A DATA DICTIONARY ARCHITECTURE FOR THE YEAR 2001

Leon Leong
Hewlett Packard
Information Networks Division
Cupertino, California USA

Summary

Data dictionaries will be a critical service in managing a company's information network. Data dictionaries are being applied in many areas: fourth generation languages, report writers, query tools, network directories and transparent network access products. Thus choosing a data dictionary with an appropiate feature set can be the key to a well managed information network. This paper gives an overview of several features that have been included in Hewlett Packard's new dictionary product, System Dictionary, how they function, and what the features can be used for.

Introduction

A data dictionary is a central repository of information about the data and resources that a company has on its computer systems and network. Dictionaries are used to store, manage and organize this information. The information about the data and resources that a company creates and uses is called "metadata"; metadata means data about data. Metadata can include information about the name, size, structure, use, ownership, location and other attributes of a company's data and resources.

Typical objects that are documented in a company's information environment include:

- data base schemas,
- file and record layouts,
- data element descriptions and their use,
- computer nodes and device configurations,
- network topology,
- users an their security information.

A dictionary does not contain the actual object; just a description of the object. For example, a company may have a field in a file called "PART-NUMBER". A data dictionary would contain information such as,

- size (10 bytes),
- data type (ascii characters),
- display length (10 characters),
- owner (manufacturing),

whereas the file would contain the actual part numbers, for example 3225490001.

Subsystems which have objects being described in a data dictionary may have their own method for describing its structure and usage. For example, a data base has a schema, a network has a directory. These other methods, however, are very specialized for the particular subsystem; for example, it is not possible to describe the network topology in a data base schema, or vice versa. A data dictionary provides users a common format for describing metadata, such that it can be shared, and not redefined for every subsystem.

At first impression, one would guess that the users of a data dictionary would be individuals such as data base administrators, system analysts and programmers. However, there are many different types of products which can also be called "users" of a data dictionary:

- Fourth Generation Languages

  A fourth generation language (4GL) can use a data dictionary to resolve data definitions at compile/load time. A 4GL programmer only needs to know the name of the files, data bases or data elements that they want to use; the 4GL then retrieves from the dictionary the data type and structure information when the program is compiled. Inconsistencies between programs are reduced because there is a single source of definitions - the data dictionary.

- Ad Hoc Query Tools and Report Writers

  An ad hoc query tool/report writer will use a data dictionary to display to a user possible data to report on. For example, the query tool will, based on a user id, determine from the data dictionary what that user is allowed to access. The query tool will then retrieve from the dictionary the files, data bases and data elements and display them to the user. The user will then choose what to report on. The query tool then uses the metadata in the dictionary to determine the optimal access path.

- Application Customization

  Many applications tend to be very specific to an organization's operation. However, if an application were to utilize the metadata in the dictionary, then the dictionary can become a customization tool for the application. For example, if all headers and screen field names were stored in a data dictionary then a subsidary of a company in a different country could change the application to their native language, just by changing the metadata in the dictionary.

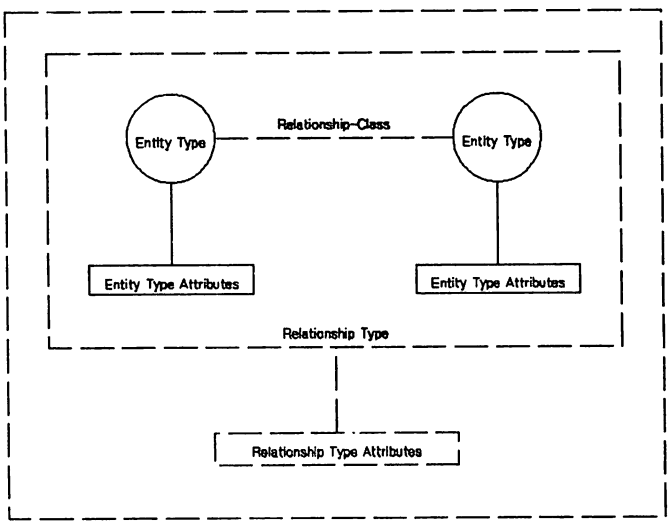
- Networks and Transparent Access

  A dictionary can be used to describe the location of nodes, and the topology of a company's computer network. In addition, a node's device configuration could also be included. This has potential uses for network configuration and management. By combining the network information with data structure information, applications can implement transparent network access for their end users.

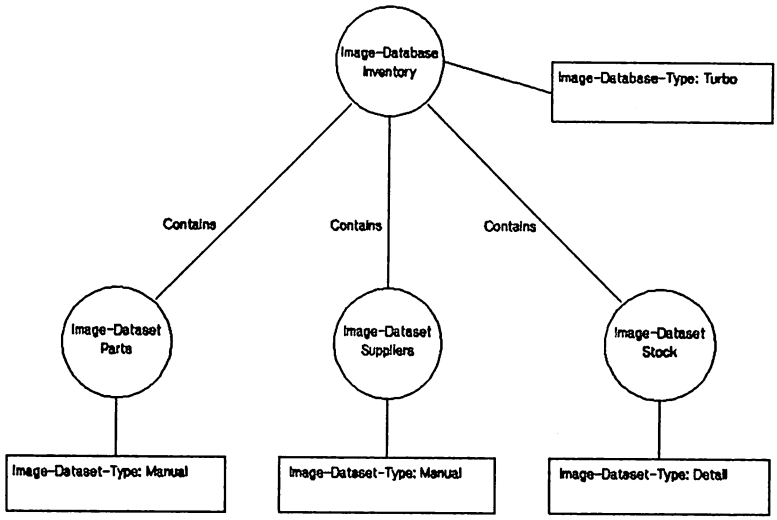
The uses of a data dictionary are many, and is only limited by the imagination of the people using it. But what of the features needed of a data dictionary, to make it usable for all these applications? Loading a dictionary can be an expensive operation; a dictionary should be evaluated for its feature set before being used as a central part of an information network. This paper describes some specific dictionary features which have been put into Hewlett Packard's new data dictionary product, System Dictionary and some the uses and benefits of those features.

Entity Relationship Model

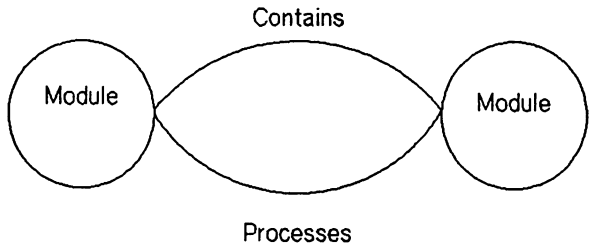
System Dictionary is based on the Entity-Relationship model. The Entity-Relationship model is composed of entities, relationships and attributes. Entities are objects in a company's information network. Every entity will belong to an entity type. An entity type is a template which describes the attributes associated with an entity. Relationships describe logical associations between entites. Attributes are pieces of information about either entities or relationships. Similarly, every relationship belongs to a relationship type; a relationship type is a template which describes the attributes associated with a relationship.



As an example, suppose a company has an Inventory Image database. The database is composed of three datasets: manual masters Parts and Suppliers, and a detail Stock. In the Entity-Relationship model, Image-Database and Image-Dataset will be entity types. Inventory is an entity of the entity type Image-Database. Parts, Suppliers and Stock are entites of the entity type Image-Dataset. The relationship type Image-Database Contains Image-Dataset, has three relationships in this example: Inventory contains Parts, Inventory contains Suppliers, and Inventory contains Stock. An attribute for the entity type Image-Dataset would be Image-Dataset-Type; for the entities Parts and Suppliers, the attribute would have a value of "manual", and for the entity Stock, the attribute value would be "detail".

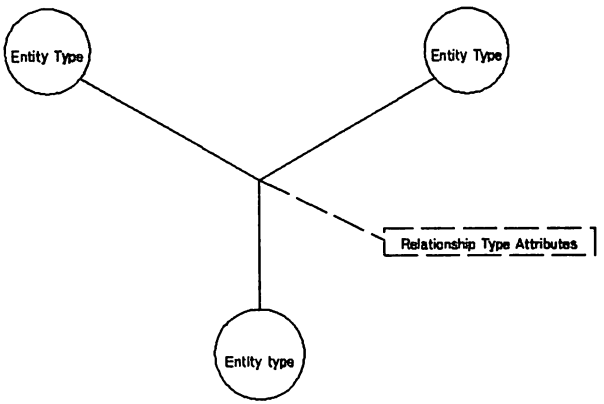


System Dictionary allows more than one relationship type to be established between entity types. For example, a code module may process another code module, or a code module may contain another code module. In order to allow for more than one relationship between entity types, System Dictionary includes, as part of the definition of a relationship type, an additional descriptor known as a relationship class. In the Image example, only one relationship type exists between the entity types Image-Database and Image-Dataset: Image-Database Contains Image-Dataset. Two relationship types exist between Modules: Module COntains Module, and Module Processes Module. The relationship classes are: Contains, and Processes.



Example of multiple relationship types
between entity types

Most relationship types will involve only two entity types. However there are cases in data modeling where, for integrity and consistency purposes, three or more entity types are needed in a single relationship type. System Dictionary allows a user to have up to six entity types in a relationship type.



A Three-Way Relationship Type

Extensibility

System Dictionary comes with a predefined set of entity types, relationship types and attributes; this predefined set is also known as the "core set". Thre core set of System Dictionary is extensive enough to cover the basic subsystems on the HP 3000. The core set includes structures which cover:

- MPE file system,
- KSAM files,
- Image/3000 data base management system,
- Network node locations,
- Code modules,
- MPE accounting structure,
- Cobol data structures.

The core set provides a stadard structure for most customers to work within. However, every customer's information model will be a superset of the core set; in addition, probably no two customer's information model will be exactly the same. Consequently, as customers adapt System Dictionary for greater use in their information network, they will want to extend the dictionary with their own entity types, relationship types and attributes to match their data model. In order to support this, System Dictionary allows a customer to create new entity types, relationship types and attributes. In addition, System Dictionary will allow relationship types to be established between existing core set entity types and customer entity types, and for customer defined attributes to be added to core set entity types and relationship types.

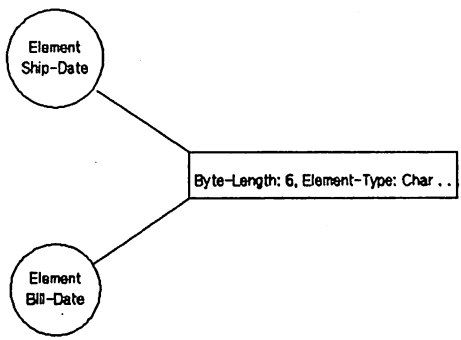
As another part of the extensibility feature, System Dictionary allows customers to specify default values for attributes, and also allows the specification of edit values for attributes. This capability provieds a form of integrity checking on attribute values entered. For example, the core set attribute Image-Dataset-Type will have the edit values "Manual", "Automatic" and "Detail" as its edit values. If any other value is entered for this attribute, an error will be returned. Customers can modify the edit values on core set attributes if they desire.

System Dictionary cannot anticipate every future use of the product today, let alone the uses by the year 2001. Extensibility allows the customer to grow their dictionary to meet their needs today and in the future, as they make more and more of their applications dictionary based.

Naming

Naming is an important part of the System Dictionary architecture. The most basic function of System Dictionary is as a name server. Users supply a name to the data dictionary system, and the dictionary system returns a set of attribute values for that name. Every entity in System Dictionary has a primary name; that is the name that it was created with.

System Dictionary supports "synonym" names for entities. Customers will often have situations where they have multiple names for an entity, and they would like to search the dictionary based on any one of the names. When a user gives a synonym name for an entity, the attribute values returned are the same as if the entity's primary name were used. For example, a customer may have a data element named "Ship-Date" in one application; in another application, the data element may be known as "Bill-Date". Since the name Bill-Date refers to the same data element, it should be created as a synonym for the data element Ship-Date. In System Dictionary, a user can create synonyms for any entity that they own; when the original entity is deleted, all the synonyms are also deleted.



System Dictionary also provides for "alias" names for entities. Alias names are different than synonyms, in that aliases are not used as keys into the dictionary. Aliases are attributes of an entity. Aliases are most useful when an entity's primary name is changed, due to naming conflicts. In the case of a naming conflict, an entity may be renamed, with the original name being assigned as an alias to the entity.

System Dictionary provides another naming feature: internal and external names. Customers will often have operations which span many countries. Each country will have its own language and terms that they use. A local data administrator may customize an application by modifying the names in the dictionary. However this can lead to problems when the originator of the application wants to make some modifications or develop utility programs which operate from the dictionary. System Dictionary provides two open modes to solve this problem: internal and external name mode.

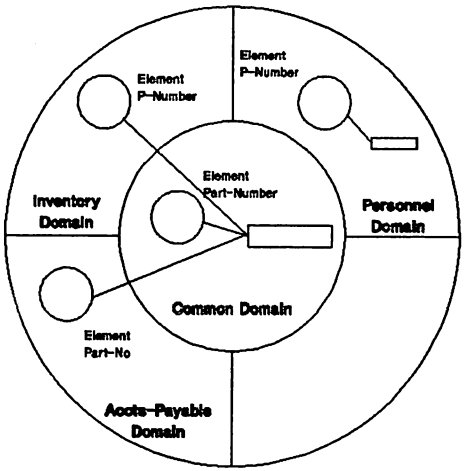
When a user creates a dictionary object, such as an entity, both an external and internal name can be supplied. If no internal name is supplied, then it defaults to the external name. External names can be modified to conform to the native language of the user; internal names can never be modified. Retrieval from the dictionary will be via the name open mode. Thus application developers which develop softwarebased on names in the dictionary can always use internal names, while the end users can use the external names.

Domains

Domains are separate name spaces in the data dictionary. Domains are useful for partitioning the dictionary by application system. Customers will often have many different application systems already in existence before purchasing a dictionary, or will have several application systems in development concurrently. Domains aid in migrating multiple applications to System Dictionary. These application systems will usually have conflicting names and definitions For example, an inventory application may have a data element "P-NUMBER", which is used to represent the object Part-Number. A human resources application may also have a data element "P-NUMBER", which in its context is used to represent the object "Personnel-Number". A third application, such as an Accounts Payable system may have a data element "Part-No"; Part-No refers to the same data element in the Inventory application known as "P-NUMBER".

These three application systems may already coexist when a dictionary is purchased to manage the overall information environment. Attempts to load the data definitions for the three application systems in the same domain lead to a variety of name conflicts. This would force the resolution of the name conflicts before the dictionary could be utilized, and it would also require changes on two the application systems. Domains are a useful feature to solve this problem. An INVENTORY domain could be created for the Inventory application, a HUMAN-RESOURCES domain for the human resources application, and an ACCOUNTS-PAYABLE domain for the accounting application. Since each domain is a separate name space, no conflicts would occur.

In order to promote the sharing of data definitions, however, a "common" domain needs to exist in the data dictionary. Users are allowed to link entities and relationships in the common domain. This linking allows the attribute definitions to be shared, yet it also allows each application system to keep its own name for entities. Local and common domain entities and relationships can be linked over time, and application system names adjusted over time as the application systems evolve. When an application system is moved to another node in the network, its definitions are separated out nicely by domain.

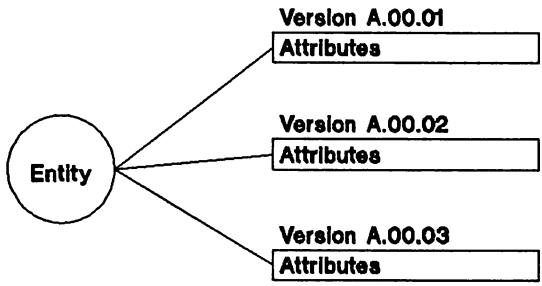


Linking definitions between domains

Version Control

Application systems are "living" systems. Application systems change, as a normal part of user feedback, growth, changing business environments and perhaps governmental regulation. A data dictionary should support the application system life cycle by having a version control feature. Version control as the name implies, allows entities and relationships to have multiple versions of values for attributes. For example, in version A, the element "Zip-Code" may have a length of five digits, while in version B of an application, the element may have a length of nine digits.

System Dictionary supports the concepts of version control. Every domain may have more than one version of entity and relationship definitions. Each version in a domain has a status: test, production, or archival. Within a domain, there may be as many test or archival versions as a user may want, but only one version may have its status set to production. Definitions in test versions may be modified, while a version which is set to a production or archival status becomes "read-only".



When a user wishes to access definitions in System Dictionary, the user must first open System Dictionary. The System Dictionary open operation requires the specification of a version in a domain,

to retrieve the defintions from. The version can be specified by one of two methods: one method would be to specify the name of the version; the other method is to specify a version status. System Dictionary will look for the version which was last set to the specified status to retrieve definitions. A dictionary without version control requires all applications to upgrade to a given version at the same time. System Dictionary's version control feature allows each application to upgrade to the version at a time which best suits the user.

Dictionary Security

System Dictionary's security system is based on the ownership and capability. A user of the System Dictionary is known as a "scope". Every entity and relationship decides what level of access, also known as sensitivity, that other scopes may have. There are three sensitivity levels: public read, which allows any other scope read access; public modify, which allows any other scope read and modify access; and private, which does not allow any other scope access. In the case of private sensitivity, System Dictionary allows the owner of an entity or relationship to explicitly assign on a scope by scope basis, read or modify access.

Domains have two sensitivity levels: public modify and private. In the case of public modify, any scope has access to the domain; in the case of private sensitivity, only the owner scope may access the contents of the domain.

Every scope is assigned a set of capabilities. These include:

- Customization, which allows a scope the ability to create, modify and delete entity types, relationship types, attributes and relationship classes,

- Domain, which allows the creation and deletion of domains,

- Version Control, which allows the creation and deletion of versions,

- Security, which allows the ability to create and delete scopes,

- Create, which allows the ability to create, modify and delete relationships and entities,

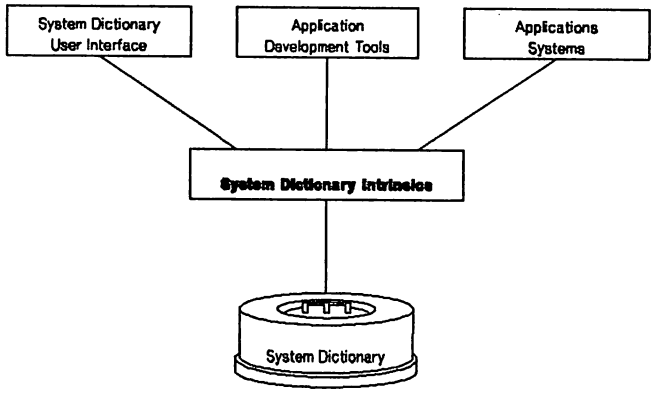
- Read, which allows the ability to read entity and relationships.

System Dictionary has one "superscope", known as the dictionary administrator (DA) scope. The dictionary administrator scope has full control over System Dictionary; this scope can perform any type of access on any domain, version, entity type, relationship type, attribute, relationship class, entity or relationship regardless of who the owner is.

Programmatic Access

An important feature of System Dictionary is that the product provides a standard, stable, and supported set of library routines, known as System Dictionary intrinsics. System Dictionary intrinsics provide independent software vendors and programmers the capability

of accessing the dictionary contents without having to know how System Dictionary's internal structures are organized. By using the intrinsics, ISV's are protected from changes in the internal structures which may occur as data management technology improves, and System Dictionary takes advantage of those improvements.



Intrinsics provide a method for integrating applications with the dictionary and thus provide a more active environment, whereby changes in the dictionary can be reflected immediately in the applications. Active dictionary environments will be commonplace as customers evolve their information networks to the year 2001; intrinsics are a method for implementing it.

Summary

This paper has touched upon several of the features incorporated in Hewlett Packard's new dictionary product, System Dictionary, and what benefits the customer derives from the features: flexibility, migration, and growth to an active dictionary environment.

Biography

Leon Leong has been with Hewlett Packard for past seven years. He is the R & D project manager for the System Dictionary project.