DATABASE PROGRAM CONVERSION: A FRAMEWORK FOR RESEARCH

Database Program Conversion Task Group
of the
Codasyl Systems Committee

Robert W. Taylor, IBM Research, Chairman*
James P. Fry, University of Michigan
Ben Shneiderman, University of Maryland
Diane C. P. Smith, University of Utah
Stanley Y. W. Su, University of Florida

ABSTRACT

As requirements change, database administrators come under pressure to change the schema which is a description of the database structure. Although writing a new schema is a relatively easy job and transforming the database to match the schema can be accomplished with a modest effort, transforming the numerous programs which operate on the database often requires enormous effort. This interim report describes previous research, defines the problem and proposes a framework for research on the automatic conversion of database programs to match the schema transformations. The approach is based on a precise description of the data structures, integrity constraints, and permissible operations. This work will help designers of manual and computer aided conversion facilities, database administrators who are considering conversions and developers of future database management systems, which will have ease of conversion as a design goal.

DISCLAIMER

This technical report is authored and presented by the Database Program Conversion Task Group of the CODASYL Systems Committee to interested members of the computing community for discussion and comment. This report neither states the opinion of, nor implies the support of, the sponsoring institutions. It is the intent of CODASYL that their work be available and in the public domain. Anyone reproducing all or part of this report is requested to include this notice where meaningful.

*Chairman's Address: K52/282, IBM Research Laboratory, 5600 Cottle Road, San Jose, Ca., 95193.

1. Introduction

Conversion of computer systems is a problem most users would rather not think about. Everyone has heard horror stories about conversion: high costs, protracted delays, or spectacular failures. A common attitude in the industry is to put off all conversions until the last possible moment in the hope that perhaps the conversion will be unnecessary after all or, at least, all aspects of the conversion can be handled at once. However, it is becoming increasingly clear that this view of conversion is outdated and unrealistic. Conversion costs can be a major factor in acquiring new equipment, planning for new technologies, or redesigning a database and application system. The U. S. General Accounting Office has estimated that 450 million dollars were spent within the Federal Government on conversion during fiscal 1977 and that 100 million dollars of this expenditure could have been saved through proper organization and planning (1). In a deeper sense, as Mayford Roark has observed (2), "change is the essence of our work": it should not be surprising that conversion is a recurrent and not a "one-shot" event. And if change/conversion is the normal state of affairs, then it makes sense to study it, plan for it, and develop tools to cope with it.

The conversion of a database application system, including both the programs and data, generally begins with the alteration of the database schema. This is followed by converting the data to reflect the new schema and by changing the database application programs to manipulate the new structures. In recent years, considerable research has been carried out to provide high level tools for data conversion (3, 4, 5, 6, 7). Experiments with prototype data conversion systems indicate that substantial productivity gains are possible by using these new tools. Until recently, relatively little was known about the program conversion problem, but there has been a growing research interest (see section 2). Except for some COBOL-to-COBOL or assembly language to COBOL packages, there are few software tools to aid users when they face the conversion of database application programs.

CH1406-8/79/0000-0299$00.75

This paper deals with the database program conversion component of the system conversion problem. A database program is an application program which runs in a database environment. The purpose of this paper is to identify the significant research problems associated with converting database programs and to present a framework within which these research problems can be studied. The paper also discusses existing research in database program conversion and identifies potentially fruitful areas for further work, particularly those which can be automated, at least partially.

1.1 Problem Statement

A *database program* is (1) a program written in a conventional programming language, with embedded data manipulation statements which interact with a database system, or (2) a statement or series of statements in a query/update language of a self-contained database system. Thus, a COBOL program containing embedded CODASYL DBTG DML statements or embedded calls to a vendor's database management system is a database program. So, too, are queries expressed in the query language facility of System 2000, for example. We assume each database program takes the database from a consistent state to a (potentially) different consistent state. No program leaves the database in an inconsistent state and depends on some other program to restore database consistency. The Database Program Conversion Task Group has focussed on the *database program conversion problem,* which may be stated as follows:

> 1. Given a database program and a database schema with which that program interacts,
>
> 2. Given also a new database schema and a definition of a restructuring to some new (logical) form,
>
> Then, to what extent is it possible to develop a computerized methodology which can aid in the conversion of the database program so that it "runs equivalently" against the restructured database?

This is far less ambitious than the "general" program conversion problem, which may well be undecidable in the sense that notions of equivalent behavior may be impossible to prove. Yet the task group feels that a solution to even this restricted problem would yield major benefits, especially as the number of database programs increases. The Task Group has not dealt with conversion of programs in a non-database setting. Preliminary investigation indicated the conversion problem only seemed tractable if the conversion system could guarantee preservation of program behavior under data restructuring. This in turn requires a declaration of the program's assumptions about the database (such as in an external schema). The conversion system must also be able to discover how the program is accessing and using the data; much of this information is explicitly available if the database is manipulated by a database data manipulation language. In non-database programs, assumptions about the data are defined in the procedural logic of the statements, and it is common to use programming techniques such as storage equivalencing and side effects in order to accomplish certain data manipulation operations. The task group felt that this sort of behavior would be very difficult to deal with in a conversion system, so consideration of non-database programs was dropped.

The notion of "runs equivalently" plays a key role in our problem statement; therefore, it is important to clarify the intent of this phrase. In a conversion, the functions performed with the database in the original application program must also be accomplished by the transformed target program, though not necessarily in the same way. However, in addition to interacting with a database, the program may also read and write non-database files and interact with a person at a terminal. In considering this problem, the task group has adopted an operational definition of what constitutes "equivalent behavior" of the program when running against the restructured database. The rule is that except with respect to the database, a restructured program must preserve the input/output behavior of the original program. In particular, this means that with the exception of database operations, the input/output behavior of the restructured program must be identical to the operation of the original program before restructuring of the database. With respect to a person operating the program from a terminal, the program must give the same requests and/or messages as before conversion. The program must also present the same series of reads and writes to non-database files as it did before conversion. However, a different combination of interactions is acceptable with respect to the database including auxiliary database storage such as log files.

This is a very strict definition of a "conversion", and many practical "conversions" do not fit our definition because they do not preserve equivalent behavior in the strict sense. Rather, practical "conversions" often contain a combination of application redesign together with conversion as defined above. However, we separate conversion from redesign in order to have a precise measure of when a program has successfully been converted. Our definition assumes that all the information in the source database has been preserved in the restructured database. If this is not the case, then it may not be possible to convert all programs. Con-

version when not all information is preserved is a different and more difficult conversion problem.

A database application system is converted when each program actually existing in the source system has been converted. No attempt is made to characterize all the possible inferences that could have been made from information in the source database, nor is there any attempt to show that any possible program which could have been written for the source database could also be written for the target database. Rather, the attempt is to capture and preserve the input/output behavior of those programs which actually existed in the source application system.

1.2 Potential Benefits of a Database Program Conversion System

While there are numerous research problems to be solved before a usable database program conversion system becomes a reality, there are many potential benefits. They can be broadly classified into (people) productivity benefits and (machine) performance benefits. Each benefit stems from the labor-intensive, costly, and often unreliable process of manually converting the schema, database, and programs.

"Program maintenance" now accounts for a large part, if not the majority of activity in a programming shop. Maintenance is much more than correcting errors. It is keeping up with new releases of software, taking advantage of new hardware, and responding to changes in application requirements. Whenever code is modified as part of the maintenance process, it must be re-tested in the overall context of the application system. This takes time--time to understand the program, time to change it, time to test and time to verify and/or debug the maintenance change and time to update the documentation. A database program conversion system as discussed in this paper will not "solve" the complete maintenance problem, but it will significantly reduce parts of it, by guaranteeing that certain standard transformations can be made to the database with all affected programs behaving appropriately. This should allow database administrators to take advantage of new technologies more quickly because the conversion cost will be less. It should also allow more rapid response to new application requirements.

With respect to performance, there are a number of advantages that a system like this could provide. For example, programs written to run in an early release of a database system could be "converted" to take advantage of new system features as they become available. Also, it is generally acknowledged that significant performance improvements are possible with a new database design once the performance flaws in a previous design are understood. At present, database design research has not reached the point where all aspects of database performance can be predicted, nor do systems provide data independence at a level which allows wide flexibility in performance tuning. Thus a database program conversion system could complement a database design system in an important way by encouraging database redesign to improve performance.

A further potential benefit of this effort is a better understanding of what is and is not convertible in the database environment. Guidelines for program and database design which take into account anticipated conversion are also expected. Finally, a potentially great benefit may be a set of recommendations for the design of future database management systems to make conversion easier.

2. Current Approaches to Database Program Conversion

The current approaches to database program conversion fall into two major categories, manual and computer-aided. Conversion strategies reflecting the manual approach tend to be "brute force operations", performed only once, often by inexperienced personnel. On the other hand, conversion strategies based on the computer-aided approach have a defined methodology with experienced personnel using software tools.

In this section we discuss the main conversion strategies employed by industry and under development in research laboratories or universities. Although some of these techniques are commonly used, very little documentation exists.

By far the majority of conversions are manual endeavors. These are not usually database program conversions but rather assembly language to COBOL and COBOL to COBOL. Only recently have COBOL to DBMS and even some DBMS to DBMS conversions been undertaken. Unfortunately, the manual approach, which is labor intensive and costly, represents the state-of-the-art in many installations.

2.1 Computer-aided Approach

Strategies in the computer-aided category can be further divided into operational and experimental. The few operational strategies are employed by industry to perform actual conversions, whereas the experimental are prototype strategies being developed by research laboratories and universities.

2.1.1 Operational Computer-aided Strategies

Operational computer-aided conversion strategies include proprietary vendor products, parameterized computer aids, and vendor conversion teams. Fixed price conversion contracts are available from some vendors for assembly language to COBOL and COBOL to COBOL conversions of non-database programs. They are computer-aided in that a number of software tools have been developed: parameterized file convertors, parameterized file comparators, and language translators. These strategies are automated to the extent that they achieve a 65-70 percent success rate (sometimes higher) in assembly to COBOL and COBOL to COBOL program conversions. When a conversion cannot be done, often the software tool will mark the portion of the program that failed, and then the conversion is completed by hand.

Some of the hardware vendors have trained conversion teams which travel from one installation to another converting brand-x's data and programs to their own. Over a period of years, the conversion team out of necessity develops considerable expertise and builds a number of conversion aids.

Vendors sometimes provide a compatibilty interface to, say, a new access method. This is a type of conversion aid in that it provides a conversion path for existing programs until such time as they are rewritten.

2.1.2 Experimental Strategies

There are a number of experimental conversion techniques, some of these have been implemented while others just exist on paper. The techniques mentioned here deal with the database program conversion problem.

DML Emulation

The DML emulation strategy preserves the behavior of the application program by intercepting the individual DML calls at execution time and invoking equivalent DML calls to the restructured database. The Honeywell "Task 609" project implemented a software package which enabled an application program to run on a restructured IDS database (8). A mapping description was used to provide the transformation from the original to the restructured database; this description was used to produce subroutines to perform the correct access in the restructured database. Some of the limitations of this prototype were: 1) retrieval only--no update allowed, 2) only IDS/I databases, 3) not all restructuring operations addressed.

Bridge Program

A second experimental strategy is sometimes referred to as a bridge program method. In this strategy, the source application program's access requirements are supported by dynamically reconstructing from the target database that portion of the source database needed. Data reconstruction is done by means of the "bridge programs". In the limiting case of the bridge program being invoked at the DML call level, this approach reduces to the previous one. The source program operates on the reconstructed database to effect the same results that would occur in the original database. A reverse mapping is required to reflect updates and each simulated source database segment that has changed must be retranslated along with any new database members. Differential file techniques can be used to ease this process (9). An example of a dynamic restructuring capability is provided in the WAND system (10).

The DML emulation strategy becomes extremely complicated when dealing with complex data structures. Such a situation may require the conversion software to evaluate each DML operation against the source structure to determine status values (e.g., currency) in order to perform the equivalent DML operation on the restructured database. Other situations require the maintenance of run time descriptions and tables for both the original and restructured database organizations, the interception of all original DML calls, and the utilization of old-new database access path mapping descriptions (human input) and rules.

Both these strategies, though straightforward in concept, have drawbacks of degraded efficiency and restrictiveness. Efficiency is degraded in the emulation strategy because each source DML statement must be mapped into a target emulation program, which uses the restructured database to achieve the same results. In the bridge program strategy, a subset of the target database must be dynamically restructured. The increased overhead in program size and/or access path length can result in a significant increase in processing requirements.

The drawback of restrictiveness comes about because the emulation and bridge program strategies probably cannot utilize the increased capabilities of the restructured database. By mimicking the behavior of the program at a detailed level, it is unlikely that new access strategies can be used. This approach may also limit the class of restructurings that can be done.

2.2 Current Research

Current research aims toward developing more generalized tools for computer-aided conversion. The drawbacks of the existing strategies described above

can be avoided by "rewriting" the application programs (using the conversion system) to take advantage of the restructured database.

Research on database application program conversion is still in its infancy and there are few published papers on this subject. This section summarizes early work in the area, excepting the work of Task Group members, which is discussed in section 4.

Mehl and Wang (11) presented a method to intercept and interpret DL/I statements to account for changes in the hierarchical order of an IMS structure. Algorithms involving command substitution rules for certain structural changes were derived to allow for correct execution of the old application programs. The approach is similar to the emulation approach discussed earlier and has the consequent drawbacks, however the work did have some optimization strategies included.

The work by Housel (12) is an extension of the work on application migration undertaken at the IBM San Jose Research Laboratory. This work uses a common language for specifying the abstract representation of source programs. The language is a subset of CONVERT (13) plus some of Codd's relational operators. The operators of the language are designed to have convenient algebraic properties to facilitate program transformation. They are designed to handle data manipulation in a general hierarchical structure called a "form". In this system, program transformation is dictated by the data mapping operations applied to the original database. It is assumed in the proposed model that the inverse of these data mapping operators exists, i.e., the source database can be reconstructed from the target database by applying some inverse operators. The program conversion is performed by substituting the inverse operators into the specification language statements for each reference to the source database. This process is followed by a simplification procedure. The author observes that the assumption of the existence of inverse operators restricts the scope of the conversion problem that can be handled in the proposed approach.

3. Difficulties in converting current Database Programs

As discussed in detail in section 4, the proposed framework emphasizes the use of program analysis and high-level data modelling and restructuring operators to accomplish database program conversion. These sophisticated tools are necessary in a conversion system because of the intricacies of current database systems and their database programs. This section gives examples of programs that are difficult to convert. The examples help to illustrate the scope of the problem and show what must be dealt with in a realistic conversion situation. The examples reveal "features" to be avoided when possible in current systems in order to facilitate future conversion of programs. The two major problems in converting database programs are inadequate representation of application requirements and dependence on execution time variability.

3.1 Inadequate representation of application requirements

In the current data models -- the relational, owner-coupled-set and hierarchical -- it is possible to specify the structure of a database in a relatively representation free way. The objects and their interrelationships in the database are made explicit in these specifications. This representation independence greatly facilitates the development of data translation systems and forms a necessary base for database program conversion systems. However, the specifications do not provide all that is needed. The single most significant deficiency in the existing models is their inability to model integrity constraints to the degree needed. Any system is governed by many different types of constraints (14, 15, 16, 17, 18, 19, 20, 21). We will consider only a few of these and their impact on the database program conversion problem.

One type of constraint governs when an instance of a relationship can (and should) exist. For example, consider the relationship "course-offering" that holds between a "course" and a "semester". This structure is illustrated in Figures 3.1 a and b. One possible way of constraining it is to say that a "course-offering" instance cannot exist unless the "course" and "semester" instances it references do. In particular, CNO and S can not have null values.

COURSE-OFFERING(CNO,S,....)
COURSE(CNO,CNAME,....)
SEMESTER(S,YEAR,....)

Figure 3.1a Relational School Database

Such constraints can be and are maintained by the programs that access the database. These constraints must be maintained if the database and its access programs are converted. A programmer can make use of knowledge about value distributions to avoid a test and yet be able to guarantee database correctness. Simple changes in either the structure or value-set of the database can invalidate the assumption.

This problem could be reduced significantly if constraints could be reduced in program logic and centralized, explicitly, as part of the data model. This is

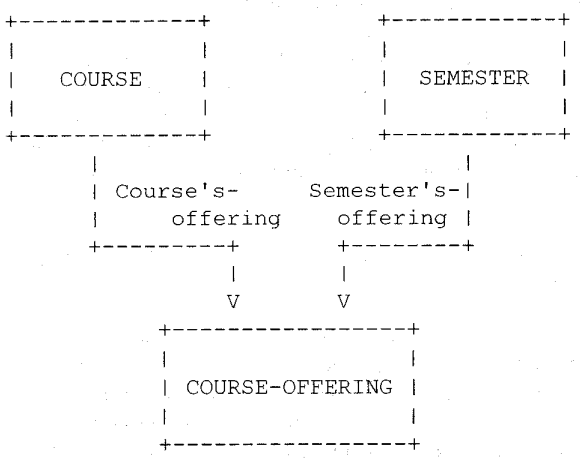

Figure 3.1b CODASYL School Database

possible only to slight and different degrees in the different data models--but nowhere sufficiently. The only constraint maintained explicitly in the relational model is tuple uniqueness (by means of key declarations). The owner-coupled-set model provides the AUTOMATIC/MANUAL and OPTIONAL/MANDATORY mechanisms for capturing some existence constraints. For example, if a "course" instance and a "semester" instance must exist in order for a "course offering" to be inserted, then "course offering" can be made an AUTOMATIC and MANDATORY member in the "course's offering" and "semester's offering" sets. This guarantees that if an attempt is made to insert a "course offering" for which there is either no corresponding "course" or "semester", the insertion will fail. However, if a "course offering" may or may not have an instructor when it is inserted, the closest we can come to having the system enforce membership is to create a "null" instructor. This may capture the essentials of a valid insertion but database inconsistency may still occur due to the operation of the DELETE (ERASE) command. The DELETE (ERASE) command has an option which could cause deletion of "course offerings" when instructors are deleted. This violates the system's integrity constraints. Another constraint not maintained by any of the models is that of numeric limits on relationship participation. For example, in the school database there may be a rule that a course may not be offered more than twice in a school year. In all existing models, a constraint like this could only be maintained by user programs. In general, constraints can be arbitrarily complex. It is desirable to design a conversion system that can deal with embedded integrity constraints; how to do this is an open problem.

A data model that captures constraints and behavior as well as structure would provide a basis for building databases and database programs that are much easier to convert. Considered as a conversion tool such a model would provide an intermediate form for converting a source system to some target. This form would be used as the target for the decompilation process and the source of a compilation process to produce the target system.

3.2 Execution time variability

Database programs can change their behavior relative to the database during execution. For example, some database systems which use a call interface to interact with the database, pass the request (retrieve, insert, etc.) as an argument. This argument is usually a program variable and thus potentially can change during execution; what appeared to be a read at compile time might become an update because the request parameter is changed. Thus any software which attempts to understand the program's behavior from a source language version of the program must (through data flow analysis techniques) make sure that the commands do not vary at run time. If there is potential run time variability, then it may be impossible to proceed further. As another example, the program may depend on records from the database being presented in a given order (order dependence) or a programmer may have intended to "process all" dependent records of a certain type, but may have written a program which will "process the first" dependent. The behavior is the same if it is known a priori that there will be only one dependent, but the application requirements could be confused. As a third example, it is easy to write programs which depend on certain status codes being returned by the database system but certain restructurings the system will cause a different status code to be returned. The difficulties raised above are *not* unique to navigational and/or hierarchical systems. Some relational systems (22) allow considerable run time variability in database commands, and status code dependency is a possibility in all systems.

Examples such as these point out that a completely general, system independent solution to this problem may not be possible. A completely automated system is probably not possible, and an interactive system makes more sense. It may turn out that pathological cases such as those discussed above do not occur frequently in practice, or are disappearing as more programs are written using development techniques which emphasize clarity, maintainability, convertibility, and straightforward use of system features.

4. Framework for Database Program Conversion

The framework for database program conversion is shown in figure 4.1; it is adapted from a model developed by Su (27) and by Su, Lo and Lam (23). The system is intended to be interactive and controlled by a Conversion Analyst interacting with the Program Conversion Supervisor. For database program conversion, the database description includes the schema description of the source and target databases and an extended description of database operations and integrity constraints as well as a description of any changes in integrity constraints between the source and target schemas. These inputs are needed to analyze application programs.
The Conversion Analyzer analyzes the source and target databases in order to classify the types of changes that have been made and to encode the descriptions in suitable internal representations. The Program Analyzer uses the source database description and matches candidate language templates against the source application program to produce a representation of the database operations and data access patterns made by the program. The internal representation of the program also includes the program control structure, the relationships among program variables and the sub-program parameter passing structure. The language templates are data manipulation language and/or host language sequences which carry out data access and manipulation operations which are meaningful and consistent with the source database schema.

The internal representation of how the database schema has been changed is used by a Program Converter to select the proper transformation rules for use in mapping the source program representation to the target program representation. The selection of transformation rules is also based on the description of the source and target databases. The transformation rules map the access patterns and the application program structure to account for the database changes made. The target program's representation is further processed by an optimizer which refines the representation, improving access paths, algorithms, and data handling. The optimized target program representation is used by the Program Generator to produce a target program.

During the entire program conversion process, a monitor program, the conversion program manager, oversees the operation of the other modules. We expect that an interactive system would be most successful, in resolving issues of database integrity and application program requirements that are raised in this paper. For example, if data referenced by an old program has been deleted or multiple data paths can be found to carry out an access then these issues can be resolved interactively.

Following the general framework described above, members of the Task Group are in the process of designing and/or implementing prototype database program conversion systems. Of course, different researchers have placed emphasis on different parts of the problem. The remaining portion of this section summarizes these research efforts.

4.1 University of Florida

The work at the University of Florida, led by Su, uses a data model (24) for defining the source and target databases. The model contains a number of constructs whose operational characteristics and integrity constraints have been explicitly defined. The intent is to include in the data model a number of constructs whose properties are generally enforced procedurally in application programs. Changing from one construct to another in this model during data base conversion will mean changes to the associated operations and integrity rules. Thus, an application program which operates on the old construct will be modified following the operational characteristics and integrity rules of the new construct. For example, the dependency property of EMP.DEPENDENT to EMP is explicitly represented in the model by calling EMP a "defined entity" and the DEPENDENT a "characterizing entity". Deletion of an employee implies deletion of dependents. If the schema is changed so that dependents are now defined entities with a non-dependency relationship, then the system will insert statements to traverse this relationship and continue to enforce the dependency relationship that was assumed in the old program.

The rules for changing the operations as the result of schema changes are called transformation rules. These rules can be formulated if the structural properties, operational characteristics and integrity constraints of the data are given explicitly in the data model. A set of transformation rules has been defined and verified to account for a number of database schema changes (23).

To convert an application program, it is necessary to have a way of modeling and representing the requirements of the program. The approach taken by Su's group is to describe application programs in terms of sequences of access patterns to be performed on the network of association types as represented in the high level data model. Four basic access patterns have been identified. The pattern *Access A via A* means accessing some data fields defining an entity type A based on some data conditions of the fields defining A. If two entity types A and B are not related by an association, the only way of relating the data of these two entity types would be by taking the mathematical relation of their comparable data fields; thus, a second

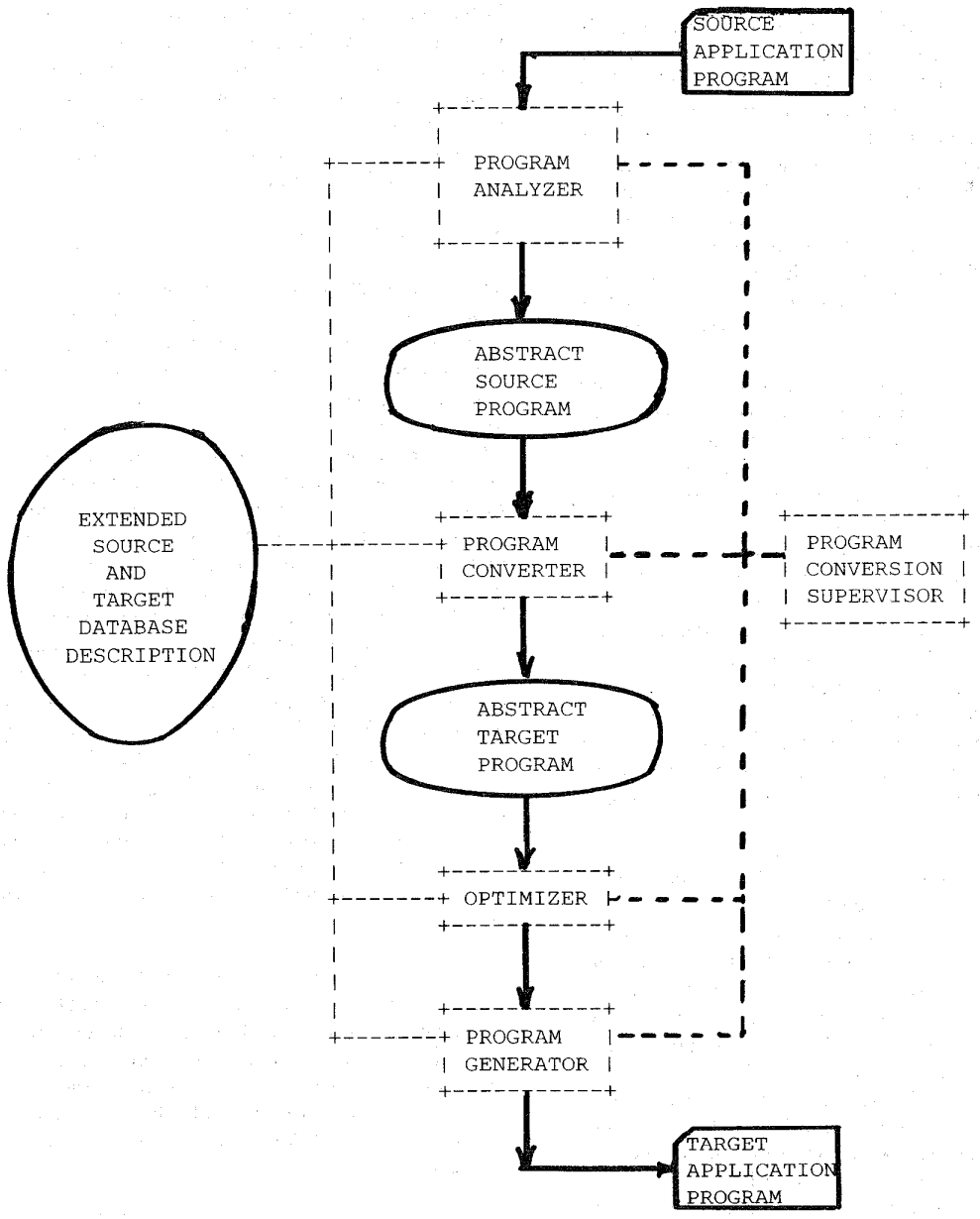

Figure 4.1 Database Program Conversion Framework

access pattern is *Access A via B through (Ai, Bj)* where Ai and Bj denote the comparable fields. If entities A and B have an association AB, then data accesses can be through their association such as *Access AB via B* and *Access A via AB* meaning some occurrences of AB are first accessed via the data condition of B and the accessed occurrences are further used as constraints to access the occurrences of A. A sequence of these basic access patterns can be used to describe the traversal of data specified in the application program. For example, given the entity types

EMP(E#,ENAME,AGE)
DEPT(D#,DNAME,MGR)

and their association

EMP_DEPT(E#,D#,YEAR_OF_SERVICE),

the data traversal and operation expressed by the query "Find the names of employees who work for

Manager Smith for more than ten years," can be described by the following sequence of access patterns:

```
ACCESS DEPT via DEPT
ACCESS EMP_DEPT via DEPT
ACCESS EMP via EMP_DEPT
RETRIEVE
```

These access patterns are very general and are independent of how the entity types A and B and their association AB are represented in the schema of the DBMS. However, in order to derive this representation of the data traversal and operation in an application program, it is necessary to have a schema and data model dependent representation of the traversal. In Su's work, the data model dependent representation, called an "access path graph" (25, 26), is used to describe how a data traversal can be interpreted in the relational, network, or hierarchical model. For these schema and data model dependent representations, a set of language templates are defined. For example, the access pattern

```
ACCESS EMP via EMP_DEPT
```

contained in the query "Get the names of those employees who have worked for department D2 for three years" would be expressed in SEQUEL as (A) below and as (B) below in a CODASYL system.

```
SELECT ENAME
FROM   EMP
WHERE  E# IN
       SELECT E#
       FROM EMP-DEPT
       WHERE D# = 'D2'
          AND YEAR-OF-SERVICE = 3
```

```
MOVE 'D2' TO D# in DEPT.
FIND ANY DEPT.
IF no such occurrence is found
   GO TO NOTFD.
MOVE 3 TO YEAR-OF-SERVICE IN EMP.
NEXT. FIND NEXT EMP WITHIN ED USING
            USING YEAR-OF-SERVICE.
      IF no other occurrences
            GO TO FINISH.
      GO TO NEXT.
```

The set of language templates are used by a language analyzer to match against the application program in order to identify the access patterns involved in the data traversals. They are also used by the language synthesizer to generate the object program. Analysis of DBTG's DML language templates is presented by Nations and Su (26).

Since the conversion takes place at a level of abstraction that is removed from an actual DBMS language, conversion from one DBMS to another to account for some schema changes is possible.

A prototype application program conversion system is under development.

4.2 University of Maryland

At the University of Maryland, the approach has been to create a new DDL and DML which would be familiar while facilitating conversion (28). In this way, the difficult problem of analyzing the complex data manipulation operations that exist in current database management systems could be avoided.

The DDL permits owner-member-coupled sets with single owner and member record types. Each set type description indicates an ordering for the member record instances. Duplicates are not allowed within a set occurrence. Figure 4.2 shows a simple schema diagram which is described by the schema in Figure 4.3.

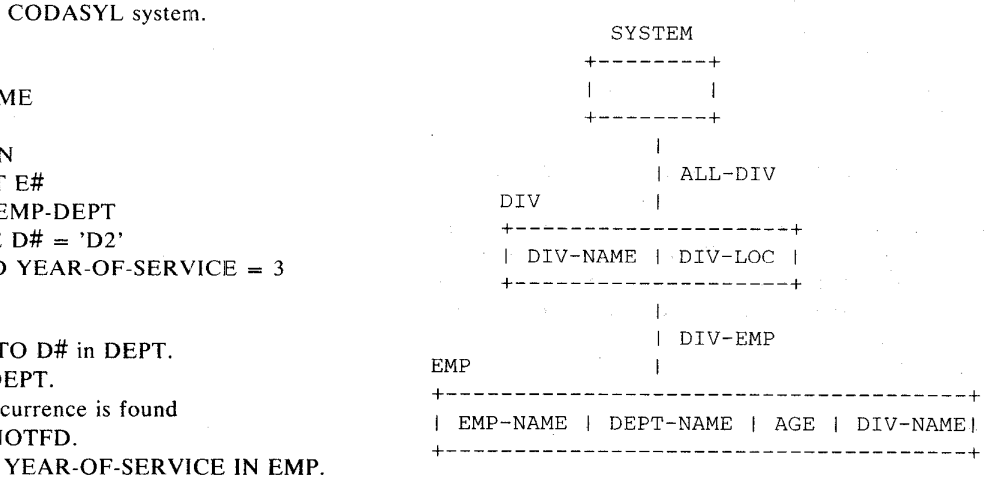

Figure 4.2 Sample Data Structure Schema

The DML permits retrieval operations which return collections of records of a single record type, accessible to the user in the host language program. The retrieval is specified by a FIND statement which indicates the target record type and a qualified access path. The access path begins with a SYSTEM owned set or a collection of previously retrieved target records. If necessary, the path can be extended by set name and record name pairs. Record names or collec-

```
SCHEMA NAME IS COMPANY-NAME
   RECORD SECTION;
       RECORD NAME IS DIV.
       FIELDS ARE.
          DIV-NAME        PIC X(20).
          DIV-LOC         PIC X(10).
       END RECORD.

       RECORD NAME IS EMP.
       FIELDS ARE.
          EMP-NAME        PIC X(25).
          DEPT-NAME       PIC X(5).
          AGE             PIC X(2).
          DIV-NAME        VIRTUAL
                          VIA DIV-EMP
                          USING DIV-NAME.
       END RECORD.
   END RECORD SECTION.
   SET SECTION.
       SET NAME IS ALL-DIV.
           OWNER IS SYSTEM.
           MEMBER IS DIV.
           SET KEYS ARE (DIV-NAME).
       END SET.

       SET NAME IS DIV-EMP.
           OWNER IS DIV.
           MEMBER IS EMP.
           SET KEYS ARE (EMP-NAME).
       END SET.
   END SET SECTION.
END SCHEMA.
```

Figure 4.3 Schema Declaration for Fig. 4.2.

tions can be qualified by boolean expressions referring to fields in the record. The output of one retrieval statement can provide input for another retrieval statement. The following two examples based on the schema diagram for Figure 4.2 illustrate the FIND statement:

1) Find all employee records for employees whose age is greater than 30.

```
FIND(EMP: SYSTEM, ALL-DIV, DIV,
          DIV-EMP, EMP(AGE > 30)).
```

2) Find all employee records for employees who work in the 'SALES' department of the 'MACHINERY' division.

```
FIND(EMP: SYSTEM, ALL-DIV,
          DIV(DIV-NAME = 'MACHINERY'),
          DIV-EMP,
          EMP(DEPT-NAME = 'SALES')).
```

STORE, DELETE, MODIFY and other operations are provided.

A conversion is considered as a sequence of transformations applied to the source schema which produces a target schema. These same transformations are also used to translate the database and to convert the DML statements written for the source schema. The goal is to preserve input/output behavior for the database system user, that is, the old programs running on the old database should produce identical results to the converted programs running on the new database.

A small number of simple transformations have been defined which can preserve input/output equivalence. It is hoped that more complex transformations can be built up from these. As more experience is gained, more powerful transformations will be added, provided it is possible to prove preservation of input/output behavior. For example, the transformation of Figure 4.2 to Figure 4.4 is probably possible.

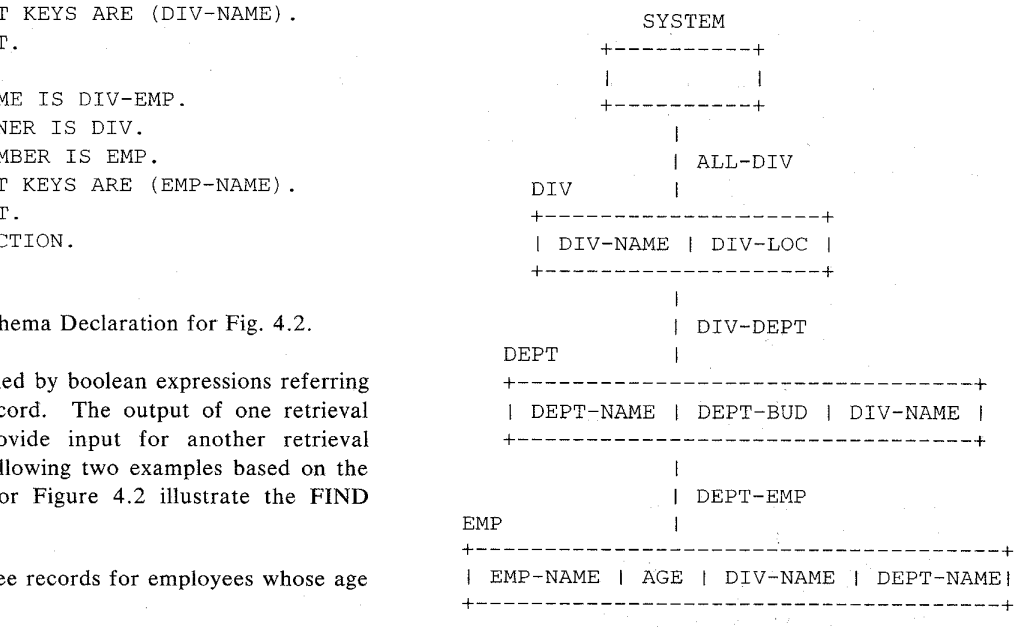

Figure 4.4 Revised Schema of Figure 4.2

The FIND statements shown earlier can be transformed into:

```
SORT(FIND(EMP: SYSTEM, ALL-DIV, DIV,
       DIV-DEPT, DEPT, DEPT-EMP,
       EMP(AGE > 30))) ON (EMP-NAME).

FIND(EMP: SYSTEM, ALL-DIV,
       DIV(DIV-NAME = 'MACHINERY'),
       DIV-DEPT,
       DEPT(DEPT-NAME = 'SALES'),
```

DEPT-EMP, EMP).

The DDL and DML syntax have been defined and a prototype system is being constructed. The goal is to gain an understanding of which transformations can be realized by a database management system designed especially for conversion. This might suggest directions for future database architectures.

4.3 University of Michigan

Beginning in 1971, the Database Systems Research Group (DSRG) at the University of Michigan conducted extensive research on database conversion. The effort produced results in data translation architecture, stored-data definition languages and prototype implementations (4). DSRG researchers began exploring the database program conversion problem in 1973. To date, the problem has been approached from three principal perspectives: DML primitives, code templates, and high level specification languages.

In the initial studies, Delong and Fry (29) and Fry and Schindler (30) investigated DML primitives as an approach to database program conversion. This method contrasts with the general framework of figure 4.1 in that, rather than decompiling a database program into a high-level representation, the program analyzer expresses individual DML statements in terms of more primative DML functions. The program converter associates source program primitives with primitives applicable to the target DML program.

The DML primitives approach was attractive for converting database programs from one DBMS to another since it did not rely on a detailed understanding of the application program and it's relationship with the schema. To test the idea, programs written in three different DML's were analyzed and a set of DML primitives was identified and classified. However, as work progressed, two serious problems arose with the DML primitives approach. First, the considerable diversity discovered in the low-level functions of different DBMS's began to suggest that it might not be possible to develop a manageable set of DML primitives comprehensive enough for use in a generalized database program conversion system. Second, DML primitives did not seem suitable for database program conversions done in response to changes in the logical structure of data. Largely for these reasons, DSRG dropped DML primitives research in favor of code templates.

The code template approach fits the general database program conversion framework of figure 4.1. Code templates are predefined sequences of host language DML statements (similar to macros) which implement a set of high level data manipulation operations. Each code template corresponds to a operator in the relational algebra. Application programs are written using nested code templates. Schindler (31) addressed the problem of developing program conversion techniques to transform a program which had been written using code templates into a target database program which accounts for the effects of a database restructuring. High-level program conversion is accomplished by using relational algebra specifications for the data conversion to transform relational algebra specifications for the templates. The approach is similar to that proposed independently by Housel (12).

While the code template approach is appealing, there are a number of challenges to be met. First, the problem of decompiling an arbitrary host language program which does not use code templates is a open problem, as discussed further in section 5.3. A second challenge is to extend the approach to handle updates as well as retrievals. Operations in relational algebra are still dependent on a given view of the set of relations being processed, and under certain restructurings, updates may be ambiguous. This is similar to the well-known view update problem. Also, there are open questions on how difficult it is to optimize symbolically relational algebra expressions. These challenges indicate that an approach based strictly on the relational algebra may not be sufficient.

A key problem to be solved within the framework is the development of a high level specification language for database programs. The language could be used instead of the relational algebra as the intermediate language in which the abstract source and target programs of figure 4.1 are expressed. The language could also be used independent of the conversion context in the specification of new applications. These new applications would then be less sensitive to database restructurings and would be able to by-pass the program analysis stage when they are subsequently converted. Current research includes the search for an appropriate high level DML/DDL pair and studies of program generation techniques for mapping high-level database programs into programs expressed in existing DMLs, such as CODASYL DBTG or IMS.

5. Research Problems to be Investigated

The approaches to database program conversion just described have shown the need for continued research before practical database program conversion systems can become a reality. This section discusses some of the research directions in database program conversion. These research problems overlap with research issues in data models, data base design, application program development and software engineering. We can expect results from these areas to be useful in database program conversion systems. At the same

time, these systems should generate new results as the diverse parts of the problem -- program analysis, high level data models, and optimization -- are meshed in a coherent system. Each of the major research areas will be discussed.

5.1 Improved Description of the Database and Application Programs

Based on the examples of section 3, it is clear that a database program conversion system needs more explicit information about the data objects in the source and target systems and their behavior under allowed operations. If conversions are to span data models, then the representation of programs may have to account for idiosyncrasies of the source system (e.g. currency behavior) and the representation of this behavior in the target system. The representation of the database structures will have to be at a level which is high enough to be realized in either data model. It will be necessary to study classes of meaningful changes to database schemas, determine their effect on application programs, and catalog their interpretations in existing DBMSs.

5.2 Equivalence of Source and Target Programs

One fundamental problem in database program conversion is to define source and target program "equivalence". While I/O equivalence has been emphasized in this paper, there are other kinds of equivalence which may be more appropriate in certain situations. As an example, suppose employees who retired prior to 1950 are deleted during conversion. The the converted program which prints all current or prior employees is not strictly I/O equivalent to the program before conversion. Yet we would probably want a conversion system to convert the "print all employees" program successfully, though perhaps a warning should be issued. As another example, suppose a schema at one point in time allows an employee to have no associated department, then the schema is changed to require each employee to have a department. A program to insert employees may not have the same behavior as previously (it now succeeds only if the department is non-null). This is the desired behavior because the application requirements have changed, but it is not strictly equivalent.

The point is that there are probably levels of "successful conversion" and research remains in identifying what level can be preserved for what kinds of database restructurings.

5.3 Analysis, Transformation and Synthesis of Programs

The design and implementation of a usable program analyzer is a major challenge, but the successful construction of this tool could yield major benefits. As described earlier, the program analyzer uses language templates and dataflow analysis techniques to identify and create from application programs a high level representation of the program's operation on the database. In order to do this, it will be necessary to pick an appropriate level of template matching and perhaps develop a template definition language for experimentation. A significant challenge will be to develop very high level templates. By doing this, it may be possible to extract less procedural representations of the program's interaction with the database and gradually move away from an access path orientation. However, use of the proper template in the proper situation is a very context dependent decision and may require sophisticated algorithms in the program analyzer. Large classes of programs will have to be analyzed to become convinced that the set of templates is widely applicable.

Another open problem is to determine whether the program analyzer can detect database integrity constraints that are enforced procedurally in the program (or when they are not but should be).

If a program analyzer can be successfully constructed, it could be used as a programmer's aid during initial writing of database application programs. Application programmers may misunderstand or misuse data relationships, resulting in erroneous or inefficient programs. For example, a programmer may try to relate two files through two data items which are not related in application terms. Or the programmer may not be aware of all the access paths available for traversing from one data entity to another. Program "improvement" of this kind should be a natural byproduct of a good program analyzer.

5.4 Optimization of application program representations

An optimization needs to be performed on the application program representation for the following reasons: (1) the original source program may not be efficiently coded or (2) an efficient application program may become inefficient after both the database and the program have been converted; the target program needs to be optimized to take advantage of the new data relationships in the target database.

If a high enough level of template can be used in the program analysis phase, then the optimization problem is closely related to the access path selection problem in database management systems (32). However, if the program representation is at a lower level, then further work needs to be carried out in this area.

6. Conclusion

The Task Group's objective is to investigate the solutions for the database program conversion problem. We intend to identify what types of program change can be done automatically or semi-automatically and what cannot be handled at all. The effort will contribute not only to helping convert existing application programs, but also will help to define the requirements of future DBMSs and illustrate programming practices which will yield more convertible database applications. The work presented here (1) provides a survey of current approaches and research efforts (2) describes some of the difficulties in program conversion, (3) presents a global model of a program conversion system based on more explicit and higher level database and program desciptions, (4) describes ongoing research undertaken by some members of this group, and (5) proposes some problems for future research. It is the hope of the Task Group that this work will provide a general framework on which future research by this group and other researchers can be conducted.

ACKNOWLEDGMENTS

The research reported herein was supported in part by the National Science Foundation under grants MCS-77-22244, MCS-76-10075, MCS-77-22505 and MCS77-01484. The Task Group would like to acknowledge helpful suggestions and contributions made by R. Marion, C. Cook, S. B. Yao, S. Navathe, B. C. Housel and G. Thomas.

EPILOG

The reader may have noticed that the word "semantic" does not appear in the main body of this paper. The Task Group found that when this term was banned from discussion and prose, program conversion issues could be discussed with much greater precision.

REFERENCES

1. Office of Management and Budget and National Bureau of Standards, "Millions in Savings Possible in Converting Programs from One Computer to Another," Report to the Congress by the Comptroller General of the United States, FGMSD-77-34, Sept. 15, 1977.

2. Roark, Mayford, "Some Approaches to the Management of Change," in *Information Processing 77,* Proceedings of the 1977 IFIP Congress, North-Holland, New York, N.Y., 1977, pp. 109-112.

3. Fry, J. P., et al., "An Assessment of the Technology for Data and Program Related Conversion," *Proceedings of the 1978 National Computer Conference,* Vol 47, AFIPS Press, Montvale, N.J., 1978, pp. 887-907.

4. Swartwout, D. E., Deppe, M. E., and Fry, J. P., "Operational Software for Restructuring Network Data Bases," *Proceedings of the 1977 National Computer Conference,* Vol 46, AFIPS Press, Montvale, N.J., 1977, pp. 499-508.

5. Shu, N. C., Housel, B. C., Taylor, R. W., Ghosh, S. P., and Lum, V. Y., "EXPRESS: A Data Extraction, Processing, and Restructuring System," *ACM Transactions on Database Systems,* 2:2 (June 1977), ACM, N. Y., pp. 134-174.

6. Bakkom, D. E., and Behymer, J. A., "Implementation of a Prototype Generalized File Translator," *Proceedings of the 1975 ACM SIGMOD International Conference on Management of Data,* ed. W. F. King, ACM, N.Y., 1975, pp. 99-110.

7. Ramirez, J. A., Rin, N. A., and Prywes, N. S., "Automatic Generation of Data Conversion Programs Using a Data Description Language," *Proceedings of the ACM SIGFIDET Workshop on Data Description,* Access, and Control, ACM, N.Y., 1974, pp. 207-225.

8. Honeywell Information Systems, "Functional Specification Task 609 Database Interface Package," Defense Communications Agency, Contract DCA 100-73-C-0055.

9. Severance, D. G., and Lohman, G. H., "Differential Files: Their Application to the Maintenance of Large Databases," *ACM Transactions on Database Systems,* 1:3 (1976), ACM, N. Y., pp 256-267.

10. Gerritsen, R. and Morgan, H., "Dynamic Restructuring of Databases with Generation Data Structures," *Proceedings of the 1976 ACM Annual Conference,* ACM, N.Y., pp. 281-296.

11. Mehl, J.W. and Wang, C.P., "A Study of Order Transformation of Hierarchical Structures in IMS Data Bases," *Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control,* ACM, N.Y., 1975, pp 125-140.

12. Housel, B. C., "A Unified Approach to Program and Data Conversion," *Proceedings of the Third International Conference on Very Large Data Bases,* ACM, N.Y., 1977, pp. 327-335.

13. Shu, N. C., Housel, B. C., and Lum, V. Y., "CONVERT: A High-level Translation Definition Language for Data Conversion," *Comm. ACM,* 18:10 (1975), pp. 557-567.

14. Abrial, J. R., "Data Semantics," in *Data Base Management,* Klimbie, J.W. and Koffemann, K. L, eds, North-Holland, Amsterdam, 1974.

15. Fernandez, E. B., and Summers, R. C., "Integrity Aspects of a Shared Data Base," *Proceedings of the 1978 National Computer Conference,* Vol 47, AFIPS Press, Montvale, N.J., 1978, pp. 819-822.

16. Hammer, M, and Mc Leod, D., "The Semantic Data Model: A Modeling Mechanism for Data Base Applications," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* ACM, N.Y., 1978, pp 26-36.

17. Lee, R.M., and Gerritsen, R., "Extended Semantics for Generalization Hierarchies," *Proceedings of the ACM SIGMOD International Conference on Management of Data,* ACM, N.Y., 1978, pp. 18-25.

18. Smith, J. M., and Smith, D. C. P. "Database Abstractions: Aggregation," *Comm ACM,* 20:6 (1977), pp. 405-413.

19. Smith, J. M., and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems,* 2:2 (1977), pp. 105-133.

20. Smith, J. M., and Smith, D. C. P., "Integrated Specifications for Abstract Systems," *IEEE Transactions on Software Engineering,* in press.

21. Stonebraker, M., "High Level Integrity Assurance in Relational Data Base Management Systems," Memo ERL-M473, Electronics Research Lab., University of California, Berkeley, Cal, 1974.

22. Youssefi, K., et. al., "INGRES Reference Manual - Version 6.1," Electronics Research Laboratory, College of Engineering, University of California, Berkeley, Memo ERL-M579, Dec., 1977.

23. Su, S.Y.W., Lo, D.H., and Lam, H., "Application Program Conversion Due to Semantic Changes," Technical Report 7879-2, Computer and Information Science Dept., University of Florida, March, 1978, submitted for publication.

24. Su, S.Y.W., and Lo, D.H., "A Multi-level Semantic Data Model and its Semantic Integrity Control," Technical Report 7778-12, Computer and Information Science Dept., Universiyt of Florida, July 1978.

25. Su, S.Y.W., and Liu, B.J., "A Methodology of Application Program Analysis and Conversion Based on Database Semantics," *Proceedings of the Third International Conference on Very Large Data Bases,* ACM, N.Y., 1977, pp. 327-335.

26. Nations, J. and Su, S.Y.W., "Some DML Instruction Sequences for Application Program Analysis and Conversion," *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data,* ACM, N.Y., 1978, pp 120-131.

27. Su, S.Y.W., "Application Program Conversion Due to Database Changes," *Proc. of the Second International Conference on Very Large Databases,* Brussels, Belgium, Sept. 1976, pp. 143-158.

28.Shneiderman, B., "A Framework for Automatic Conversion of Network Database Programs under Schema Transformations," *Third Jerusalem Conference on Information Technology,* J. Moneta, ed., North-Holland, Amsterdam, 1978.

29. Delong, S. and Fry, J. P., "An Approach to the Migration of DBMS Applications," DSRG Working Paper 900, University of Michigan, June, 1975.

30. Fry, J. P., and Shindler, S., "Towards the Migration of Database Applications," DSRG Technical Report 76-ST1, University of Michigan, April, 1976.

31. Shindler, S. "Templates for Structured DML Programs," Working Paper ST 2.1, Data Translation Project, School of Business Administration, University of Michigan, Ann Arbor, Dec. 1976.

32. Selinger, P.G., "Access Path Selection in a Relational Data Base Management System," *Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data,* ACM, N.Y., 1979.