Information transferral within a distributed data base via a generalised mapping language*

R. H. Bonczek, C. W. Holsapple and A. B. Whinston

Krannert Graduate School of Industrial Administration, Purdue University, West Lafayette, Indiana, 47907, USA

This paper investigates some of the problems encountered in the management of a data base that is logically distributed. The term logical distribution is used to connote a situation wherein users in various locales are responsible for creation and maintenance of their own portions of a data base, but a user in one locale can access data maintained by a user in another locale. An example of this situation is drawn from the realm of water quality management. A generalised mapping language is proposed as a mechanism for information transferral within a distributed data base, and a general data structure for supporting the mapping function is illustrated. The presented method accommodates a variety of user views, is independent of whether the data base is geographically distributed or centralised, furnishes a straightforward security mechanism and provides a basis for treating the contingency of uninformed or non-programming users.

(Received November 1976)

In view of the rapidly expanding data base field and the frequently decentralised organisational environment within which a data base management system must function, the issue of distributed data bases is of topical concern. This paper presents a method for addressing the problem of distributed data bases. A data base is considered to be distributed if users in various localities are responsible for creating and maintaining their own portions of the data base and a user in one locality can access data maintained by a user in another locality. Such a situation can arise on a variety of machine configurations (e.g. Asenhurst and Vonderoke, 1975; Canady *et al.*, 1974; Farber, 1975). Although we deal specifically with the problem of co-ordinating area wide management of water resources, it should be noted that the method presented is generally applicable to settings of physical or managerial decentralisation in both the private and public sectors. The perspective taken here is logical in nature; the question of resolving physical or hardware incompatibilities (Schneider, 1975; Anderson *et al.*, 1971) is not within the present scope. Resolution of incompatible data structures to allow transferral of data within a distributed data base will serve as our focal point. Briefly, the methodology being proposed involves the utilisation of a non-procedural mapping language (Bonczek and Whinston, 1977) to define a mapping of data values out of one network data base into another network data base without requiring user specification of any intermediate data structures. The user has merely to supply the map; neither preprocessing nor postprocessing relating to normal forms is required. The mapping is able to handle special cases of the above situation such as linear list to tree, linear list to network, tree to network, network to tree, etc. Prior to detailing the specifics of the mapping approach, we describe a water resource management context in which it is useful.

Area-wide water quality planning

Section 208 of the Federal Water Pollution Control Act Amendments of 1972 calls for area-wide implementation of technical and management planning, with the objectives of satisfying 1985 water quality goals and establishing a plan for municipal and industrial facilities construction over a twenty year horizon. An emphasis is placed on locally controlled (area-wide) planning. Our concern here is confined to those aspects of technical planning which can be enhanced by utilisation of a data base management system.

A river basin may be conceived as consisting of a number of areas. To be designated under Section 208, an area must be in need of a complex control program and must exhibit either impairment (i.e. water quality within the river segment is substandard) or a need for preclusion of desired uses. An area-wide plan is constrained in that it must conform to the basin-wide plan, must account for existing treatment facility plans and must be implementable from a managerial standpoint. The objectives of technical planning within a designated area are to identify the priorities of water quality problems in the area; recognise constraints in methods for dealing with these problems; formulate alternatives for the satisfaction of stated water quality goals; develop cost data for each alternative; select the least-cost feasible alternative, given existing regulatory authority and qualitative restrictions; and update the plan as needed.

To satisfy these objectives, the planner for a designated area must be able to store, manipulate, retrieve and analyse large volumes of data. It is imperative that the data be stored in a manner that takes their intricate interrelationships into consideration. The planner must be able to utilise effectively a collection of germane application programs which provide selective retrieval of a multitude of data configurations; which generate plots, statistical analyses and projections; and which perform large scale simulations and optimisations. Each of these programs requires a particular configuration of data for input. In relatively unstructured decision activities, the types of analyses and reports which the planner needs are subject to frequent modification. In traditional file-oriented systems this requires that a new report generator be written every time a new type of report is needed. Typically, a local planner is not a computer programmer; presumably the planner's time is too valuable to be preoccupied with writing report generators, writing programs to maintain data files, and interfacing large scale application programs with data files. A system which obviates these crude necessities has been described in Holsapple and Whinston (1976). The planner's interface with the system occurs through a non-procedural English-like query language (Bonczek, Haseman and Whinston, 1976a; 1976b).

Each designated area within a basin is responsible for administering its own data base. This data base contains information that is applicable to the area, such as the land use descriptions and plans, the river's state characteristics within the area, the area water quality characteristics and goals, and the various area treatment plans. But to simulate an area's water quality for a given treatment plan, data such as treatment

*Research supported in part by: Office of Water Research and Technology Grant 143460-76 and National Science Foundation Grant MCS-67-24675

plans and water quality characteristics of other designated areas within the basin are required. The precise method for accomplishing this depends upon the nature of the distribution. In the ensuing section we present a taxonomy for identifying the basic types of distribution that can occur.

Varieties of data base distribution

The taxonomy is developed within the context of congruent data base implementations among local data bases. An example of a local data base is the data base for a particular designated area of water quality management. An example of a global data base is the set of local data bases for all designated areas within a basin. The local data bases may or may not be situated on the same hardware; as previously noted we are not here concerned with word size or character representation code conversions. If on the implementation level, all local data bases within the global scope utilise the same basic Data Description Language (DDL) and the same basic Data Manipulation Language (DML) commands, then we say that they are congruent. Where there are hardware variations within the global context, utilisation of a data management system that is largely machine independent is required (e.g. Haseman and Whinston, 1977) there may well be variations from one locality to another in certain non-essential DDL features and in high level or extended DML commands, but at some implementational resolution level the DDL and DML are identical. Another type of interlocality disparity may occur in user views of the data management system. Even though one local data base system utilises implementation features identical to those of another, the users of the first data base system may have a very different conceptual view and may therefore utilise different commands than the user of the second system (Bonczek, Holsapple and Whinston, 1976).

The global data base is characterised according to two dimensions. The first represents the relative degree of volatility in underlying logical structures of the local data bases. The second is indicative of the degree of uniformity of underlying logical structure among local data bases. The degree of volatility ranges from static, wherein logical structures of local data bases are not subject to change over some time horizon, to dynamic, wherein there are frequent alterations of logical structures over the time horizon. Uniformity refers to situations where identical logical structures are defined for all local data bases; or at least an appreciable subset of the logical structure of each local data base is identical to a subset of the others. This framework yields the following four extreme cases:

1. Static structures and uniformity of structures.
2. Static structures and non-uniformity of structures.
3. Dynamic structures plus a desire to maintain uniformity across all local structures.
4. Dynamic structures with no uniformity.

Prior to describing how these cases can be handled by the mapping language and processor previously mentioned we provide a cursory illustration of the mapping procedure (for greater detail see Bonczek and Whinston (1977)).

Generalised mapping language

The generalised mapping language has the following distinctive characteristics:

1. A linguistic formulation conducive to structural analysis and facile implementation of language constructs;
2. A high level of non-procedurality that renders the language convenient to non-programming users;
3. Generality in the sense of capacity to directly map information from one network data structure into another network data structure, thereby obviating the crude necessity of constructing intermediate linear data structures;
4. Flexibility in terms of the ability to handle traditional functions of retrieval (typically, network data structure to linear data structure) and loading (typically, linear to network data structure); these are actually special cases of the third characteristic.

The mapping language has a context-sensitive grammar. Using Chomsky's concept of transformational grammars, the mapping processor applies inverse transformations to statements in the mapping language in order to arrive at corresponding expressions in a language derived from a context free grammar. An expression in this context free language is compiled using well known methods of syntax directed analysis (Aho and Ullman, 1972). The specific implementation used, including the precedence tables, is described in Bonczek, Haseman and Whinston (1976a), and Haseman and Whinston (1977). Information from the compiled expression serves as input to network traversal routines (Bonczek, Haseman and Whinston, 1976b) which make the requested extractions from one data base and insertions into another (Bonczek and Whinston, 1977). This is subject to any conditions specified in the original mapping statement and conditions imposed by the security system (Cash, Haseman and Whinston, 1976). If a mapping statement is ambiguous (i.e. does not identify a unique path in the source data base and a unique path in the target data base) the mapping processor prompts the user for clarification. All of these processes (ambiguous statement resolution, security, etc.) can be characterised as inverse transformations on the original mapping command; thus the language processing techniques are sufficient for the processing of the map.

The mapping processor utilises a Data Manipulation Language for the purpose of interfacing with a data base. The DML which has been implemented (Haseman and Whinston, 1977) is in FORTRAN and so it is largely machine independent. FORTRAN is also used for implementation of the mapping processor proper. Since FORTRAN can serve as a host language for the DML, the mapping processor can treat any DML command as a FORTRAN subroutine. Thus the mapping processor is able to perform any needed manipulation of a data base by using appropriate subroutine calls.

The mapping processor is maintained as a collection of overlays as indicated in **Fig. 1.** The CONTROL overlay is responsible for initiating input, buffering internal data and calling the other overlays of the system. The PARSER is composed of several parts. The PREPROCESSOR overlay performs dictionary lookups for synonyms and noise words; as such it performs inverse transformations. The IT overlay performs the bulk of the inverse transformations (e.g. restructuring the mapping statement and identifying levels of retrieval when multivariate functions appear in the mapping statement). The SECURITY overlay also consists of inverse transformations based upon the security status of the user issuing the mapping statement. The COMPILER overlay performs the precedence parsing of the expression (context free) that results from the inverse transformations, thereby producing an internal object code for evaluation in the execution routines.

The components of the PATH FINDER overlay find a path(s) in a network data base that corresponds to the map command. This path finding function can be considered to be an inverse transformation, which suggests that the path finding phase should precede execution of the COMPILER overlay. This is precisely what occurs and the compiler makes use of information from the path finder to order conditional clauses of the mapping statement in a heuristically optimal fashion. The INTERACTIVE module of the path finder is also responsible for resolution of ambiguities as they arise.

The EXECUTION routines consist primarily of two parallel

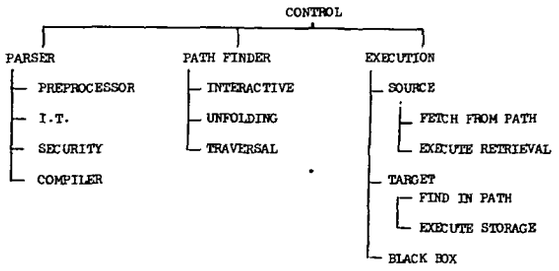

Fig. 1 Structure of the mapping processor

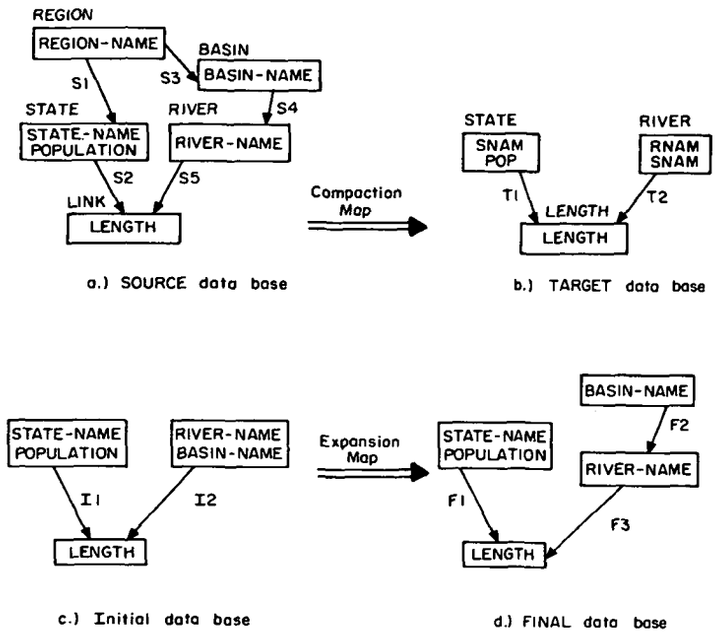

Fig. 2 Data base compaction and expansion

sets of programs: one for retrieval of data from a source data base(s) and the other for storing information into the target(s). As data are fetched, they are tested for satisfaction of all conditions in the compiled map. Data which meet all conditions are stored into the appropriate path of the target(s). The other portion of EXECUTION is labelled BLACK BOX. This is not currently implemented; its purpose is to perform necessary encoding and decoding of data that is to be transported to a data base with a physical structure that differs from that of the source data base. Issues such as word length and character codes must be considered in this type of operation.

Mapping problems may be partitioned into two categories. These are data base compaction and data base expansion. **Fig. 2** presents simple examples of these two cases. Each rectangular box, called a record type, represents a group of data item types. For instance the record type STATE is composed of data item types STATE-NAME and POPULATION. Associated with each record type are a number of record occurrences which contain data values. The record type STATE forms a template which describes the composition of each record occurrence associated with it. A sample occurrence of STATE is composed of the data item values INDIANA and 5,000,000. Similarly an example of an occurrence of the record type RIVER is the data value WABASH. Each arrow indicates a 'set' which specifies a relationship between occurrences of two record types. This is a one-to-many relationship between occurrences of the 'owner' record type and the 'member' record type. The arrow points from the owner of the set to the member. In Fig. 2(*a*) for example set S4 associates many occurrences of RIVER with each occurrence of BASIN (i.e. each basin consists of several rivers). Note that there is a many-to-many relationship between occurrences of STATE and RIVER via occurrences of LINK. Each occurrence of LINK is composed of a data item value that denotes the length of that portion of a river which is in a state; the particular state and river associated with a given length (i.e. LINK occurrence) are indicated by the data values of the occurrence of STATE and the occurrence of RIVER which respectively own S2 and S5, each having the LINK occurrence as its member.

The mapping problem is that of specifying the conditions under which data values organised according to one logical structure (source data base) are to be inserted into the form of another logical structure (target data base). Since defining the logical structures is not strictly a part of the mapping process, throughout the ensuing discussion all logical structures are assumed to be previously defined. As can be observed in Fig. 2 the procedures of data base compaction and data base expansion are of a complementary nature. The most familiar variety of compaction is data retrieval which extracts data values from a source data base into a linear target which is used for display or as input to application routines. The most familiar variety of expansion is simple data base loading which 'explodes' data values from a linear source into a network target.

The compaction map of Fig. 2 can be specified as follows:

> Taking SOURCE to TARGET, map STATE-NAME to SNAM, POPULATION to POP, LENGTH to LENGTH, RIVER-NAME to RNAM, and BASIN-NAME to BNAM relating S1 and S2 with T1, and S3, S4 and S5 with T2.

The expansion map is:

> Taking INITIAL to FINAL, map STATE-NAME to STATE-NAME, POPULATION to POPULATION, RIVER-NAME to RIVER-NAME, LENGTH to LENGTH, BASIN-NAME to BASIN-NAME, relating I1 to F1, I2 to F2 and F3.

An important special case of network structure is a linear or strictly tree-like data structure. In this case, there is only one possible relationship between two record types; therefore no relationships need to be specified to execute a map. In a full network a multitude of relationships may exist between record types; inclusion of relationship correspondences in the map command prevents ambiguities that arise in their absence. An alternative to user specification of relationship correspondences is to allow the mapping processor to prompt the user in order to elicit this information (Bonczek and Whinston, 1977). Finally, we note that conditional restrictions may be included in a map command to limit the data values which are subject to the map; this is useful for security purposes as well as checking for errors in the data. For example the compaction map for Fig. 2 could be restricted by adding the clause

$$\ldots \text{for } 1{,}000{,}000 < \text{POPULATION} < 18{,}000{,}000 \text{ and } 0 < \text{LENGTH} < 375.$$

Visualisation

We first of all examine how the generalised mapping procedure is used to handle case I of static and uniform structures within a distributed data base. Suppose the user of data base A requires information from data base B. Since there is uniformity of structure user A needs only to match like names in the mapping statement, specifying B as the source and A′ as the target. A′ is probably a scratch data base for temporary use, though the data values could be mapped directly into A. Having the data in this form at A, it is immediately useable by applications set up to handle analyses of data base A. For cosmetic purposes it may be desirable to define another command which does not require redundancy; this is a special case of the MAP command. Note that MAP is also quite capable of handling different names for the same attribute

within the context of uniform structures; if the query system already permits definition of synonyms, accomplishing the map in this manner adds unnecessary complexity.

In case II, user A is allowed to view data base B as being uniform with data base A. Since the data bases A and B are static the data base administrator can devise and store a map of B into the terms and structure used by A, as a one shot operation. If this is done then the user can proceed as in case I. An alternative method, which is more cumbersome for the user, is the performance of a single direct map; this requires that the user be knowledgeable in the particulars of other data bases than his own.

Suppose that in case III changes are made to local data bases on a weekly basis. Since the changes are to be made uniformly across all data bases the data base administrator needs to prepare only a single mapping command which for any locality will map data base X_t into X_{t+1}. This command is used simultaneously at all localities and thereby preserves global uniformity. Data transferral among localities can then proceed as in case I.

For case IV we store a number of maps in the data base as depicted in **Fig. 3.** We first of all note that Map 1, which takes data base B into a form that is uniform with data base A, also contains all information necessary to accomplish the reverse procedure. Also observe that maps are stored only if one area is permitted to access another (e.g. there is no direct access between area A and area D). This case is handled just like case II except there must be a continuing update of stored maps. Notice that in any instance where one locality makes a change in its own data structure, no reprogramming is required in the applications at other localities. At most, all that is needed is a modification in the appropriate stored map. The instigator of the change is responsible for modification of the pertinent stored maps; such modifications may be effected automatically, since all information needed for the modifications is present in the locality's specification of the change to its own data structure.

Additional considerations

Maps stored for the purpose of providing a user with a uniform view of all data bases, can also be used as security mechanisms. Data items or particular paths of a given local data base that are to be protected from outside scrutiny may be precluded from all (or some) stored maps that are available to users in other localities. Since maps may contain conditional clauses, conditional security (Cash, Haseman and Whinston, 1976) may also be accommodated. Not only are certain large portions of the data base protected, but on a more detailed level, access to a range of data values for a particular data item type may be effectively restricted.

Thus far we have considered the specification of a map that takes data from a single source into a target data base. At first glance, the situation wherein data is to be gathered from multiple sources into a target may seem to be quite complex. However, by virtue of the utilisation of stored maps which effectively provide uniformity of data structures, the data transferral is fairly straightforward. This is particularly true in the instance where the values of the same data item types are drawn from all of the sources. For example, a target data base may be constructed from all (or some, if the map is conditional) values of the data item type POLLUTER-NAME that exist in some specified source data bases:

taking AREA-1, AREA-2, AREA-3 to AREA-4 map
POLLUTER-NAME to POLLUTER-NAME . . .

To obtain a target where different data item types are drawn from various sources a series of maps of this variety may be needed.

When the types of data stored in data base A (e.g. water quality data) are incommensurable with those stored in data base B (e.g. land use data) then the user (or DBA) must define the target data base. This is not required (though it is allowed) for the previously examined cases, where the existing local logical structure furnishes an implicit target that is usable by existing application routines. When data bases are incommensurable then the source structure (or a view of it according to an existing map) must be known, the target must be defined and a map is written accordingly.

The preceding discussions presuppose that the user knows the location of data to be retrieved. However, the uninformed or non-technical user may be unaware of the data base(s) which contain the information required; it could even occur that such a user is unaware of the logical distribution of the global data base. Such a state of affairs is not uncommon in situations where users are planners and managers who typically are not expected to be experts in data base technicalities.

The solution depends upon the volatility of the types of information the user needs from other data bases. If the same type of data is repeatedly required from the same sources then the appropriate map may be stored and executed in response to the pertinent user query. For instance to simulate water quality, the simulation package requires certain types of data from other locales. The types of data required are invariant, though the data values undergo frequent modification. A user query, requesting execution of this application, invokes an execution of the pertinent stored maps before performance of the desired simulation; and the entire mapping process is invisible to the user.

If the types of information requested by the user are subject to frequent change then the query system must be devised in such a way as to elicit the information needed to construct the required map. This involves interactively prompting the user.

A general data structure for mapping support

A crucial aspect of successful implementation is the existence of an organised, comprehensive and compact means for representing information about various user views and the maps required to support those views. We use the term deep structure to refer to the description of a locality's data structure according to which its data values are actually organised. Surface structure refers to a particular user view of a locality's data structure. There is one deep structure per locality. Recall that in the foregoing discussion, the mapping feature has been used in two ways, namely:

1. It serves as a means for specifying the sources, targets and conditions of data transferral. As such, it may be
   (*a*) defined by the user at the time of transferral, or
   (*b*) stored if it is subject to frequent use.
2. The mapping facility provides a method for supporting a

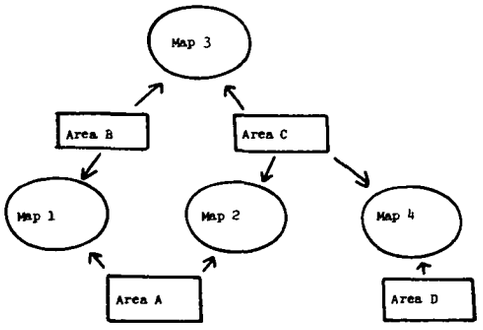

Fig. 3 Storage of maps

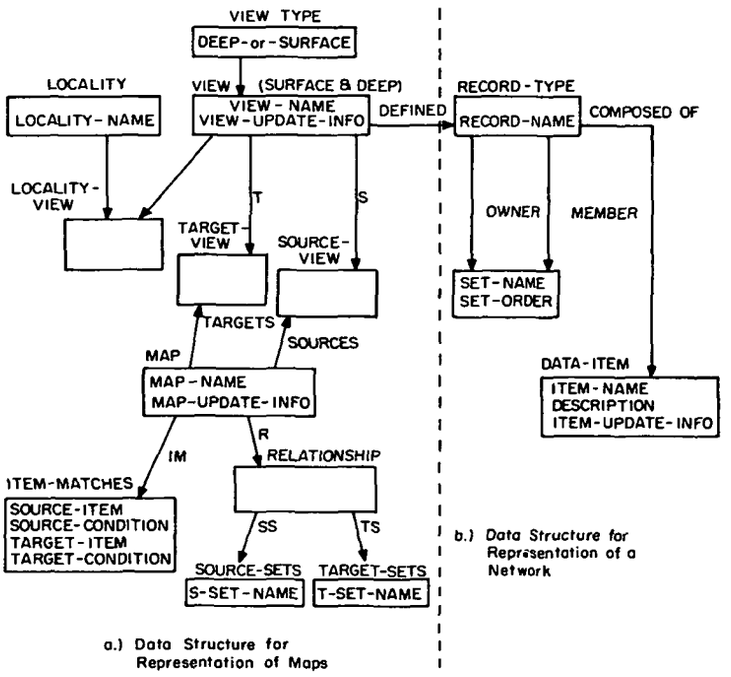

Fig. 4 General data structure for the support of data transferral within a logically distributed data base

variety of surface structures (each subject to its own security constraints) for a single locality's deep structure; i.e. a map may serve as a transformation that takes a deep structure into many surface views. A map is stored for each of the available surface structures based on a particular deep structure. There may be

(*a*) many surface structures that can be used at one locality with reference to its own data base

(*b*) many surface structures used at many localities with reference to a particular locality's data base, and

(*c*) a given surface structure may be shared among some localities.

Fig. 4 illustrates a data structure capable of supporting both of these methods of utilising the mapping feature; some version of this structure is incorporated into all local data bases. Furthermore, the depicted structure:

(*a*) allows the acquisition (subject to security constraints) of a full description of the logical structure of any deep or surface view

(*b*) provides a dictionary facility for detailed description of the meaning of each data item in each surface or deep structure

(*c*) provides a mechanism for specification of access privileges for all data item types (a particular locale's data retrieval ability is constrained by the views which are available to it)

(*d*) can accommodate information about the time of the most recent update of a particular map

(*e*) can serve as a basis for monitoring the most recent update to occurrences of a particular data item existing in a particular user view.

Within the data structure of Fig. 4(*b*), all deep and surface structures are represented as networks. Since the relational data base view has been shown to be equivalent to the network view (Bonczek, Haseman and Whinston, 1976a), an appropriate network surface structure is stored should a relational view be desired; translation of this structure to give the appearance of a relational data base is an exercise in cosmetics. From the figure, we see that each VIEW is DEFINED in terms of its RECORD-TYPEs. Each RECORD-TYPE is COMPOSED of zero, one or many DATA-ITEMs; a pair of RECORD-TYPEs may be associated by declaring one to be the MEMBER and the other to be the OWNER of a SET. This information is helpful when a user is devising a map from one view (or several views) into another, since it provides a complete description of the structure of each view as well as a dictionary facility for ascertaining the meaning of any data item.

The structure depicted in Fig. 3(*a*) is utilised to store all maps (see uses 1(*b*) and 2 outlined above). We first of all notice that a VIEW may be either deep or surface. Moreover, for a given LOCALITY, there may be many VIEWs and a particular VIEW may be used in many LOCALITIES; a LOCALITY is associated with a VIEW by means of the LOCALITY-VIEW record type. There may be many TARGET-VIEWS for each MAP, or there may be many SOURCE-VIEWs. Associated with each MAP are a series of ITEM-MATCHES and a series of RELATIONSHIPS, each of which relates a series of SOURCE-SETS with a series of TARGET-SETS. This general structure is capable of simplification for specialised purposes, as well as elaborations for further refinement.

A simple illustration of the schema of Fig. 4 (at the record occurrence level) may be provided by utilising the two data structures given in Fig. 2(*a*) and (*b*). Suppose that these are two views which we want to represent in a data base that is structured as shown in Fig. 4. In this example, we do not consider whether these are deep or surface views nor whether they are views of the same or different localities, since the occurrence level representations of such conditions are straightforward. Parenthetically we point out that there is nothing special about the names (TARGET and SOURCE) of the two views being considered.

Fig. 5 depicts the occurrence level representation for the two views which corresponds to the data structure of Figure 4(*b*). Each oval in Fig. 5 denotes a record occurrence of the type indicated in the right margin. In some occurrences, data values for all items are not shown. Arrows emanating from an occurrence point to occurrences of another record type which are owned by that occurrence via the set indicated in the right margin (on the same level with the arrows). For example, two occurrences of the record type VIEW are shown (SOURCE and TARGET). Recall from Fig. 2 that there are five record types in the SOURCE view and three in the TARGET view. These give rise to the eight occurrences of the record type named RECORD TYPE. Arrows associated with the set DEFINED indicate which of these eight occurrences belong to SOURCE and which belong to TARGET. Recall that on the occurrence level there must be a one-to-many relationship between an occurrence of an owner record type and occurrences of the member record type which are associated with it via a particular set. In graphical terms this means that no occurrence may be pointed to by more than one arrow of a given set type. Two arrows may point to an occurrence if each represents a distinct set. For instance, the occurrence 'T1' of the record type SET is pointed to by two arrows: one of the set type OWNER and one of the set type MEMBER. This shows that the set T1 (of Fig. 2(*b*)) has the record type STATE (of Fig. 2(*b*)) as its owner and the record type LENGTH (again see Fig. 2(*b*)) as its member. In Fig. 5 the row of occurrences of the record type called RECORD-TYPE appears twice. This is solely for the purpose of diagrammatic clarity. In the actual data base each occurrence in the RECORD-TYPE row appears only once. With further examination of Fig. 5 it can be verified that the two views of Fig. 2(*a*), (*b*) can be accommodated by the structure given in Fig. 4(*b*).

We now examine the manner in which maps may be stored in the structure of Fig. 4(*a*). **Fig. 6** continues with the above example, showing how the SOURCE and TARGET views

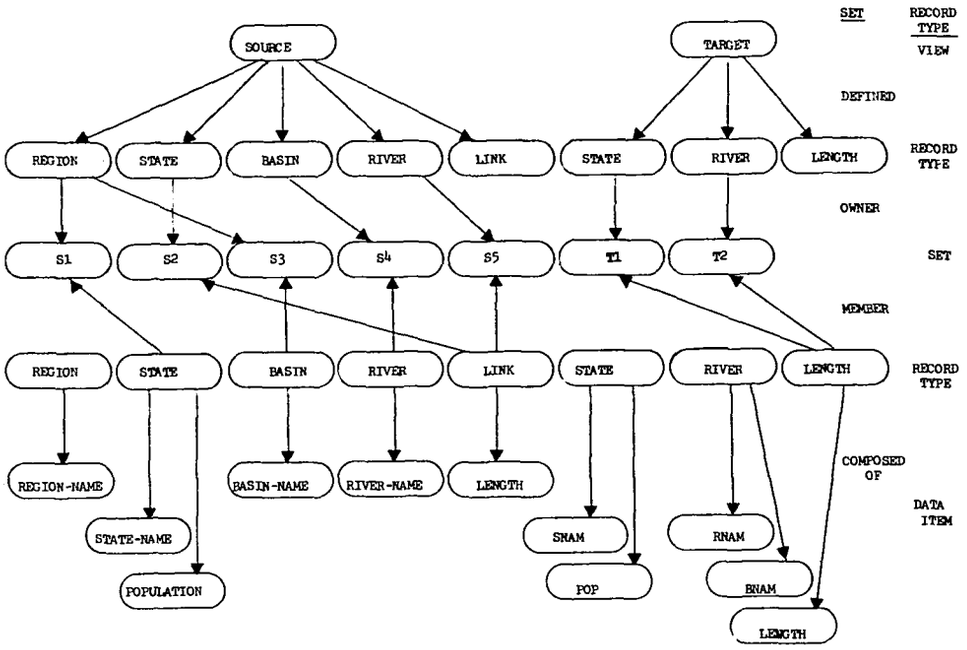

Fig. 5 Example of stored data structures at the occurrence level

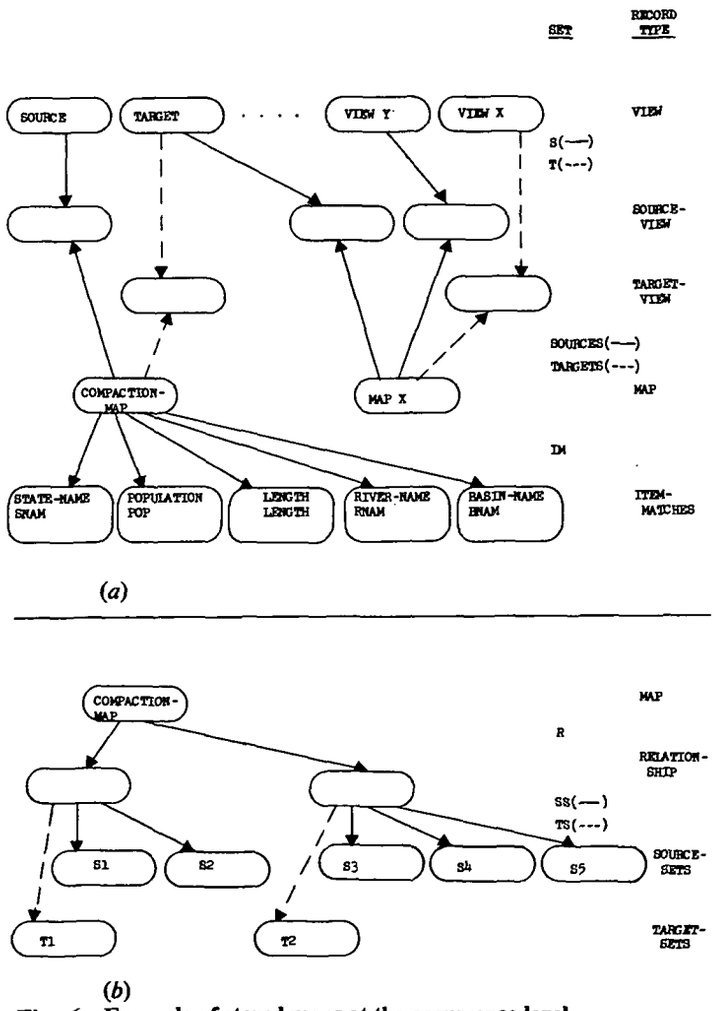

Fig. 6 Example of stored maps at the occurrence level

may be related to a particular map, called 'COMPACTION MAP'. Occurrences of the record type ITEM-MATCHES (e.g. 'STATE-NAME':'SNAM') correspond to the items matched in the mapping statement given in a previous section. Fig. 6(*b*) shows the corresponding matches of set relationships for that same mapping statement. Observe that Fig. 6(*a*) includes two additional occurrences of VIEW, namely 'VIEW X' and 'VIEW Y'. It also includes another occurrence of the record type MAP (i.e. 'MAP X'). This is used to demonstrate the occurrence level structure in the case where a map takes two sources into a target view. As shown in Fig. 6(*a*), the two sources of 'MAP X' are 'TARGET' and 'VIEW Y'; the target is 'VIEW X'. Similar occurrence level structures result for maps with multiple targets.

A scenario

Finally, we present a scenario of the procedures followed as a result of query submission. Each locality has its own copy of the query processor, which includes the mapping processor and is based on underlying DML (Bonczek and Whinston, 1977; Bonczek, Haseman and Whinston, 1976a). This supports all queries that exclusively access the local data base. When a request is made that requires a map (the map being either user supplied or drawn by the query processor from the local data base) that request is analysed by the local mapping processor and undergoes the following conceptual steps. For each requested source, the mapping processor generates a query in compiled form. This query is dispatched to the source, requesting extraction of data. Upon arrival at the source, the query is executed (subject to some priority scheme and concurrency policy) by the query processor of the source locality. The result is the generation of an extraction file which can be accessed by the query processor of the target for purposes of display, insertion into a target network (as specified in the map), or execution of a local application routine. Further research is required for a determination of both the machine configuration(s) most amenable to utilisation of the concepts outlined herein and the precise protocols to be observed in update of local data bases.

Conclusion

We have used the term distributed data base to connote a situation where control (i.e. creation and maintenance) of data values and data structure is distributed (without overlap) among users in various localities, but where access is general, subject to security constraints. The generalised mapping

language was briefly described and shown to be capable of handling data transferral in the four extreme distributed data base environments. A generalised data structure for supporting the mapping function is illustrated. The presented method accommodates a variety of user views of data base structure, is independent of whether the data base is geographically distributed or centralised, furnishes a straightforward security mechanism and provides a basis for coping with the contingency of uninformed users who may even be unaware of the logical distribution.

References

ASENHURST, R. L., and VONDEROKE, R. H. (1975). A Hierarchical Network, *Datamation*, February 1975.

CANADY, R. H., HARRISON, D., IVIE, E. L., RYDER, J. L., and WEHR, L. A. (1974). A Back-end Computer for Data Base Management, *CACM*, October 1974.

FARBER, D. J. (1975). A Ring Network, *Datamation*, February 1975.

SCHNEIDER, G. M. (1975). DSCL—A Data Specification and Conversion Language for Networks, *Proceedings of ACM SIGMOD Workshop*, San Jose, California, May 1975.

ANDERSON, R. D. *et al.* (1971). The Data Reconfiguration Service—An Experiment in Adaptable Process to Process Communication, *Proceedings of Symposium on Problems in the Optimization of Data Communication Systems*, Palo Alto, California, October 1971.

BONCZEK, R. H., and WHINSTON, A. B. (1977). A Generalized Mapping Language for Network Data Structures, *International Journal of Systems*.

HOLSAPPLE, C. W., and WHINSTON, A. B. (1976). A Decision Support System for Area-wide Water Quality Planning, *Socio-Economic Planning Sciences*.

BONCZEK, R. H., HASEMAN, W. D., and WHINSTON, A. B. (1976a). Structure of a Network Data Base Query Language, Krannert Technical Report, Krannert Graduate School of Industrial Administration, Purdue University, April 1976.

BONCZEK, R. H., HASEMAN, W. D., and WHINSTON, A. B. (1976b). Automatic Path Determination in a Network Data Base, Krannert Technical Report, Krannert Graduate School of Industrial Administration, Purdue University, April 1976.

HASEMAN, W. D., and WHINSTON, A. B. (1977). *An Introduction to Data Management*, Richard Irwin Co., Homewood, Illinois.

BONCZEK, R. H., HOLSAPPLE, C. W., and WHINSTON, A. B. (1976). Extensions and Corrections for the CODASYL Approach to Data Base Management, *International Journal of Information Systems*.

CASH, J. I., HASEMAN, W. D., and WHINSTON, A. B. (1976). Security for the GPLAN System, *International Journal of Information Systems*, August 1976.

AHO, A. V., and ULLMAN, J. D. (1972). *The Theory of Parsing, Translating and Compiling*, Vol. 1, Prentice Hall, Englewood Cliffs, NJ.

Book review

Advanced ANS COBOL with Structured Programming, by G. D. Brown, 1977; 497 pages. (*John Wiley*, £13·50)

High Level COBOL Programming, by G. M. Weinberg, S. E. Wright, R. Kauffman and M. A. Goetz, 1977; 252 pages. (*Prentice-Hall for Winthrop*, £13·55)

Both these books have the aim of allowing people to produce programs which are easily readable and maintainable and which are correctly structured. It is not therefore surprising to find chapters on structuring, style and testing in each. However their approach differs greatly.

High Level COBOL Programming takes a philosophical outlook to the problem. It considers the basic requirements of any program; the businesslike approach necessary to programming; and the need for good management in a programming environment. From this start it considers how programming should be controlled and how good quality programs can be written, pointing out that only too often standards are written purely as a result of a programming disaster of some type and not through any professional process. Much of the rest of the book is devoted to the ways in which various tools can aid the programmer. The authors argue that preprocessors allow programmers to use shorthand, while still producing fully readable outputs. By the use of macros, they can provide facilities to overcome some of the trickier COBOL areas, such as DO-WHILE constructs and the necessity for GO TO statements. The preprocessor described in the book is MetaCOBOL. This has an extension which can be used to aid the testing and control of the programming.

A further tool described is the source program management system LIBRARIAN which can be used to update libraries; maintain security, providing reliable back-up; and to provide a complete audit trail of changes made. Its ability to do simple SYNTAX checking and various levels of control listings is also pointed out.

In its arguments on methods of style to be used, techniques to be evolved and methods of testing a program, the use of these tools is presumed. The authors declare their interest in the software tools in that several of them are from the software house which develop them. It is a pity that names of similar products were not at least mentioned.

Advanced ANS COBOL with Structured Programming concentrates on the advantageous use of some of the more complex areas of COBOL. The ANS compiler used the latest IBM 370 version, and any differences from the '74 standard are noted. The book is well written, with excellent examples of many areas. The information on the SEARCH verb; the COBOL report writer; and the sort feature will be of particular assistance to readers and are considerably clearer than in any manufacturers' manual or similar reference book.

The book could be treated simply as a reference book, but any COBOL programmer would be likely to gain considerable knowledge from a thorough study of it. There are some criticisms of the book. Although structured programming is covered, it is not really dealt with in sufficient depth to justify the title. A chapter on the types of diagnostics which arise, even if only on IBM compilers, might have been of more general interest than one on IBM JCL and machine/language interface in general.

To summarise, two well written, easily readable books. Anyone possessing the software tools mentioned in *High Level COBOL Programming* would probably find it invaluable; anyone considering these or similar tools would find it interesting and others would probably enjoy reading it even if the cost might be hard to justify. The second book, *Advanced ANS COBOL*, wold not be out of place in anybody's computer library. Any IBM installation might seriously consider what part this book could play in a training programme.

J. A. EMERSON (Horsham)