will hinder more than it will help.

Implementing a data dictionary is a major undertaking. A great deal of analysis, design and planning is required to set standards and procedures regarding the role and use of the dictionary. A number of technical and operational issues must be addressed, such as: dictionary maintenance, security, version control, and deciding on the number of physical dictionaries to be implemented.

The strategy chosen may be different for each of several uses of the data dictionary. The dictionary may be called upon to serve different roles in application development, end user computing, in conjunction with purchased application packages as opposed to in-house systems.

Using Hewlett-Packard's *System Dictionary* as a point of reference, let's concentrate on application development, and examine an approach to effective dictionary implementation.

1.1 Objective of Data Dictionaries

Data dictionaries are generally used for a combination of the following functions.

1) Centralized documentation of the data and programs on a computer.

2) Storing physical file specifications and record layouts so that programs always address the data correctly.

3) Storing logical attributes of the data which can be used to generate programs automatically.

4) Storing descriptions of the data on a computer which enable end users to generate reports.

1.1.1 Documentation

In the old days, all good system analysts used to fill in special forms which defined the record layouts for all of the files in the systems they were designing. Each programmer was given a copy of any layouts which affected his/her programs so that there could be no chance for confusion. Of course, in practice, the analyst would change the layouts quite regularly with the result that each programmer would end up with a different version of the file specifications.

The idea of data dictionaries emerged as a method of using the computer to ensure that documentation is always up to date.

Data dictionaries are not, however, the only possible solution to the problem. One of the most useful features of modern database systems is that they usually include some form of self-documentation of their structure. For example, all the vital information concerning an IMAGE database can be found by running QUERY and using a simple command. To some extent this has delayed the necessity to introduce data dictionaries since storing information in a dictionary is rather pointless if the same information is easily available to everyone anyway.

However, it is unlikely that any database system will ever be able to contain enough information about itself to make a dictionary totally redundant. An important function of a data dictionary is to keep a record of which programs process the various databases and data elements on the system. This information is particularly vital for finding which programs will be affected by changes to a database but cannot be obtained directly without checking every program on the system indivually.

Whereas the definitions of individual entities in a system could theoretically always be stored within the entity itself, the only way to effectively document the relationships between the elements is in a dictionary. Unfortunately, this sort of information is very difficult to maintain correctly since it is not easy to ensure that all of the relationships are actually included in the dictionary. If

there were some automatic way of verifying that the dictionary were complete, there would be no need for the dictionary in the first place!

1.1.2 Storing physical file specifications

The most immediate benefits of a data dictionary almost always come from the ability of programs to extract physical data attributes directly from the dictionary.

This fact was realised long ago when it became popular to establish COBOL COPY libraries containing the record layouts of the files on a system. Whenever a COBOL program was compiled the relevant layouts were extracted from the library by the compiler. This freed the programmer from worrying about the hand-written record layouts mentioned above.

The system worked very well but was obviously limited by the restriction to a single language and by having to recompile all programs every time the library changed. Actually compiling the programs was no real problem but remembering which programs needed to be compiled was much more difficult.

With a proper data dictionary it is possible to extract file specifications from the dictionary every time the program is run so that any changes in the dictionary are always reflected automatically without altering or recompiling programs. To anyone who has used COBOL with copy libraries on a large system this sounds marvelous but as we shall see run-time access to a dictionary introduces all sorts of other problems.

It is worth noting that database systems such as IMAGE have to some extent reduced the benefits of extracting physical attributes from a dictionary since the same information is readily available from the database itself. In fact, it would probably be true to suggest that a program or compiler should never obtain physical data attributes from a dictionary. In the ideal world a database should, at the very minimum, hold a complete description of its physical structure and all programs should use this description to get at the data.

1.1.3 Logical Data Definitions

As well as storing a description of how the data is physically held on a computer we can also include the logical attributes of the data such as standard heading text or edit masks for data elements in the dictionary.

It is very important to understand the difference between "physical" and "logical" attributes.

As an example, consider the description of an IMAGE item in a dictionary. The physical specification defines the length and type of the item. These are physical facts which can never be sensibly contradicted by any program. The logical attributes, however, are not absolute and can often be altered for an individual program without obtaining invalid results.

In some cases the logical attributes are merely suggestions for programmers who need not take any notice if they do not wish to. In other cases, the attributes may represent "standards" which a program must follow unless there are good reasons for not doing so.

Just as the physical attributes of a database can be included in the database itself, there is no reason why some of the logical attributes could not also be included. But whereas the physical attributes of a database are finite, the possible logical attributes are limitless and for this reason it is very difficult to devise database systems which can fully cope with logical attributes.

Physical attribute definitions can be safely extracted from a dictionary automatically during program development without necessarily informing the programmer what is happening, however for logical attributes, the programmer must be given the opportunity to ignore the attributes if he/she wishes.

In the context of application processing, three methods for extracting logical attributes from a dictionary are possible:

1) Run Time. The logical attributes are freshly calculated every time a program is run.

2) Compile Time. Alterations to logical attributes in the dictionary will take effect only when a program is recompiled.

3) Development Time. The logical attributes are copied into the programmers code automatically when the code is initially developed. Alterations to attributes are not reflected in existing programs unless the programs are actually altered.

In fact, it is not likely that we would wish for logical attributes to be decided by a production program at run time. The effects of changing the edit mask for an item might be disastrous on a report where the print positions had been carefully counted by the original programmer.

The same sort of problems also arise when attributes are extracted automatically at compile time. Programmers are forced to test their programs everytime they compile them even if they haven't actually changed the code.

1.1.4 Describing Data for End Users

Most application systems offer the user some helpful information about how to operate the system and what data presented on screens or in reports represents.

A user trying to work with a report generator is not usually so lucky. The report generator itself will of course be able to explain how to create reports but will not know anything about the data being inspected or what it means.

Users can only look in the dictionary to try to find out what there is in the databases available to them. It is quite possible that the physical and logical

attributes described in the dictionary, together with the documentation intended for the computer department, will be enough to get the intelligent user started. However, the user needs information about the data which is of no use to the computer staff and will not exist in the dictionary unless it is put there specifically for this purpose.

1.2 Problems Associated With Data Dictionaries

Having looked at the uses of data dictionaries we now concentrate on the difficulties which inevitably arise when attempting to actually take advantage of the benefits.

These problems are described under the following headings.

1) Standardisation

2) Security

3) Conflicts between Applications

4) Version Control

5) Administration

6) Prototyping

7) System Performance/Reliability

8) Proliferation of Dictionaries

1.2.1 Standardisation

It is extremely unlikely that there will ever be a single standard for data dictionaries even for one particular computer. Even if a such a standard were to emerge, there would be features missing which someone would need to use.

Suppliers of 4GLs often provide their own data dictionary for use with their product and are reluctant to base their language around any possible "standard" dictionary which does not really fit their needs anyway.

The result is that there are often several varieties of dictionary on the same machine all containing the same information but in a different format. Often the dictionaries deteriorate from their role as a centre for data definition into files which have to be maintained simply because 4GLs will not run if they are not present.

1.2.2 Security

A data dictionary contains some extremely important information and a lot of people are going to be legitimately inspecting and altering its contents.

The result of someone accidentally changing something when he shouldn't can be extremely traumatic, especially if programs are accessing data definitions at run time.

A dictionary therefore requires a security system which permits people to access what is available to them but can stop anyone going where they are not allowed. In fact, the security requirements for a data dictionary are much more intricate than we would expect from a typical application system.

1.2.3 Conflicts Between Databases

On large computer installations where several different databases are present, it can be difficult to represent all the databases conveniently in a single dictionary.

It is possible that each database uses a different name for what is essentially the same thing or uses the same name for completely unrelated entities. Somebody has to take the time to sort out the conflicts and however this is achieved, the resulting dictionary is likely to be very confusing.

This problem is most likely to occur on sites which are trying to establish a dictionary after many years of working without one. If a dictionary is used from the start the need to avoid conflicts between databases can be turned into an advantage.

1.2.4 Versions

It would be nice if it were always possible to establish a dictionary which never changed once it had been set up. Unfortunately, computer systems usually develop even after they are "live" and the dictionaries must also change.

When a change is required in the dictionary it is necessary to create a new version or copy of the dictionary so that the amendments can be tested while the old version is still in use. Once the new version is tested and the relevant program alterations have been implemented and tested, the old dictionary can be replaced by the new copy at the same time as the new versions of the programs are moved into production.

Problems start when several unrelated amendments to a system are being implemented at the same time and each programmer makes his own version of the dictionary to test his changes. We have to be sure to control these test versions of the dictionary very carefully and ensure that when a test version becomes the production version, it actually includes all of the changes which may have been implemented since the copy was originally taken.

A possible solution to this is to only allow one test version to exist at any given time. However, this leads to a situation where there is always something in the test dictionary which is not actually working yet and so the dictionary can never go live. In order to actually get the dictionary into production new developments must be suspended until current work is completed.

1.2.5 Administration

Any data dictionary needs an administrator to keep control of the contents of the dictionary.

Without such control, a dictionary is liable to degenerate to the lowest state which is capable of supporting the 4GLs which use it. It will probably become cluttered with definitions which are not actually used but cannot be removed for fear that they are.

The administrators main job is to ensure that the contents of the dictionary are complete and correct. Given the diversity of information in the dictionary and the range of people who will use it, this is not a simple job and usually requires a dedicated (and expensive) individual.

1.2.6 Prototyping

Contrary to accepted opinion, using a data dictionary makes system prototyping very difficult if the designer has to enter definitions into the dictionary before he can get anything working. The modern approach to prototyping which involves the programmer and end user working together is strongly inhibited when the user cannot request immediate alterations and additions to data specifications.

For successful prototyping with a dictionary, we need a development system which can make changes to the dictionary immediately during the design process or can temporarily function independently of the dictionary during the prototyping phase.

Both these methods deviate from the standard approach of most 4GL systems which regard the data dictionary as a relatively static predefined source of information.

1.2.7 System Performance/Reliability

As we have seen there are an enormous number of potential users for a data dictionary ranging from production batch programs attempting to obtain the attributes of a file to end users trying to generate their own reports. At the same time we also require that the dictionary should have a sophisticated security system and be accessible in a friendly interactive fashion.

However good the software, there are bound to be problems with the speed and reliability of such a complicated system.

Since everyone on the computer is theoretically connected to the dictionary, there will be a bottleneck as all of the various programs atttempt to extract information. Worse still, a failure in the dictionary will bring the whole computer to a halt.

Careful consideration must also be given when taking backups of the dictionary or bringing a new version into use. These will be operations which may require every single user on the machine to stop working and, on larger sites, this may not be feasible.

Obviously, these problems can be reduced by ensuring that the data attributes are extracted at compile time so that production systems do not access the dictionary. Even so, program development and end user reporting systems will still be vulnerable to weaknesses in the dictionary software.

1.2.8 Proliferation of Dictionaries

A simple solution to the many of the inherent drawbacks of data dictionaries is to create several dictionaries rather than a single master dictionary.

Instead of keeping a centralized dictionary it may be better to provide a separate dictionary for each application on the computer. A special dictionary for the end user report generator would also normally be appropriate in this case.

Some control over the creation of dictionaries must be maintained because if things get out of hand, it is likely that there will be a separate dictionary for each program and each user on the system. Programmers may even keep dozens of versions of different dictionaries on tapes in their desks!

Once this happens, the basic advantages of centralized documentation are lost. There is no longer a single place which gives the correct definition of the data and although the individual dictionaries may be useful for the functions that they support the need for a "master" dictionary will inevitably arise.

2 SYSTEM DICTIONARY

System Dictionary is much more than a simple data dictionary since it is intended to be used for many purposes other than the storage of data definitions. We shall, however, concentrate on its role as a data dictionary in this paper.

2.1 The System Dictionary Database

System Dictionary is basically a database designed specifically to store information about a computer system. As its name suggests, System Dictionary is intended to be a central database and it is not expected that there will be vast numbers of dictionaries on a single machine. It may be that the proliferation effect will eventually overtake the initial aims but it is evident that many of the features of the dictionary are intended to prevent this happening.

For readers familiar with IMAGE, a brief comparison of an IMAGE database with System Dictionary is a good introduction.

The System dictionary contains 'entities' and 'relationships' which are very roughly equivalent to the master and detail records in an IMAGE database. The entities and relationships have 'attributes' which are like the fields of an IMAGE

dataset.

The key to an IMAGE Master set may be any length or type, but access to an entity type in the System Dictionary is always achieved through a a 32 byte 'key'.

The names of relationship types in the dictionary are always formed from the names of the entity types which they relate. This is equivalent to suggesting that detail sets in an IMAGE database should contain the names of the master sets to which they are chained. For example, a detail set representing an order line on an invoice would be called something like "ORDER contains STOCK-ITEM" if it were transformed into an equivalent relationship type in a System Dictionary. This is more descriptive than "ORDER-LINE" (or even worse "ORDER-DETAIL") which is the name usually selected for this purpose.

Finally System Dictionary allows variable length attributes to be assigned to entities. Variable length fields are not supported by IMAGE and most modern relational databases but this capability is vital for databases which are to be used as a dictionary.

In summary the following expressions are roughly equivalent :

IMAGE	SYSTEM DICTIONARY
Master Dataset	Entity Type
Master Record	Entity
Detail Dataset	Relationship Type
Detail Record	Relationship
Field	Attribute

This new terminology may seem irritating but for once there is a good reason for introducing new jargon. Remember that the dictionary will be used to store data about data. We will have entity types called "RECORD" and "IMAGE-DATASET" etc. and we can at least avoid some confusion by using new words.

2.2 Features of System Dictionary

In order to make the System Dictionary database suitable for use as a dictionary several special features have been included which would not normally be expected in a more conventional database system.

2.2.1 Extensibilty

Unlike IMAGE, there is no schema for a System Dictionary. It is fully flexible so that entity types and attributes can be added or altered at any time. When a new dictionary is created (using the utility SDINIT) it always contains a standard set of entity and relationship types called the core set. You can then customize the

dictionary for your own needs by adding new entity and relationship types or changing the attributes of existing types.

The motivation for providing this exstensibilty in the dictionary is to overcome the problem of standardisation which forces software suppliers to produce their own dictionaries. Since the System Dictionary can be customized, it is not necessary to rely on the original specifications of the core set and software suppliers can add new features to the standard dictionary if they wish.

In reality, there is still great pressure on everyone to conform to the accepted standards. It takes some courage to invent a completely new entity type especially if it is likely that several other people will do exactly the same thing but use a different name.

2.2.2 Programmatic Access

A command driven utility program is automatically supplied, to serve as a user interface to the dictionary. This interface can be augmented or replaced with user written programs that access the dictionary programmatically via a set of documented intrinsics. In this way, special purpose user interfaces or utility programs may be written with the capability to access the full set of dictionary functions. This enables dictionary users to create their own customized interfaces to the dictionary.

2.2.3 Security and Scopes

Any user or program which opens a dictionary must specify a SCOPE and the relevant password before access is granted. The scope name is like a user name which everyone has to provide when opening the dictionary in the same way that as everyone has to give a name before logging on to MPE. Entities and relations are always accessible to the scope which created them but can be secured against read or modify access by other scopes.

Note that security works at the entity level. This is equivalent to being able to secure individual records in an IMAGE database.

2.2.4 Domains

A single system dictionary may be split into several domains which are effectively "logical" dictionaries within the same physical dictionary. This is intended primarily for situations where many applications run on the same computer but have very little else in common. Each domain behaves as an individual dictionary and functions independently of the other domains which reside in the same physical dictionary.

The aim is to avoid needless conflict while still keeping all the domains under the control of the same dictionary.

All dictionaries initially contain a single domain called the common domain and new domains are created as required by the dictionary administrator.

2.2.5 Versions

System Dictionary recognises the need for different versions to exist at the same time and permits creation of many versions of each domain within a single dictionary. The versions are labelled "test", "production", or "archive" and the software prevents you from updating production or archive versions.

This method of creating versions within the same physical dictionary ensures that programmers cannot simply use COPY to create new versions at will but the administrator must still overcome the inherent problems of version control described earlier. System Dictionary allows several test versions to co-exist and he must always ensure that when a test version becomes a production version it includes any updates which have taken effect since it was originally created.

2.2.6 Aliases

An alias is an alternate name for an entity or relationship which is to be used instead of the actual name in a particular situation. The most common type of alias is the IMAGE-ALIAS for an entity which will be the name to be used when the entity appears in an IMAGE database.

Typically it is preferable not to use aliases but there are circumstances especially when databases have already been designed with no consideration for System Dictionary, when it is appropriate.

For example, we may have a database where item names have been prefixed for some reason so the item name for customer number is A100-001-CUS. This is not really a suitable name for the element in the dictionary and it would be better to call the element CUSTOMER-NUMBER and include the actual item name as the IMAGE-ALIAS.

2.2.7 Synonyms

A synonym is an alternate name for an entity which is used for a completely different purpose to the aliases. If the actual name of an entity is long, it is nice to be able to supply a short name which may be used by anyone who accesses the entity regularly.

If an element were called "INDIVIDUAL-CUSTOMER-NUMBER", the administrator might supply "ICN" as a synonym to save typing the full name. Any number of synonyms may be given to a single entity.

2.2.8 Internal and External Names

Every entity type and entity in the System Dictionary has an internal and an external name. Normally these names are the same but it is possible to alter the external names of entities or entity types.

When a program opens the System Dictionary, it specifies whether it will use the

external or internal names to access the dictionary. By using internal names, the program can ensure that the names in the core set have not been changed. This mode is intended for standard software which will operate on many different sites.

Programs developed for a particular dictionary can use the local external names which will be known to the users of that particular dictionary only.

2.3 Representing IMAGE and MPE Files

Although the System Dictionary is totally flexible, the core set of entity and relationship types imposes standards on how the structure of IMAGE databases and MPE files should be represented.

The following list shows the most important entity and relationship types from the core set which are used for this purpose.

Entities:

ELEMENT
RECORD
IMAGE-DATASET
KSAMFILE
FILE
IMAGE-DATABASE

Relationships:

RECORD contains ELEMENT
IMAGE-DATASET contains RECORD
KSAMFILE contains RECORD
FILE contains RECORD
IMAGE-DATABASE contains IMAGE-DATASET
IMAGE-DATASET key ELEMENT
KSAMFILE key ELEMENT
IMAGE-DATASET IMAGE-DATABASE chains

2.3.1 Representing an Image Database

Each item in an IMAGE database corresponds to an entity with type ELEMENT in the System Dictionary. Although there is no single attribute which defines the IMAGE item type, the attributes COUNT, ELEMENT-TYPE and BYTE-LENGTH can be combined to form the IMAGE type. Other attributes of the elements such as DISPLAY-LENGTH, DECIMALS, EDIT-MASK etc. enhance the item specification beyond what is included in the IMAGE schema.

The datasets are represented by the entity type IMAGE-DATASET which has an attribute called IMAGE-DATASET-TYPE to specify whether the set is a master or a detail. It may seem reasonable to expect a relationship type called 'IMAGE-DATASET contains ELEMENT' to be used to assign elements to the datasets. In fact, we need to create another entity of type RECORD and the elements are associated with this entity using the relationship type 'RECORD contains ELEMENT'. The start position of each element within the record is indicated by an attribute of this relationship. The dataset is then linked to the record by establishing a relationship of type 'IMAGE-DATASET contains RECORD' between the dataset and the record.

The database itself is represented by an entity called IMAGE-DATABASE and the datasets are assigned to the database using the relationship type IMAGE-DATABASE contains IMAGE-DATASET.

Finally we need to indicate the keys and chains in the database. For master sets the relationship type 'IMAGE-DATASET key ELEMENT' is used to define one element which is the key to the dataset. The chains to a detail set are represented by a complicated relationship which links five entities and specifies the dataset, search element, sort element, master dataset and database involved in the chain.

There are other entity and relationship types in the core set concerning security classes and the devices used to store datasets but these are not described here.

You will notice that the method for representing a database is fairly flexible in that it permits us to assign datasets to more than one database and to assign records to several datasets. This could be useful in situations where two databases contain copies of the same dataset or where two datasets in a database have the same fields. It remains to be seen whether it will become common to take advantage of this flexibility or whether people will prefer to create a separate entity for each dataset.

2.4 Maintaining the System Dictionary

Several tools are available for maintaining the contents of the system dictionary.

Firstly there is a utility called SDMAIN which is a command driven tool for directly accessing the dictionary. There are commands for adding, amending or deleting entities and relationships as well as facilities for achieving

administrative functions such as customisation or creating new versions and domains.

The big disadvantage with SDMAIN is that it is cumbersome to use and extremely unfriendly. It achieves for System Dictionary what QUERY does for IMAGE.

Another method of setting up a dictionary involves running a standard utility program called SDDBD which loads the format of an existing IMAGE database into a dictionary. The resulting definition only includes information which can be extracted directly from the database so logical attributes such as standard headings or edit masks are not loaded.

3.0 Utilizing the Dictionary in Application Development

Having identified the objectives of data dictionary use, and some of the common pitfalls lets examine the role a data dictionary can serve in an application development environment, using System Dictionary as a point of reference. When integrating application development with a data dictionary, we should be careful to capitalize on the strengths of the dictionary while avoiding its weaknesses, such that we use the tool effectively. In this context two features of System Dictionary are particularily intruiging: programmatic access, and extensiblity.

Programmatic access means there is a set of intrinsics making it possible for a program to open a dictionary, read information from it, update information and so on. This may not be of practical importance to many HP3000 shops, however it presents an opportunity to software suppliers to interface *application development* software with the data dictionary.

This opportunity is made even more attractive by the capability to extend the structure of the dictionary. System Dictionary can be customized to fit the needs of any application development effort. It does not have to be language, application, or vendor specific. The possibility of having one centralized system wide dictionary becomes a feasible reality.

To picture this opportunity envision your application development tools (text editor, flow charts and diagrams, COBOL generators, COPYLIBS etc.) being replaced by sophisticated application generator software. This software becomes the analysts' workbench, and it is actively involved in the definition of all application system processing; ie: data entry and inquiry screens, reports, batch processing, command procedures, security, and menus. The software also develops and stores the underlying file structures (IMAGE, KSAM, MPE), and generates source code for the application as a natural result of the development effort.

Application generator software represents the state of the art in fourth generation development tools. Along with tremendous increases in system development productivity, it also brings changes to the roles of application developers, end users, and the data dictionary.

With the application generator automating the detailed programming tasks, analysts are able to focus their energies on system analysis and design. Users can

actively participate in the design phase, assisting the analyst in prototyping sessions made feasible by the capabilities of the software.

As one proceeds through the design and construction of an application, the system generator can be in constant communication with the data dictionary, offering lookups to existing entity definitions, as well as the opportunity to load new entities and relationships into the dictionary.

With this approach we create a symbiotic relationship between our application development tools and the data dictionary. When working with the application generator we retain full functionality of its native operating characteristics, but at the same time, avail ourselves to the centralized store of existing data documentation. The data dictionary can be maintained and updated automatically in a consistent fashion by the application generator.

In this role, the data dictionary is accessed at system development time, offering time saving assistance to the system developers. Existing data definitions can be extracted as programs are developed. Not to act as a constraint however, we can also create new entities for this application as needed. This is crucial in order for prototyping activities to succeed. When the development project is completed, the software can update the dictionary automatically, loading the definitions of any new entities created for the application. We can also load new relationships into the dictionary, identifying the data entities accessed by this application. These "where used" relationships will facilitate impact analyses required when changes to data definitions are contemplated.

The programmatic interface between the application generator and the dictionary can be implemented with the following objectives in mind:

1) The existence of the dictionary should be reasonably transparent to the average user. The dictionary is there to assist in the development effort - we should not be unduly tied to it or restricted by it.

2) Since the application generator holds all of the specifications of our application - both data and processing - it should be responsible for loading definitions into the dictionary.

3) The presence of the dictionary should not discourage or hinder application prototyping.

4) The application generator already maintains information about the application. There is no need to duplicate this information in the dictionary, unless it would be of use in the development of other applications.

3.1 Accessing System Dictionary

When contemplating how the dictionary can best serve our interests in application development, we must consider two general situations with respect to our data:

1) The application database(s) already exist, and are defined in the dictionary.

2) The application database(s) do not yet exist, and are not defined in the dictionary.

The term *database* as used above, refers to the aggregate of data processed by the application. This data may reside in one or more Image Databases, KSAM, or MPE files. The two general situations outlined above may be combined to formulate additional situations; eg: some of the files already exist, but some do not; of the files that do not yet physically exist, some of them are defined in the dictionary; etc.

How we approach these situations with respect to dictionary usage impacts on our development methodology.

If we adopt an approach that demands all data elements be defined in the dictionary prior to being referenced in the creation of an application program, then prototyping will be very effectively stifled. This approach requires a development methodology that begins with a rigorous definition of all of the application data. These definitions would be held in the dictionary and be accessible during program development. This latter activity of designing the application processing would need to proceed in a very predetermined fashion.

A development methodology centered around prototyping relies on the ability to draw on existing definitions, amend existing defintions, and create new definitions, of both data and processing, throughout the design phase. With the proper tools and expertise, this methodology can result in an application engineered to the customer's needs.

A flexible approach to interfacing with the dictionary can accomodate either development methodology. Accessing the dictionary can be approached this way:

1) Existing data definitions can be extracted from the dictionary as needed, during program development. This applies to data structures (Databases or files) that already exist, as well as new data structures that we need to create. Consider that for a new file or Database under development, individual field definitions or complete record layouts may already be defined in the dictionary, belonging to another file or database.

2) New data definitions can be created "on the fly", as needed. This is required in order to undertake effective prototyping. The new definitions would initially be stored locally, specific to this application. At some point in time, these new defintions can be uploaded to the dictionary.

Once the application is complete, it can be loaded into the System Dictionary so that a complete list of the entities processed by the application is available.

3.2 Customizing the Dictionary

A certain amount of customisation may be required to a System Dictionary for it

to suit our application development purposes. This customization can be undertaken by using the "extensibility" feature, adding additional entities and relationships to the core set.

Firstly, we may have a need for additional logical attributes, describing individual data elements. The core set already provides several logical attributes such as DISPLAY-LENGTH, DECIMALS, and EDIT-MASK. Depending on your needs you may wish to extend this list with other attributes like MATCH-PATTERN for example.

Another useful attribute can be attached to a number of entities, marking them "Private" to this application. Elements or Records marked as "Private" would not be accessable to other applications. Once the development effort is completed, new entities created by the application can have their definitions marked "Public", and hence be available to other development projects.

A number of customized relationship types might also be in order, such as:

* SYSTEM processes ELEMENT
* SYSTEM processes RECORD
* SYSTEM processes IMAGE-DATASET
* SYSTEM processes IMAGE-DATABASE
* SYSTEM processes KSAMFILE
* SYSTEM processes FILE
* SYSTEM owns ELEMENT
* SYSTEM owns RECORD
* SYSTEM owns IMAGE-DATASET
* SYSTEM owns IMAGE-DATABASE.

The first group of relationship types are used to identify the entities which are processed by a particular application. The second group is used to indicate which application was responsible for creating the entities in the database. Later on, these relationships can be extracted from the dictionary, providing valuable data administration information.

3.3 Loading Applications into the System Dictionary

At any time during development of an application, it should be possible to load the application into the System Dictionary. Definitions of new data elements created for this application must be loaded, as well as a complete set of relationships, such as: which application created this element, which application processes it.

The loading process is very important. It constitutes the updating of a valuable corporate resource, and as such should be controlled under the data administration function. It makes sense then to approach the loading in a batch fashion, at the conclusion of the development project.

Having the application generator accomplish the loading of the dictionary is one of the more important benefits of integrating development tools with the

dictionary. It ensures that the dictionary will be updated automatically, consistently, and accurately.

3.4 A Summary of the Dictionary Interface

The most important aspects of the dictionary interface can be summarized by the following points.

1) There is no run-time access to the dictionary. The interface is entirely in the application generator module.

2) The presence of the data dictionary does not impose a development methodology. One can choose to use the dictionary or not use it, draw existing defintions from the dictionary, create new definitions as needed.

3) Once an application is complete, the following information can be loaded automatically into the dictionary.

   i) A complete definition including logical attributes of the data accessed by the application.

   ii) Relationships which specify the databases, datasets, files and elements processed by the application.

Summary

A comprehensive dictionary product, such as Hewlett-Packard's System Dictionary, combined with sophisticated application development tools can go a long way toward solving a number of traditional dictionary implementation problems.

The features of System Dictionary make it possible to implement a single, system-wide dictionary database. (What a data dictionary was meant to be in the first place!) The dictionary administrator is provided with the necessary tools to address security and version control issues. Extensiblity facilitates standardisation - one dictionary can suit all (or most) purposes.

The new breed of fourth generation software can be interfaced to the dictionary such that we obtain the benefits of a centralized repository of data definitions, with few of the traditional problems. Prototyping is not stifled, the dictionary is accessed neither at run time nor at compile time, alleviating some system performance and reliability bottlenecks. The development software maintains the dictionary automatically, consistently, and accurately.

The data dictionary assumes an unobtrusive, yet immensely helpful role in application development.