CONVERSION TECHNOLOGY, AN ASSESSMENT

Two factors make conversion necessary: changes in users' functional requirements and changes in their performance requirements. These changes may necessitate the acquisition of new software and hardware which, in turn, may require changes to the existing data programs.

For example, the acquisition of a DBMS to replace a file management system requires the integration of the original files into a data base system and modification of the programs to interact with it. The replacement of a current DBMS with a new data base system to provide additional functionality may require a different way of logically structuring the information (its data model) and a different kind of language interface. The establishment of a communication network between differing systems to implement data sharing will require dynamic, in real time, conversion of data between different nodes in the sense that the same item will repeatedly undergo conversion as it is needed, or alternatively, it requires conversion of programs when they travel to different nodes to access data there.

Even when the acquisition of new software or hardware is not warranted, changes in a users' functional and performance requirements can require data and program conversion. What makes conversion difficult is the proliferation of data models and levels and styles of DBMS interfaces, internal data representation and hardware architecture.

When conversion is necessary, users will extract data from their source environment and restructure them to the form required for the target environment. While the extraction and restructuring may themselves be complex, these processes are frequently complicated further by the undisciplined nature of the source data. For example, data may exist in duplicate or contain numerous errors and multiple inconsistencies. The whole process of extracting it from its source, "cleaning the data," restructuring it to a desired form and loading it into the intended environment is generally referred to as data conversion or translation.

The panel chairman was James P. Fry, director of the data base research group in the Graduate School of Business Administration at the University of Michigan. Other panel participants were Edward Birss, Hewlett Packard; Peter Dressen, Honeywell Information Systems; Nancy Goguen, Bell Laboratories; Michael Kaplan, Bell Laboratories; Eugene Lowenthal, MRI Systems Corporation; Vincent Lum, IBM Corporation; Robert Marion, Defense Communications Agency; Shamkant Navathe, New York University; Steven Schindler, University of Michigan; Arie Shoshani, University of California; Diane Smith, University of Utah; Stanley Su, University of Florida; Donald Swartwout, University of Michigan; Robert W. Taylor, IBM Corporation, and Beatrice Yormark, Interactive Systems Corporation.

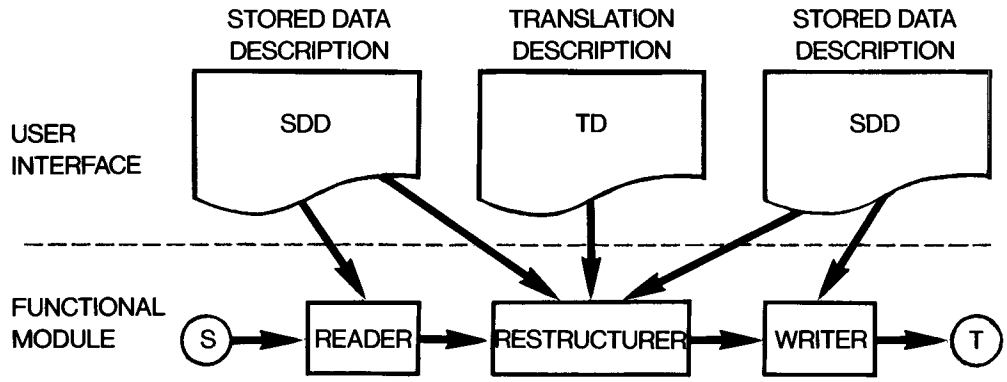

Figure 1. A Data Conversion Model

After a data conversion, particularly one involving extensive restructuring, the application programs which process the original data may not run correctly against the new data. A small amount of restructuring may require only a simple modification, while extensive restructuring may require extensive rewrite or redesign. The process of modifying the programs to process the restructured data is referred to as application program conversion or translation. (This report will not consider the problem of program conversions not related to a change in data structure.)

In brief and conceptual terms, the data translation or conversion process can be represented diagrammatically in Figure 1. As shown in this diagram, a data-translation system generally requires three components: a reader, a restructurer and a writer.

While the capability of each component depends on the individual design of a data-translation system, the reader, in general, accesses data from its source environment to prepare it for further processing. The accomplishment of this process unquestionably requires a description of the data and, thus, a data-description language.

The writer is the functional inverse of the reader, and puts the transformed data into the target environment. It, too, requires a description of the data structure and shares with the reader the need of a data description language.

The restructurer functions quite differently from that of the other two components. This component, in general, extracts data from its source or internal forms and restructures it to a desired format or structure. This process usually requires a translation description language.

In a data-conversion system with a limited application intended, the three components may not be distinguishable, nor the need for the two languages clear. For example, if one wishes to create a conversion system merely to translate EBCDIC characters into ASCII, one can create a simple system with one component and a simple data-description language embedded in it.

However, if development of a broadly applicable data translation system is the goal, clearly one must have a reader and writer capable of accessing and putting out data in all kinds of environments and a powerful restructurer capable of all manners of manipulating data and of creating some data as well. Such a generalized system implies the need for versatile data-translation description languages.

Figure 2 represents a general approach to data base program conversion or translation. To convert a program, one must determine the functions of the program and its semantics. Programmers making assumptions about the state of the data may not, and currently need not, state these assumptions explicitly in their programs. Therefore, one usually must provide more information about the semantics of the program than that provided in the program text and its documentation.

The program conversion process also needs information about the data structure the program originally ran against, the new structure it must run against and how the two relate. These data descriptions could be the same as those used to drive the data-translation process. The program, the description of its semantics and the description of the data conversion are inputs to the program conversion process. It

uses them to determine the data originally accessed and how to accomplish the same access in the new structure, and it produces a new program to do this. Currently, a combination of manual translation, emulation and bridge programs accomplishes this conversion process and an automatic or semiautomatic program-conversion technology is only in the early stages of research.

Currently, as the common approach to data conversion, one develops customized programs for each transfer of data from one environment to another. This approach is inherently expensive: The programs are developed for use only once and their development costs cannot be amortized. Further, poor reliability results from the greater likelihood of making errors as data is passed from program to program.

For a large data-conversion effort, tracing data back through several passes to the source of an error in the data at a particular point can become unmanageable. As an alternative, one may search for a broader approach to data conversion with a generalized system.

Data exists in many varied and complex forms. Figure 3, an expanded form of the diagram in Figure 1, indicates some of the transformations that need to take place in a data conversion. This diagram illustrates the capabilities needed by the read-and-write process in a data-conversion system.

Unloading of data from its source permits reducing the complex physical structure of the source data base to a very simple physical structure, a sequential data stream. The source data base contains not only the information that interests the user, but also a large amount of control information specific to a particular system.

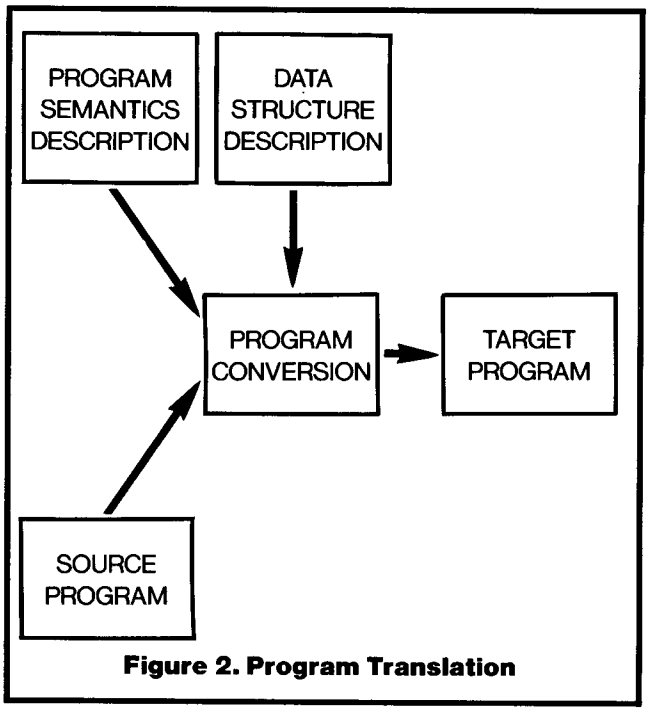

Figure 2. Program Translation

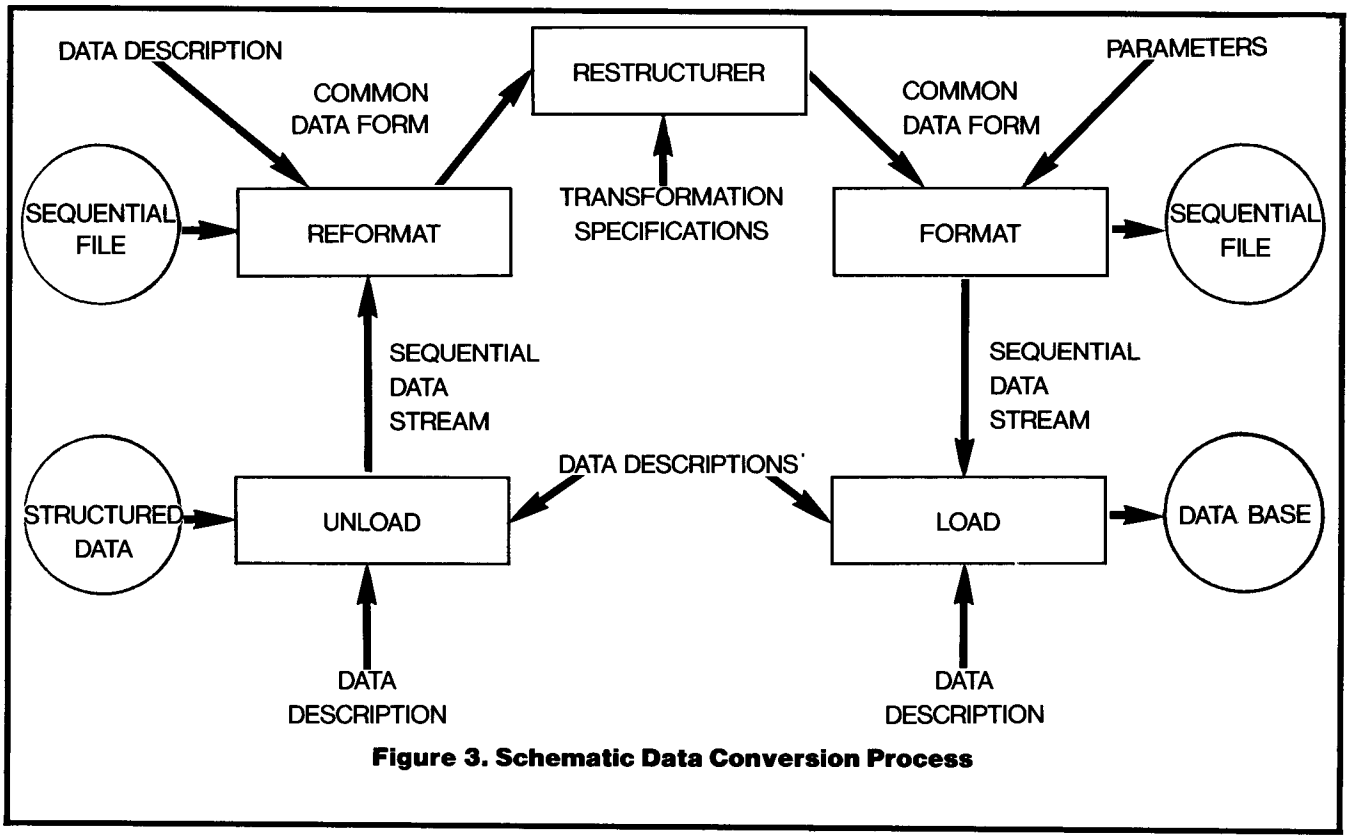

Figure 3. Schematic Data Conversion Process

This control information is used by the system for such data management functions as overflow chaining, index maintenance and record blocking. For example, many systems mark a record to be deleted rather than actually delete it and the unload transformation can remove such system-specific information.

Another factor that causes complexity in the unloading process is the frequent use of pointers in a source system. Pointers are used for two basic purposes: to represent relationships that exist between record instances and to implement alternative access paths to the data. During the unload transformation, the second class of pointers may be discarded without loss of information. The first class of pointers, however, contains information that the transformation must preserve.

The reformat process creates a common data form of the source data for the restructuring step in the conversion process. In the case of a simple sequential file not under DBMS control, it may enter the conversion process at this point. Depending on the design of a system, this step may involve editing (encoding or recoding of items), limited data extraction, correction and the like . . .

Since this step seeks to create a data structure of the source data without system-dependent information, one can consider the mapping between the input and the output of the reformat process to be generally one-to-one. While this step looks simple functionally, its actual application and implementation can be quite complex. For example, an application program may use the high-order bits of a zoned decimal number for its own purposes, knowing that these bits are not used by the system. Such specifications of nonstandard item encodings present a difficult problem in data conversion.

The load process is the counterpart of the unload process and needs no further clarification. Note, however, that the use of a common data form provides additional benefits, such as easing the portability problem.

The restructuring process undoubtedly represents the most complex process of a generalized data-conversion system. The languages for this mapping process can differ widely (for example, some procedural and other nonprocedural) and the models used to represent the data in the conversion system are also quite divergent. (For example, some use network structures; others use hierarchical structures.)

Generally, there are two implementation techniques: an interpretive or generative approach. In the first approach, the action of the system will be driven by the descriptions written in the system's languages via the general interpreter implemented for the particular system. In the generative approach, the data and mapping descriptions are fed into the compiler(s) which generates a set of customized programs executable on a certain machine.

Currently, available tools have limited capabilities. Because it is impossible in this short report to provide an exhaustive survey of all the vendor-developed conversion tools, we will highlight the spectrum of capabilities available to the user by providing examples from specific vendor shops.

The repertoire of vendor conversion tools begins at the character-encoding level of data conversion with the provision of hardware/firmware options and continues through the software aids for conversion and restructuring of data bases.

Depending on a diversity of conditions, the need to develop software tools varies from vendor to vendor. Probably the most prevalent file-conversion tool is a COBOL foreign file-processing aid. This type of facility allows the direct reading or writing of a particular class of files such as EBCDIC tapes or Honeywell Series 200/2000 files within a COBOL.

Although a relatively widespread facility, its capabilities are nevertheless limited. For example, some do not handle unlabeled tapes, while others cannot process mixed-mode data types. Aside from the work of Behymer and Bakkom [DT11], which was aimed toward achieving a general conversion bridge with a particular vendor, to our knowledge there are no vendor supported *generalized* file-translation tools.

In contrast, tools have been developed that have their main applications in a data base environment. One example of a data base conversion aid is the I-D-S/II migration aid provided by Honeywell. Because of the large volumes of data involved and the fact that the user cannot afford to shut down the whole data-processing shop, a coexistence approach was adopted.

The first step is to reformat the I-D-S/I data base into the I-D-S/II format, making the necessary data-type conversions and pointer-mechanism adaptions. This step allows the data base to be processed in the I-D-S/II mode, but not optimally. Additional steps in this migration include the generation of the additional I-D-S/II pointer fields (I-D-S/II requires "prior & header" chain pointers, which are allocated in Step 1 but not filled in) and the restructuring of the I-D-S/II (coexistence) to the more sophisticated capabilities of I-D-S/II.

Some data-base restructuring tools specific to a particular DBMS have been developed by DBMS users. One example of this type of tool is REORG [R11], a system developed at Bell Laboratories for reorganization of UNIVAC DMS-1100 data bases. REORG provides capabilities for logical and physical reorganization of a data base using a set of commands independent of DMS-1100 data-management commands. A similar capability has been devel-

oped at the Allstate Insurance Company.

In addition to the above, there are also software companies and vendors which will do a customized conversion task on a contractual basis.

Over the past seven years, a great deal of research on the conversion problem has been performed. The results are summarized in Figure 4. The University of Michigan, the University of Pennsylvania, IBM, SDC and Bell Laboratories initiated projects, as well as a task group of the CODASYL Systems Committee. In many cases, interaction and cross-fertilization among these groups led to some consensus on appropriate architectures for data conversion.

In 1970, the CODASYL Systems Committee formed a task group (originally called the Stored Structure Description Language Task Group) to study the problem of data translation. The group presented its initial investigation of the area in the 1970 SIGMOD (then SIGFIDET) annual workshop in Houston [SL1]. In 1972, the group was reformulated as the Stored-Data Description and Translation Task Group and presented a general approach to the development of a detailed model for describing data at all levels of implementation [SL4,DT2]. The most recent work of the group specifies the data-conversion model and presents an example language for describing and translating a wide class of logical and physical structures [SL8]. The stored-data definition language allows data to be described at and distributed to the access path, encoding and device levels.

The nonprocedural approach to stored-data definition set forth by Taylor and Sibley [SL3,6] provided one of the major foundations for the development at the University of Michigan (see Figure 4) of data translators. In concert with Taylor's language, Fry and others [DT1] initiated a model and design for a generalized translation.

The translation model was tested in a prototype implementation of the Michigan Data Translator in 1972 [UT2,4], and the results of the next implementation, Version I, were reported by Merten and Fry [DT4].

In 1974, the work of the Data Translation Project of the University of Michigan focused on the data base restructuring problem. Navathe and Fry investigated the hierarchical restructuring problem by developing several levels of abstractions, ranging from basic restructuring types to low-level operations [R6].

Later, Navathe proposed a methodology to accomplish these operations using a relational normal form for the internal representation [DT12]. Version II of the Michigan Data Translator was designed to perform hierarchical restructuring transformations, but the project did not implement it. Instead, the research was directed into the complex problem of restructuring network type data bases. To address this problem, Deppe developed a dynamic data model—the Relational Interface model—which simultaneously allowed a relational and network view of the data base [UR3].

This model formed the basis of the Version IIA design and implementation of generalized restructuring capabilities [UT8,9,10]. Another component necessary for the development of a restructurer was the formulation of a language in which to express the source-to-target data transformations. This language, termed Translation Definition Language (TDL), evolved through each translator version beginning with a source-to-target data item "equate list" in the Version I Translator to the network-restructuring specifications of Version IIA. While the initial version of the TDL was quite simplistic, the current version, the Access Path Specification Language [DT16,R9], provides powerful capabilities for transforming network data bases.

Concurrent with the work at the University of Michigan, Smith at the University of Pennsylvania (see Figure 4) also took a data-description approach and developed a stored-data definition language (SDDL) for defining storage of data on secondary storage devices and a translation description language (TDL) [SL2,DT2]. Three levels of data base structures, the logical, storage and physical, are described, using the SDDL. To describe the source-to-target data mappings, a first-order calculus language was used.

Following from this work, Ramirez [DT3,6] implemented a language-driven "generative" translator which created PL/1 programs to perform the conversion. One of the first reports on the utilization of generalized translation tools was provided by Winters and Dickey [TAI]. Using the translator developed by Ramirez, they installed it on their system, and applied it to converting IBM 7080 files.

In 1973, another major data translation research endeavor was initiated at the IBM Research Laboratory in San Jose, California. Researchers in this project—Housel, Lum and Shu, later joined by Ghosh and Taylor—adopted the general model as specified in Figure 1, but made several innovations.

First, in the belief that programmers know well the structure of the data in a buffer being passed from a DBMS to the application program, the group concentrated its effort on designing a data-description language appropriate for describing data at this stage. Second, regardless of the data model underlying any DBMS, the data structure at the time it appears in the buffer of an application program will be hierarchical. The general architecture, methodology and languages reflecting these beliefs is reported in Lum and others [DT14].

In addition, the San Jose group felt that while it is desirable to have a file with homogeneous record types, it is a fact of life that many of today's data are still in COBOL files in which multiple-record types frequently exist within

	CODASYL STORAGE STRUCTURE DEFINITION LANGUAGE TASK GROUP:	UNIVERSITY OF MICHIGAN:	UNIVERSITY OF PENNSYLVANIA:	IBM RESEARCH, SAN JOSE:	SYSTEM DEVELOPMENT CORPORATION:
1970	Specifications of the data conversion problem and possible approaches to solving it [SL]				
1971		Model of logical structure structure to physical structure mappings of DBMS [SL3,6]	Model and languages for Stored-Data description and translation description [SL2] (Moore School)		
1972	Stored-Data Description and Translation Task Group: Model of generalized data conversion architecture [SL4,DT2]	Data Translation Model [DT1]			
1973		Restricted prototype data translator (sequential files) [UT3,4, DT4]	Restricted prototype data translator [DT3,6] (Moore School)	Restricted prototype data translator [DT3,6] (Moore School)	Model and Language for data translation (AIM [DT5] Define [SL7])
1974		Model of restructuring [R6]			Data translation prototype [R3,4]
1975			External test of Penn Translator [TA1] (Moore School) Scope of project broadened to generation of business systems	Convert [R2]	
1976	Detailed model of stored data [SL8] Group redirected attention to program conversion	Michigan Data Translator Operational Prototype (network restructuring and implementation of Translation Description Language [DT13, 16]	Dynamic restructuring [R8] (Wharton School)	Implementation of XPRS [DT15]	
1977		Access Path Specification Language [R2]		Laboratory of external tests of XPRS	ADEPT Current development data translation [DT17]

Figure 4. Historic Context of Data Conversion Efforts

the same file. As a result, the group concentrated on designing a data-description language which can describe not only hierarchical records (in which a relational structure is a special case) but also most of the commonly used sequential file structures. This language, DEFINE, is described by Housel and others [SL7].

The philosophy of restructuring hierarchies is further reflected in the development of the translation-definition language CONVERT, as reported by Shu and others [R2]. This language, algebraic in structure, consists of a dozen operators, each of which restructures hierarchical files into another file. The language possesses the capability of selecting records and record components, combining data from different files, built-in functions such SUM and COUNT and the ability to create fields and vary selection of the basis of a record's content (a CASE statement).

A symmetric process occurs at the output end of the translation system. Sequential files are created to match the need of the target-loading facility. The specification of this structure is again made in DEFINE.

A prototype implementation, originally called EXPRESS but renamed XPRS, is reported in [DT15].

Another restructuring project reported by Shoshani [R3,4] was performed at the System Development Corporation in 1974–1975. To avoid the complexities of storage structure specification (indexes, pointer chains, inverted tables and the like), the company chose to use existing facilities of the systems involved. In particular, it advocated the use of query and load (generate) facilities of data base management systems. However, when such facilities do not exist, reformatters from the source (such as index sequential file) to a standard form and from the standard form to same output file had to be used. Given that data bases can be reformatted to and from a standard form, the company concentrated on the problem of logical restructuring of hierarchical data bases in this form.

The language used in the project for specifying the restructuring functions (called CDTL for Common Data Translation Language) was designed to be conceptually simple. For the most part, it provides functions for for specifying a mapping from a single field (or combination of fields) of the source to a single field of the target.

For example, while a DIRECT would specify a one-to-one mapping of source items to target items, a REPEAT would specify the repetition of a source item for all instances of a lower level in the target hierarchy. In both cases, only the source and target fields need to be mentioned as parameters. In addition, there are more global operations, such as the INVERSION operator, which causes parent/dependent record relationships to be reversed.

The system also supported extensive field restructuring operators, where individual field values could be manipulated according to prescribed language specifications. Since most of these operators are local, there is the possibility that they could be used in combinations that do not make sense globally. Therefore, a further component of the system was built to perform "semantic analysis," which checks for possible inconsistencies before proceeding to generate the target data base.

The Bell Laboratories data translation system ADAPT (A Data Parsing and Transformation system), currently under development, is generalized translation system driven by two high-level languages [DT17]. With the first language, one describes the physical and logical format and structure of the data and uses it for various tests and computations while parsing the source data and generating the target data. The second language is used to describe the transformations which are to be applied to the source data to produce the target data. Extensive validation criteria can be specified to apply to the source and target data.

Two processing paths are available within the ADAPT system: a file translation path and a data base translation path (see Figure 3). A separate path for file translation responds to real-world considerations: Many types of conversions do not require the capabilities and associated high overhead involved in using a data base translation path.

Additional research effort examines the development and acceptance of a standard interchange form. An interchange form would increase the sharing of data bases and provide a basis for development of generalized data translators. The Energy Research and Development Administration (ERDA) has been supporting the Interlaboratory Working Group for Data Exchange (IWGDE) in an effort to develop a proposed data interchange form. The proposed interchange form [GG2] has been used by several ERDA laboratories for transporting data between the laboratories. Additional work on development of interchange forms has been pursued by the Data Base Systems Research Group at the University of Michigan [UT14].

Navathe [R10] has recently reported a technique for analyzing the logical and physical structure of data bases with a view to facilitating the restructuring specification. Data relationships are divided into identifying and nonidentifying types to draw an explicit schema diagram. The physical implementation of the relationships in the schema diagram is represented by means of a schema-realization diagram. These diagrammatic representations of the source and target data bases could prove to be very useful to a restructuring user.

So far, we have concentrated on the data aspects of the conversion problem. It is necessary to deal as well with the problems of converting the application programs which operate on the data bases. Program conversion, in general, may be motivated by many different circumstances, such as hardware migration, new processing requirements or a decision to adopt a new programming language.

Considerable effort has been devoted to special tools such as those to assist migration among different vendor's COBOL compilers, and general-purpose "decompilers" that have been developed to translate assembly language programs to equivalent software in a high-level language. While progress has been made in developing special-purpose tools for a limited program-conversion situation, little progress has been made in obtaining a solution to the general problem of program conversion.

There are three factors which can affect application programs:

- Alterations to the data base physical structure, for example, the format and encoding of data or the arrangement of items within records.
- Changes to the data base logical structure, either the deletion or addition of access paths to accommodate new performance requirements or changes to the semantics of data, for example, modification of defined relationships between record types or the addition or deletion of items within records.
- Migration to a new DBMS, perhaps encompassing a data model and/or data-manipulation language different from the one currently in use.

The actual impact of these data base changes on application programs is a function of the amount of data independence provided by the DBMS. Data independence and its relationship to the conversion problem are discussed elsewhere [GG3]. We assume here incomplete data independence and that therefore some degree of program conversion is required in response to data base schema changes. In fact, whereas most commercial DBMS provide application programs with insulation from a variety of modifications to the physical data base, protection from logical changes—particularly at the semantic level—is minimal. Examples of semantic changes likely to have a profound effect on application programs include the following:

- Changes in relationships between record types, such as changing a one-to-many association to a many-to-many association or vice-versa.
- Deletion or addition of data items, record types or record relationships.
- Changing derivable information ("virtual items") to explicit information ("actual items") or vice-versa.
- Changes in integrity, authorization or deletion rules.

Various properties of data base application programs greatly complicate the conversion problem. For instance, many DBMS do not require that the record types of interest (or possibly even the data base of interest) be declared at compile time in the program; rather, these names can be supplied at run time. Consequently, at the compile time, incomplete information exists about what data the program acts on. Other troublesome problems occur when programs implicitly use characteristics of the data which have not been explicitly declared (for example, a COBOL program executes a paragraph exactly 10 times because the programmer knows that a certain repeating group only occurs 10 times in each record instance).

Complexity is introduced whenever a data manipulation language is intricately embedded in a host language such as COBOL. The interdependence between the semantics of the data base accesses and the surrounding software greatly complicates the program analysis stage of conversion. Because of these considerations, substantial research has been devoted to alternatives to the literal translation of programs. In particular, some currently operational tools utilize source program emulation or source data emulation at run time to handle the problem of incomplete specification of semantics and yet still yield the effects of program conversion.

The DML substitution conversion technique, which can be considered an emulation approach, preserves the semantics of the original code by intercepting individual DML statement calls at execution time and substituting new DML statement calls which are correct for the new logical structure of the data base. Two IBM software examples which provide this type of conversion methodology are the ISAM compatibility interface within VSAM (this allows programs using ISAM calls to operate on VSAM data base) and the BOMP/DBOMP emulation interface to IMS.

This program conversion approach becomes extremely complicated when the program operates on a complex data base structure. Such a situation may require the conversion software to evaluate each DML operation against the source structure to determine status values (such as currency) to perform the equivalent DML operation on the new data base.

Generalization of this approach requires the development of emulation code for the following cases: maintain the run time descriptions and tables for both the original and new data base organizations, intercept all original DML calls and utilize old-new data base access path mapping description (human input) and rules to determine dynamically what set of DML operations on the new data base are equivalent to each specific operation on the source data base.

Although a conceptually straightforward approach, it has several drawbacks. The drawbacks can be categorized as degraded efficiency and restrictiveness. Efficiency is degraded primarily because each source DML statement must be mapped into a target emulation program, which uses the new DBMS to achieve the same results. The increased overhead in program size and/or processing requirements can be significant.

The drawback of restrictiveness comes about because

the emulation approach inhibits the utilization of the increased capabilities of the new DBMS and/or data structure through the modeling of the old methods. Additionally dependence upon the old program semantics limits the sets of permissible new data structures that must support all of the semantics of the source program if the source program is to continue to execute in the same manner.

Note that the rules can be quite complex, even for the limited situation of which the data structure changes preserve semantic equivalence. Therefore, in some instances, just the limited task of determining if a change in data structure (given no change in the data model) will support a set of source programs will be an extensive task.

The second method in use today is sometimes referred to as the bridge program method. In this technique, the source application program's access requirements are supported by reconstructing from the target data base that portion of the source data base needed. Data reconstruction is done by means of "bridge programs." The source program is then allowed to operate upon this reconstructed portion of the source data base to effect the same results that would occur if the source data base were not modified. Of course, a reverse mapping is required to reflect and update, and each simulated source data base segment must be prepared before it is needed by the application program.

This approach suffers from the same types of disadvantages inherent in the emulation approach. Efficiency problems for complex/extensive data bases and programs performing extensive data accessing can make this method prohibitively expensive for practical utilization. This technique is generally found as a "specific software package" developed at a computer installation rather than as a standard vendor-supplied package.

Differing from the emulation and bridge program approaches, current research aims toward developing more generalized tools to automatically or semiautomatically modify or rewrite application programs. The drawbacks of the existing approaches described above can be avoided by rewriting the application programs which would take advantage of the new structure and semantics of a converted data base and by using a general system to do the conversion rather than using ad hoc emulation packages and bridge programs.

Research on application program conversion is still in its infancy. Consequently, very few published papers on this subject exist.

Mehl and Wang [PT6] presented a method to intercept and interpret DL/1 statements to account for some order transformations of hierarchical structures in the context of the IMS system. Algorithms involving command substitution rules for various structural changes have been derived to allow the correct execution of the old application programs. This approach works only for a limited number of order transformation of segments in a logical IMS data base. Since it is basically an emulation approach, it has the drawbacks discussed earlier.

A paper by Su [PT12] gives a general model of application program conversion as related to data base changes resulting from a data base transformation. An attempt was made to identify the tasks required for the automatic or semiautomatic conversion of application programs due to data base changes. The paper stresses two main points: 1. the need for extensive analysis of an application program including the analysis of program logic, data variable relations, program-subprogram structure, execution profile and so on and 2. the use of data base translation operators to determine what program transformations are required to account for the effects of these operators. The idea of the use of a common language to describe the operations of source queries and the data translation statements is also proposed.

An approach to the transformation of DBTG-like programs in response to a data base restructuring was proposed by Schindler [PT10]. The approach is based on the concept of code templates, which are predefined sequences of host language—DML statements (roughly analogous to assembly language macros). Application programs can be written as nested code templates.

The code templates are devised so that each one corresponds to an operator in the relational algebra. An application program is then mapped into a relational expression, transformations are performed on the expression to accommodate the data base restructuring and a new program is generated by mapping the transformed expression back into code templates. This approach suggests that a level of logical data independence may be achieved through current programming technology.

The work by Su and Reynolds [PT15] studied the problem of high-level sublanguage query conversion using the relational model with SEQUEL [Z5] as the sublanguage, DEFINE [SL7] as the data description language and CONVERT [R2] as the translation language. Algorithms for rewriting the source query were derived and hand-simulated. In this study, query transformation is dictated by the data translation operators which have been applied to the source data base.

The purpose of this work was to study the effects of the CONVERT operators on high-level queries. Only restricted types of SEQUEL queries were considered. This work demonstrates that a general program conversion system should separate the data model and schema-dependent factors from the data model and schema-dependent factors from the data model and schema-independent factors; an abstract representation of program semantics and the semantics of data translation operators need to be sought so

that data conversions at the logic level (especially the type which changes the data base semantics) and the DBMS level can be attempted.

Two independent works carried out about the same time by Su and Liu [PT13] and Housel [PT14] take a more general approach to the application program conversion problem. The former work is based on the idea that the same data semantics (a conceptual model) can be modeled externally by various existing data models (relational, hierarchical and network) using different schemas. Application programs are mapped into an abstract representation which represents program semantics in terms of the primitive operations (called access patterns) that can be performed on data entities and associations.

Transformation rules are then applied on the abstract representation based on the types of changes introduced by the data translation operators. The transformed representation is then mapped into another intermediate representation (called access path graphs) which is dictated by the external model and specific schema used for the target data base. This representation is then modified by an optimization component and used for the generation of target programs.

This work stresses that the semantics of both the source and target data base be made explicit to the conversion system and be used as a basis for application program analysis and transformation. The conversion methodology described is for program conversion to account for data conversion at the logical level as well as the DBMS level.

Housel extends the work on application migration undertaken at the IBM San Jose laboratory. This work uses a common language for specifying the abstract representation of source programs as well as for specifying the data translation operations. The language is a subset of CONVERT with some of Codd's relational operators [GG4h.

The operators of the language are designed to have a simple semantics and convenient algebraic properties to facilitate program transformation. They are designed to handle data manipulation in a general hierarchical structure called a "form" as well as relational tables. In this system, program transformation is dictated by the data mapping operations applied to the source data base. The proposed model assumes that the inverse of these data mapping operators exists; that is, the source data base can be reconstructed from the target data base by applying some inverse operators on the target data base.

More precisely, it is assumed that $M^1(T) = S$ where S is the source data base, T is the target data base and M is the mapping function. Thus, program conversion is done by substituting the inverse $M^1(T)$ into the specification language statements (the abstract representation of the source program) for each reference to the source data base. This process is followed by a simplification procedure to simplify the resulting statements (the target abstract representation of the program). The author points out that the assumption on the existence of $M^1(T)$ restricts the scope of the conversion problem handled by the proposed approach.

Presently, the Data Base Program Conversion Task Group (DPCTG) of the CODASYL Systems Committee is investigating the application group conversion problem. The group is looking into various aspects of the problems including decompilation of COBOL application programs, semantic changes of data bases and their effects on application programs, program conversion techniques and methodologies and so on.

To this date, the work on application program conversion is still very much at the research stage and more progress has to be made before we can start actual implementation. The problems of automatic application program conversion are multitudinous and extremely complex. Current research indicates that program conversion is possible for some types of data conversion, but the complexity of program conversion depends on how drastically the data has been modified.

Further research needs to be undertaken to determine what can be done automatically, what can be done semiautomatically and what cannot be done at all. A fully automatic tool is hard to achieve. Building semiautomatic systems or systems which provide aids for manual conversion would be a more realistic goal.

The current research has uncovered several problems which need to be investigated further before the implementation of a generalized conversion tool can be attempted. The following issues are believed to be important for further research.

Based on the work by Su and Liu [PT13] and the study of the DPCTG group, it is quite clear that a program conversion system would need more information about the semantic properties of the source and target data bases than the imformation provided by the schemas of the existing DBMS. Semantic information of the data bases is an important source for determining the semantics of application programs, which is the real bottleneck of the application program conversion problem.

Future research needs to be conducted to model and describe the semantics of application programs, study the meaningful semantic changes to data bases and their effect on application programs and derive transformation rules for program conversion which account for the meaningful changes. Several existing works on data base semantics [M11,SM1,2,3,DL29] may provide a good basis for future works on this subject.

Data conversion may alter the semantic contents of the source data base. A converted application program may or may not perform the identical operation on the target data

as the source program on the source data. For example, it may not retrieve, delete or update the same data as the source program because some records may be deleted and data relations may have been changed in data conversion.

It is not clear at all how we can prove, in general, that a target program generated by a conversion system still preserves the original intent of the source program. Naturally, if the source data can be reconstructed from the target data without losing the original data relations and occurrences, we can establish the equivalence relation between the source and target programs based on the same effort they have on the source and target data.

Program conversion via decompilation is a technique whereby a data base application program is first transformed into an operationally equivalent higher-order language or an abstract representation and then returned to a usable language level in a converted form. The transformation to a higher-order language level is a decompilation process, and the process of returning the program to a language level appropriate to conventional compilers is a compilation process.

The underlying concept is that the decompilation to a higher-order language can produce a functionally equivalent program that does not contain the DBMS, data model and data dependencies that inhibit the conversion process. That is, the decompiled program has the same "intent" while being unrelated to the changed DBMS environmental conditions. The changed environmental conditions should be easily incorporated into the program during the process of compiling the program back into a form appropriate to the new system.

Some researchers think that this would be the preferred method to effect DML/host language program conversion. It should avoid many of the efficiency/restriction drawbacks inherent in current automated methods, while being more cost-effective and less error-prone than current manual methods (such as program rewrite).

One likely disadvantage to this method is that to use it to convert existing data base application programs, the programs may first have to be manually altered to place DML-related code in a structured format. This disadvantage is to be expected because of the ambiguity inherent in the organization of DML/host language programs.

However, the development of structured-programming templates designed for DML-related code should provide a means for creating programs that are convertible by the decompilation method. Structured templates might also provide the needed insight toward the development/selection of an appropriate high-level language into which programs can be compiled. Some initial concepts of data base program templates have been proposed by the University of Michigan [PT10].

A system which provides assistance to conversion analysts would seem to be a practical tool and a feasible task. Given the information about data changes and semantics of the data, a system can be built to analyze application programs to 1. identify and isolate program segments which are affected by the data changes, 2. detect inefficient code in the programs, 3. produce a program execution profile [GG1] which gives an estimate of the computation time required at different parts of the program and 4. detect, in some cases, the program code which depends on the programmer's assumption of data values, ordering of records, record or file size and so on.

The data obtained in 1, together with some online editing and debugging aids, would speed up the manual conversion process. The data obtained in 2 and 3 would be useful for producing more efficient target programs, and the data obtained in 4 would help the conversion analyst eliminate the "implicit semantics" in programs which makes the program conversion task (manual or automatic) extremely difficult.

A more complete cross-referencing than that usually produced by today's compilers can assist the conversion analysts in identifying ramifications of changes to programs. An example of such a product is the Data Correlation and Documentation system produced by PSI-TRAN Corporation.

One technique, sometimes used during a conversion process that has been initiated by a data base structure change, is to alter the names of affected data base items in the DDL only and use errors generated by the compiler to locate those program segments needing changes. A more complete cross-referencing system would be a much better tool, if it were available.

As the result of data conversion, multiple-access paths to the same data may occur. This is because redundant data may be introduced or new access paths may be added in the course of data conversion. In this situation, a conversion system will have the choice of selecting a path to generate the target program. The efficiency of the program during execution time may depend on the selection of optimized access path during program conversion.

Also, for reasons of achieving generality, some program conversion techniques proposed [PT13,14] convert small segments of programs or the equivalent of DML statements separately. It is necessary to do a global optimization or simplification to improve the converted program. Techniques for program optimization related to program conversion need to be investigated.

The prototype systems described above have been used in a few conversions. While some of these tests were made on "toy files," a few of the tests involved data volumes from which realistic performance estimates can be extrapolated.

The translator developed by Ramirez at the University of

Pennsylvania [DT3,6] processes single sequential files to produce single sequential target files. Facilities exist for redefining the structure of source file records, reformatting and converting accordingly. Conversion of the file can be done selectively, using user-defined selection criteria. Block size, record size and character code set can be changed, and some useful data manipulation can be included.

The translator was used in several test runs on an IBM/370 Model 165. The DDL to generated PL/1 code expansion ratio was 1:4, so coding time was reduced.

A further test of the Penn Translator was conducted by Winters and Dickey [TA1]. An experiment was conducted comparing a conventional conversion effort against the Penn Translator (slightly modified). Two source files stored on IBM 1301 disks under a system written for the IBM 7080 using the AUTOCODER language were converted to two target files suitable for loading into IMS/VS. Much of the data was modified from one internal coding scheme to another. The conversion required changing multiple source files to multiple target files.

The conventional conversion took seven moths versus five months for the generalized approach, a productivity improvement of roughly 30 percent. Time for adapting the translator, learning the DDL and adapting to a new operating system is included in the five-month figure. Without these, an estimate of three months was made for the conversion using the generalized approach.

The translator described in [R3,4] was implemented during 1975–1976. The translator could handle single, hierarchical files from any of three local systems—TDMS, a hierarchical system which fully inverts files; DS/2, a system which partially inverts files, and ORBIT, a bibliographic system which maintains keys and abstracts. Data bases were converted from TDMS to ORBIT, from TDMS to DS/2, vice-versa and from sequential files to ORBIT. TDMS files were unloaded, using an unload utility. Target data bases were loaded by target system load utilities.

The total effort for design and implementation was about three manyears. The system was implemented in assembly language on an IBM/370 Model 168, and occupied about 40K bytes, not including buffer space which could be varied. The largest file tested was on the order of 5 million characters and the total conversion time was about one minute of CPU time per 2.5 megabytes of data. The work was discontinued in 1976.

The prototype file translator developed at Honeywell by Bakkom and Behymer [DT11] performed file conversions (one file to one file) among files from IBM, Honeywell 6000, Honeywell 2000, Honeywell 8200 sequential and indexed sequential files. Data types of fields could be changed, as well as field justification and alignment. New fields could be added to a record and fields could be permuted within a record.

File record format (fixed, variable, blocked and so on) could be changed, and a compare utility was available for checking and consistency of files with different field organizations and encodings. Tests of up to 10,000 records were run. Performance of 15 milliseconds per record was typical (Honeywell Series 6000 Model 6080 computer). The prototype has been used in a conversion/benchmark environment, but has not been offered commercially.

Version IIB, Release 1.1 of the Michigan Translator was completed for the Defense Communications Agency in October 1977 [UT16]. It offers complete conversion/restructuring facilities for users of Honeywell sequential, ISP or I-D-S/I files. Up to five source data bases of any type may be merged, restructured or otherwise reorganized into as many as five target data bases, all within a single translation.

Data base description is accomplished by minor extensions to existing I-D-S DDL statements. Restructuring specification is easily indicated via a high-level language. Tests performed to date included a conversion of a 150,000 record I-D-S/I data base with a total elapsed time of 24 hours (500 milliseconds per record). A given translation can be broken off at any point to permit efficient utilization of limited resources and also protect against system failures. The user is provided with the capability of monitoring translation progress in real time.

Test cases with the XPRS system have focused on functionally duplicating earlier real conversions done by conventional methods. Several cases have been programmed. Each case involved at least two input files. Generally, there was a requirement to select some instances from one file, match with instances in another file, eliminate some redundant or unwanted data and build up a new hierarchical structure in the output.

In several cases there was a need for conditional actions based on flags within the data. In all cases, the XPRS languages were found to be functionally adequate to replicate the conversion. A productivity gain of at least 50 percent in total analysis, coding and debugging time was achieved. Test runs were conducted on several thousand records. Performance was deemed adequate in that XPRS can restructure data at least as fast as it can be delivered from direct access storage. No detailed performance comparisons were made comparing XPRS-generated programs with custom-written programs.

Given that several prototype data-translation systems are operational in a laboratory environment, there is a little question concerning the technical feasibility of building generalized systems. The remaining questions pertain to the use of a generalized system in real-world data conversions involving a wide variety of data structures, very large data volumes and significant numbers of people.

Three major questions to be resolved are:

1. Are the generalized systems functionally complete enough to be used in real conversations, and if not, what will it take to make them functionally complete?
2. Can the people involved in data conversions use the languages? What additional features are necessary to enhance usability?
3. Overall, what is the productivity gain available with the generalized approach?

Within the next year, prototype systems will be exercised on a variety of real-world problems in data translation, and concrete answers to these questions should be available. The systems being further tested for cost-effectiveness are the Michigan Data Translator, the IBM XPRS system and the Bell Laboratories ADAPT system.

To date, preliminary results have been promising. A significant sample size on which to do analysis of productivity gain should be available at the end of the year of testing.

A number of factors must be taken into account in measuring the cost-effectiveness of the generalized data translator versus the conventional conversion approach. These factors include:

- Ease of learning and using the higher-level languages which drive the generalized translators.
- Availability of functional capability to accomplish real-world data conversion applications within the generalized translators.
- Overall machine efficiency.
- Correctness of results from the conversion.
- Ability to respond in timely fashion to changes in conversion requirements (conversion program "maintenance").
- Debugging costs.
- Ability to provide "bridge-back" of converted data to old applications.
- Ability to provide verification of correctness of data conversion.
- Capabilities for detection and control of data errors.

The languages used to drive generalized data translators are high-level and nonprocedural; they provide a user-friendly interface to the translators. Since the languages are high-level, programs written in them have a better chance of being correct. Experience to date with DEFINE and CONVERT, the languages which drive XPRS, has shown that users can learn these languages within a week; it has also shown that some practice is necessary before users start thinking about their conversion problem in nonprocedural rather than procedural terms.

In early test cases, the languages which drive generalized data translators have been found to be functionally adequate for many common cases. In those cases lacking a feature, a "user hook" facility is often provided. However, forcing a user to revert to a programming language "hook" defeats the purpose of the high-level approach, and interfacing the hook to the system requires at least some knowledge of system interfaces. Thus, high-level languages must cover the vast majority of the cases to succeed; otherwise, users will perceive little difference over conventional approaches.

Facilities for detecting and controlling data errors in the generalized systems are very important, and most of the prototypes do not yet do a complete job in this area. However, the generalized packages offer an opportunity for generalized, high-level methods for dealing with data errors during conversion. It could well be that once these error packages are developed, they will contribute to even larger productivity gains than have been experienced to date.

The high-level language approach to driving generalized translators should provide the ability to respond to changes in conversion requirements with relative ease. Since large conversions often take one or more years, it is not unusual for the target data base design to change or for new requirements to be placed on the conversion system. In other words, in a large conversion effort, the programs are not as "one-shot" as is commonly believed. In large conversions, the savings in conversion program maintenance could be significant.

Generalized systems can also be used to map target data back to the old source data form, assuming the original conversion was information-preserving. This capability provides a means for verifying the correctness of the data conversion. In addition, this capability can be used as a "bridge-back" to allow users to continue to run programs which have not yet been converted against the data in the old format. Using a generalized system in this way allows phased conversion of programs without impacting user needs during the conversion period.

In an environment where a generalized translator is used regularly as a tool for conversion, costs associated with the debugging phase should be decreased. Common, debugged functional capabilities will be utilized, whereas it is unusual in the conventional approach for common conversion modules to be developed. Thus, each new conversion system requires debugging.

The usability of generalized data translation systems must also be evaluated. Experience to date indicates that the languages are easy to learn and use. However, it would be wrong to think that these prototypes are mature software products or that they can be used in all conversions.

One question concerns the level of users of the generalized languages. Current prototypes have been used by application specialists and/or members of a data base support group. The systems have not yet been used by pro-

grammers, and the question remains whether programmers (as opposed to more senior application specialists and analysts) will be able to use the systems productively. There is no negative data on this point; the systems have not been used widely enough.

At present, all the systems require a user to describe explicitly the source data to be accessed by the read step, using a special data-description language. These data-description languages are generally easy to learn and use; they resemble statements in the COBOL data division. However, the writing of the description is a manual process which can be tedious because a person may have to describe a file with hundreds of fields.

Ideally, a data-conversion system should be able to make use of an existing data description, such as those existing in a data dictionary or a system COBOL macro library. As evidenced by the Michigan Data Translator [DT16], it is reasonable to expect that such an interface will be available as data-conversion systems evolve. Note, however, that a data dictionary or COBOL macro library link may not necessarily solve the problem. Data in current systems is not always fully enough defined to be converted. This is especially true with nondata base files. In these files, data definition often is embedded in the record structures of the programs, and a full definition depends on a knowledge of the procedural program logic.

Even with existing data bases, some fields and associations may not be fully defined within the system data base description. Thus, the user can expect a certain amount of manual effort in developing data definitions. If existing documentation is incomplete, this can be a time-consuming task, although it probably must be done regardless of whether a generalized package is used.

Another area where a user may have to expend effort is in the unload step of the data-conversion process. The data-description languages used to drive the read step have a limited ability to deal with data at a level close to the hardware (pointers, storage allocation bit maps and so on).

Generally one assumes that a system utility program can be used to unload source data and remove the more complex internal structures. An alternative is to run the read step on top of an existing access method or DBMS with the accessing software removing the more complex, machine-dependent structures. While acceptable choices in a great many environments, including most COBOL environments, some cases may exist where neither approach will work.

For example, a load/unload utility may not exist, or a file with embedded pointers which was accessed directly by an assembly language program might not be under the control of an access method. For these cases, the user is faced with complexity during the unload step. The complexity associated with accessing the data would appear to be a factor for either the conventional methods or for the generalized approach. However, in cases such as those above, some special purpose software may have to be developed.

Note that some research [SL8] has examined the difficulty of extending data-description languages to deal directly with these more complex cases. That research has concluded that providing the data-description language with capabilities to deal with more complex data structures greatly complicates the implementation and has an adverse affect on usability. Thus, special-purpose unload programs will continue to be required to deal with some files.

Two approaches have been used in the prototypes—a generative approach in the Penn Translator and XPRS and an interpretive approach in the Michigan Data Translator. In the generative approach, a description of the input files, output files and restructuring operations is fed to a program generator. From these descriptions, special-purpose programs are generated to accomplish the described conversion.

In both the Penn Translator and XPRS, PL/1 is the target language for the generator. The generated PL/1 programs are then compiled and run. In the interpretive approach, tables are built from the data and/or restructuring description. These tables are then interpreted to carry out the data conversion.

In data-conversion systems, as in other software, an implementation based on interpretation can be expected to run considerably more slowly than one based on generation and compilation. Initial experience with prototype data translators has shown that there is much repetitive work, strategies for which can be decided at program compilation/generation time. Also, there is a good deal of low-level data handling, such as item type conversions.

Thus, those implementations based largely on an interpretive approach run more slowly, and the ability to vary bindings at run time does not appear to be necessary. Interpretation was chosen in the prototypes for ease of implementation. In the future, it can be expected that a compilation-based approach or a mixture of compilation with interpretation will be the dominant implementation architecture. However, for medium-scale data bases, the machine requirements of the interpretive data-conversion prototypes are not unreasonable and overall productivity gains are still possible.

Performance measurements with conversion systems based on the generative approach indicate that generalized systems can be quite competitive with customized programs. In one case, the program generated by the data-conversion system ran slightly faster than a "customized" program which had been written to do the same job. However, this example could well be the exception, and it would be naive to expect this in general.

The reason generalized packages can be competitive is that they often have internal algorithms which can plan access strategies to minimize I/O transfer and/or multiple passes over the source data. "Customized" conversion programs written in a conventional programming language are often not carefully optimized, since the expectation is that the programs will be discarded when the conversion is done.

A second architectural difference involves the use of an underlying DBMS or not. In both the Penn Translator and XPRS, the generated PL/1 program, then executing, accesses sequential files, performs the restructuring and writes sequential files. On the other hand, the Michigan Data Translator functions as an application program running on a network-structured DBMS. Thus, the interpreter makes calls to the underlying DBMS to retrieve data during restructuring and puts restructured data into the new data base.

The two approaches offer different tradeoffs. For example, the Michigan Data Translator can make use of the existing extraction capabilities of a DBMS and perform partial translations easily. In addition, since it operates directly within the network data model, a user does not have to think of "unloading" data to a file model and then reloading it back; rather, the user describes a network-to-network restructuring much more directly.

On the other hand, when converting nondata base data to a data base, the use of an underlying DBMS as part of a data translator implies a second-order data-conversion problem: The nondata base data must be converted into the DBMS of the data-conversion system, which may or may not be difficult. It can be difficult, for example, when the data model of the data being converted differs significantly from the data model of the DBMS upon which the conversion system is based. Also, the use of an underlying DBMS may also require more online storage, whereas the file-oriented conversion systems can be made to run tape-to-tape. This can be important in very large data base conversions.

In the future, one can expect that data-conversion systems will offer a variety of interfaces to accommodate various kinds of conversion situations. For example, it is possible to interface the "file-oriented" conversion systems to run as application programs on top of existing DBMS. It is also possible to develop "reader programs" to load nondata base data into conversion systems based on a DBMS. In addition, more automated interfaces to data dictionary packages can be expected to improve usability and obviate the need for multiple data definitions.

One possible performance problem with generalized conversion systems lies in the unload phase. For reasons of usability, generalized conversion systems usually rely on an unload utility program to access the source data, thus isolating the conversion package from highly system-specific data. A potential problem with this approach is that the unload package may not make good use of existing access paths or may tend to access the source data in a fashion which assumes that the data has recently been reorganized (with respect to overflow areas and so on).

In cases where the data is badly disorganized, a customized unload program which accessed the data at a lower level might run considerably faster; for very large data bases, this might be the only feasible way to unload the data. It is not clear how common this case is, and one can usually make the argument that the "special" unload software could be interfaced to the generalized package. However, from a practical standpoint, the unloading phase on a very large, badly disorganized data base is a performance unknown. More sophisticated unload utilities may have to be developed as part of the generalized packages.

Detailed performance and productivity figures for major conversions should be available in about one year. Expectations are that machine efficiency of the generalized packages based on a generation/compilation approach will be acceptable (no worse than a factor of two) when compared with conventional conversion programs.

Additional enhancements to improve usability can be expected, especially in the areas of data error detection and control and interfaces to data dictionary software. If the savings in conversion program analysis and coding times—often 50 percent or more—are confirmed, then the generalized conversion systems will be ready for extensive use:

To identify guidelines for both reducing the need for conversion and for simplifying conversions which are required, one must consider the entire application software-development cycle. Poor application design, poor logical data base design, inadequate use of and inapprorpiate DBMS selection could each lead to an environment which may prematurely require an application upgrade or redesign. This redesign could, in many cases, require a major data base conversion effort.

The set of guidelines specified below is not intended as a panacea. Instead, it is meant to make designers aware of strategies which make intelligent use of current technology. It is doubtful that all conversions could be avoided if a project adhered strictly to these proposed guidelines. However, adherence to the principles set forth by these guidelines could certainly reduce the probability of conversion and, more importantly, simplify the conversions that are required.

With respect to application design and implementation, the more the application is shielded from system software and hardware implementation details, the easier it becomes for a conversion to take place. For example, a good sequential access method hides the difference between tapes, disks and drums from the application programs which use the access method.

The logical data base design should be specified with a clear understanding of the information environment. A good logical data base design reduces the need to restructure because it actually models the environment it is meant to serve. Introduction of data dependencies in the data structure should, if possible, be kept to a minimum. An analysis of the tradeoffs between system performance and likelihood of conversion should definitely be made.

Selecting the wrong or nonoptimal DBMS, given the application requirements, is also a key problem which can lead to unnecessary and large conversion efforts. The prospective user of a DBMS should, for example, carefully evaluate the data-independence characteristics of a proposed DBMS.

The underlying principle of the guidelines which follow is that decisions can be made at the system design and implementation stages which are crucial to the stability of the applications.

Many of the decisions made during the requirements-analysis stage of system development affect the long-term effectiveness of the application system (data base design as well as application programs). Questions such as what functions does the application require, who will the data base serve and how will they use the data base, what are the possible future uses of the data and what are the performance constraits of the application are answered at this stage. It is essential that the designer understand the information environment as much as possible at the outset to lessen the probability that frequent conversions will be necessary.

Requirements analysis should focus on information needs and should minimize constraints being imposed by the physical environment, since it can distort the designer's view of the application system's true objectives. The influence of the physical environment should be considered secondarily so the designer can be fully aware of the resulting compromises to the logical requirements.

This is not intended to imply that consideration of the physical environment is unimportant. Indeed, if the physical environment is ignored, the effect could be development of a set of requirements that are impossible to meet within existing physical and cost constraints.

Three underlying principles motivate this discussion of application program design. They are design for maintainability, design for the application and data independence. Keeping sight of all of these during the design of the application program will lessen conversion effects by rendering the application as free as possible from physical considerations.

Designing for maintainability implies that the application should be written in a high-level language with a syntax that permits good program structure. Structured programming techniques such as top-down program design and implementation should be used throughout. The system should be modular with relatively small, functionally oriented programs.

The programs should all be well-commented and organized for readability. Design reviews and program walkthrough also help to expose errors in the overall design and "holes" in the application logic at an early stage. It has been well-documented that these steps help ease making program modifications.

One error which is often made in designing programs in a DBMS environment is to let the capabilities of the DBMS drive the design rather than the application. This design error can yield programs which are unnecessarily dependent upon the features of a specific DBMS. For example, in System 2000 one can use a tree to represent a many-to-many relationship instead of using the LINK feature.

The parent/child dichotomy that results is an efficient but arbitrary contrivance that cannot easily be undone later on. The key principle here is to concentrate on what results are desired rather than on the implementation details of achieving these results. Simplicity and generalization of the design will provide a very high level of interface to the application programmer, which will, in turn, minimize the total amount of software and provide the greatest degree of portability, maintainability, device independence and data independence.

Of extreme importance in program design is the notion of data independence insulating the application program from the way the data is physically stored.

In the area of application design, the motivating factor for mitigating the effects of a conversion is to insulate logical operations from physical operations. One of the concepts applied to achieve this is layered design. That is, designing the application as a series of layers, each of which communicates with the system at a different level of abstraction. One can visualize this as an "onion," with hardware as its core and layers of successively more sophisticated software at the outer layers. The user interacts with the outermost skin of the onion, at the highest level of abstraction.

If application programs are written at the outermost layers of the onion, then these programs are smaller, easier to understand and, therefore, easier to modify or convert than programs written at lower layers. For example, introduction of a new mainframe will require conversion of the software which references the particulars of the mainframe.

However, since the layers are constructed so that physical machine and device independence is realized above some level, only the software below that level is subject to modification. To the extent that application programs stay at the outermost layers (above the critical layer), reduced conversion effects can be achieved.

We can thus summarize the goal of program design as follows:

- To provide the highest possible application interface to the program.
- To maximize program independence from the characteristics of the mainframe, peripherals and data base organization.
- To maximize portability of the application program through the use of high-level languages.
- To maintain a clean program/data interface.

Poor implementation decisions can go a long way toward diminishing the returns of a good design. Equally important to intelligent design is a set of programming techniques and standards which prohibit programmers from introducing dependencies in code. For example, a "clever" programmer may introduce a word size dependency in a program by using right and left shifts to effect multiplication and division.

Of course, there are no hard and fast safeguards against using tricky coding techniques; an effort must be made to make the programmer conscious of the consequences of this kind of coding. In particular, a programmer should not be allowed to jump across layers of the onion, such as using an access method to read or write directly data bases.

Perhaps the most costly mistake a designer can make is an error in the data base design because it has a direct effect on the information that is derivable and the application programs that are created. Incorrect or unanticipated requirements can lead either to information-deficient data bases or overly complex and general design. An inadequate logical design has the potential for complex user interfaces or extremely long access time. A poor physical design can lead to high maintenance and performance costs.

Unfortunately, data base design is still an art at the present time. Two surveys report the results in the area to date. Novak and Fry [DL26] survey the current logical data base design methodology, and Chen and Yao [DL34] review data base design in general. The work of Bubenko [DL31] in the development of the CADIS system and the abstraction and generalization techniques of Smith and Smith [DL29,30] show promise.

An accurate logical design can still be unnecessarily data-dependent. Dependencies are inadvertently or deliberately introduced in the interest of improving system performance. In essence, "purity" is compromised to gain processing efficiences. Since optimization is a worthwhile goal, insisting on absolute purity may be unreasonable. However, the data base designer should at least be aware of contrivances and, therefore, be in a position to evaluate the relative effects a design decision may have.

Designers should become sensitive to their decisions by asking: "How will the data model be affected by a future change in performance requirements? Have I done a reasonable job in insulating applications from data structure elements that are motivated strictly by performance considerations?"

Some examples of induced data dependencies in logical data base design which may impact upon conversion are:

- The use of owner-coupled sets in DBTG to implement performance-oriented index structures or orderings on records.
- Storing physical pointers (or data base keys) in an information field of a record.
- Combining segment types (in DL/1) to reduce the amount of I/O required to traverse a data base.

Selection of a DBMS can have a major impact on conversion requirements. It is important in evaluating a DBMS to consider products exhibiting the highest level user interface. A high-level DBMS is characterized by both a powerful set of functions and a high degree of data independence from the point of view of the application.

With respect to functions, that is, the DML, the distinction between "high-level" and "low-level" has traditionally centered on whether the DBMS provides user operations on sets of records (select, retrieve, update or summarize *all* the records or tuples which satisfy some conditions) or whether one is restricted to record-at-a-time processing ("navigation"). The DBMS with the "high-level" set operation approach is significantly more desirable than the navigational record-by-record approach.

DBMS prospects should evaluate the data-independence characteristics of a proposed product. Systems are preferred which support an "external schema" or "subschema" feature which permits the record image in the application program (the user work area) to differ significantly from the data base format.

However, the subschema concept is only one aspect of data independence. In general, it is necessary to determine in what ways and to what extent the application interface is insulated from performance or internal format options. For instance, will programs have to be modified if a decision is made to add or delete an index, if the amount of space allocated to an item is increased or decreased or if chains are replaced by pointer arrays?

Other conversion related questions about DBMS products include the following:

- Are there adequate performance and formatting alternatives? Are there too many (unproductive or incomprehensible) turning options? Are there adequate performance measurement techniques and tools to guide the exercise of these choices?
- Does the system automatically convert a populated data base when a new format option is selected?
- Aside from tuning, does the DBMS gracefully accom-

modate at least simple external changes, such as adding or deleting a record or item type?

- Are there other useful high-level facilities associated with or based on the DBMS, such as a report writer, query processor, data dictionary, transaction monitor, accounting system, payroll system and so on?
- Is there a utility for translating the data base into an "interchange form," such as a machine-independent, serial stream of characters?
- Is the vendor committed to maintaining the product across new operating system and hardware releases/upgrades? Conversely, is the vendor prepared to support the product in order released of the operating system, so the user will not be forced to upgrade?
- What hardware environments are currently supported and what is the vendor's policy regarding conversion to another manufacturer's mainframe?
- What programming language interfaces are available? Can the same DBMS features be used if there is a migration, say, from COBOL to PL/1?
- How intelligent is the system's technique for organizing data on the media? Specifically, will performance deteriorate at an inordinate rate as updating proceeds? How often will reorganization (cleanup) be required? Does the DBMS have a builtin reorganization utility? How does the user determine the optimal time to reorganize?
- Are the language facilities and data-modeling facilities of DBMS adequate for the anticipated long-term requirements of the enterprise? What is the risk of having to convert to a new DBMS?
- Likewise, are the performance characteristics and internal storage structural limitations adequate to meet the long-term requirements (response times, data base sizes) of the enterprise?
- Are there facilities to assist the user in converting data from a non-DBMS environment or from another DBMS? For instance, can a data base be loaded from one or more user-defined files?

The cost and performance of processor logic and memory continue to improve at a fast rate. As a result, overhead costs are more acceptable, especially when such costs save people's time and work and provide user-oriented functions that do not require a computer expert.

In particular, one can now think about using generalized conversion tools not only when it is required as a result of hardware or software changes, but also as a result of a changing application that requires a new, more efficient data base organization. What could have been a prohibitive cost for a data base conversion in the past may not be a major factor in the future.

At the same time, the cost/performance improvement contributes to the proliferation of data bases and therefore accentuates the need of generalized conversion tools. The more cost effective is the process of accessing and maintaining data, the more data is collected on computers. Improvements in hardware (as well as software) technologies create more need for data and program conversion. In addition, the emergence of new technologies, such as communication networks, add another level of sophistication to the way that data can be organized and used.

Distributed data bases, where multiple data bases (or subsets of data bases) may reside on different machines, require tools for the integration and the correlation of data. Invariably, data will need to move from system to system dynamically, possibly moving between different hardware/software systems. In this environment, generalized tools for dynamic conversion will become a necessity.

In recent years, two promising approaches to data management hardware technologies have been pursued. One is the specialized data mangement machine and the other is the back-end data management machine. Both approaches can help simpliy the conversion problem.

The specialized data maangement hardware is based on the idea of using some kind of an associative memory device, a device that can perform a parallel access to the data based on its content. Such a device eliminates the necessity for organizing the internal structure of a data base using indexes, hash tables, pointer structures and so on, which are primarily used for fast access.

As a result, the data can be essentially stored in its external logical form, and the data management system can use a high-level language based on the logical data structure only. The conversion process is simplified since data is readily available in its logical organization. The functions of unloading and loading of the data base can be greatly simplified.

Also, no restructuring will be required because of a change in data base use, since the physical data base organization can be to a large degree independent of its intended use. In addition, the the program conversion problem is simplified as a result of the program interfacing to the DBMS using a high-level logical language.

Similar benefits can be achieved if back-end machines are used. A back-end machine is a special processor dedicated to managing storage and data base on behalf of a host computer. The primary motive for the back-end machine is to offload the data management function from the host to a specialized machine that can execute this function at much lower cost.

From a conversion standpoint, the separation of data management functions from the host promotes the need for a high-level logical interface that provides the advantages discussed above. Another advantage is that it is possible to migrate from one host machine to another without affecting the data bases and their management, alleviating the need

for data conversion if the same back-end machine is used with the new host.

Mass storage devices, such as video disks, make storing very large data bases, in the order of 10 to the 10th power characters, cost-effective. Converting large data bases of this size compounds the cost considerations merely by the processing of this large amount of data. As a result, such data bases will tend to stay in the same environment for longer periods of time. The use of specialized data management machines or dedicated back-end machines in conjunction with these mass storage devices can help postpone the need for data base conversion.

Finally, we should mention the growing use of minicomputers in supporting data management functions. DBMS now exist for many minicomputers, and more are forthcoming. The proliferation of minicomputers which support data bases can only increase the needs for generalized conversion tools.

Much of the work over the past years in the data-management area has concentrated on techniques that clearly separate the logical structure of the data base from its physical organization. This concept, called "data independence," was introduced to emphasize that users need not be exposed to the details of the physical organizations of the data base, but only to its logical relationships.

This led to the development of data access and manipulation languages that depend on the logical data model only. The effect of this trend is similar to that of using specialized data management machines and back-end machines: the simplification of the unload and load functions, since the interface to the DBMS is provided at the logical level only, and the simplification in program conversion for similar reasons.

At the user end of the spectrum, it seems reasonable to assume that the diversity of data models (network, relational, hierarchies and other views that may be developed in the future) will be required for many more decades. This is especially true since there are problem areas that seem to map more naturally into a certain model. Furthermore, users often do not agree on the same model for a given problem area.

Obviously, this state of affairs only accentuates the need to generalize conversion tools that can restructure data bases from one model to another. Even with the development of large-scale associative memories, data structures will likely provide economic rationales for their contrived use. Another possibility is the use of a common underlying data model that can accommodate any of the user views. However, this approach will still require some type of a dynamic conversion process between the common view and each of the possible user views.

There is much work and controversy in developing standards for DBMS. Standards that are oriented to determine the nature of the DBMS are hard to bring about even in a highly controlled environment because of previous investment in application software and data base development and because of disagreement. For example, there is still much controversy whether the network model proposed by the CODASYL committee is a proper one.

It seems reasonable to assume that there will always be nonstandard DBMS. Further, even if a standard can be adopted, different DBMS implementations will still exist, resulting in different physical data bases for the same logical data base. In addition, one can safely assume that restructuring because of application needs will still be necessary, and that changes in the standard itself may require conversion.

A standard that is more likely to be accepted is one that affects only the way of interfacing to a DBMS. In particular, from a conversion standpoint, a standard interchange data form (SIDF) will be most useful. A SIDF is a format not unlike a load format for DBMS. Any advanced DBMS has a load utility that requires sequential data stream in a prespecified format. If a standard for this format can be agreed upon, and if all DBMS can load and unload from and to this format, then the need for reformatting (as described earlier) is eliminated.

The conversion process can be reduced to essentially restructuring only, given that unload and load are part of the DBMS function. A preliminary proposal for such a standard was developed by the ERDA Inter-Working Group on Data Exchange (IWGDE) [GG2]. However, it is only designed to accommodate hierarchical structures. Consideration is now being given to the extension of the standard to accommodate more general structures (networks and relations). We believe there are no technical barriers to the development of a SIDF and that putting such a standard to use would alleviate a major part of the data conversion process.

What can we expect in the next five years and beyond in the data base conversion area? The state of the art has advanced enough to give hope for generalized tools. Within the next five years, we can expect more generalized conversion systems to become operational, but some additional work would be required for moving them from one environment to another.

We can expect to have a standard form developed and agreed upon. It will probably take longer before manufacturers will see the benefit of adopting a standard form and provide load and unload facilities using it. However, we can expect them to provide some conversion tools to convert data bases from other systems to their own.

It will probably take as much as 10 years before the commercial availability of a generalized converter and the adherence of manufacturers to a standard interchange form.

Another area of concern is the application program conversion resulting from a data base conversion. We cannot expect that a generalized solution for this problem will be achieved within the next five years. However, this problem will be simplified to a large extent as hardware and software development trends promote interfacing at the logical level.

BIBLIOGRAPHY

(DL) LOGICAL DATA BASE DESIGN

DL26 Novak, D. and Fry, J., "The State of the Art of Logical Data base Design," *Proceedings of the Fifth Texas Conference on Computing Systems,* IEEE, Long Beach, 1976, pp. 30–39.

DL29 Smith, J. M., and Smith, D. C. P., "Data Base Abstractions: Aggregation and Generalization," *ACM Transactions on Data Base Systems* 2, 2 (1977):105–133.

DL30 Smith, J. M., and Smith, D. C. P., "Data Base Abstractions: Aggregation," *Communications of the ACM* 20,6(1977):405–13.

DL31 Bubenko, J. A., "IAM: An Inferential Abstract Modeling Approach to Design of Conceptual Schema," *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* ACM, N.Y., 1977, pp. 62–74.

DL34 Chen, P. P., and Yao, S. B., "Design and Performance Tools for Data Base Systems," *Proceedings of the Third International Conference on Very Large Data Bases,* ACM, N.Y., 1977, pp. 3–15.

(DT) DATA TRANSLATION

DT1 Fry, J. P., Frank, R. L., Hershey, E. A., III, "A Developmental Model for Translation," *Proc. 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* A. L. Dean (ed.), ACM, N.Y., pp. 77–106.

DT2 Smith, D. C. P., "A Method for Data Translation Using the Stored Data and Definition Task Group Languages," *Proc. of the 1972 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 107–124.

DT3 Ramirez, J. A., "Automatic Generation of Data Conversion Programs Using a Data Description Language (DDL)," Ph.D. dissertation, University of Pennsylvania, 1973.

DT4 Merten, A. G., and Fry, J. P., "A Data Description Approach to File Translation," *Proc. 1974 ACM SIGMOD Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 191–205.

DT5 Housel, B., Lum, V., and Shu, N., "Architecture to an Interactive Migration Systems (AIMS)," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 157–169.

DT6 Rameriz, J. A., Rin, N. A., and Prywes, N. S., "Automatic Conversion of Data Conversion Programs Using a Data Description Language," *Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 207–225.

DT7 Frank, R. L., and Yamaguchi, K., "A Model for a Generalized Data Access Method," *Proc. of the 1974 National Computer Conference,* AFIPS Press, Montvale, N.J., pp. 437–444.

DT8 Taylor, R. W., "Generalized Data Structures for Data Translation," *Proc. Third Texas Conference on Computing Systems,* Austin, Texas, 1974, pp. 6-3-1 ff.

DT9 Univac, UNIVAC 1100 Series Data File Converter, Programmer Reference UP-8070, Sperry Rand Corporation, March, 1974.

DT10 Yamaguchi, K., "An Approach to Data Compatibility, A Generalized Access Method," Ph.D. dissertation, The University of Michigan, 1975.

DT11 Bakkom, D. E., and Behymer, J. A., "Implementation of a Prototype Generalized File Translator," *Proc. 1975 ACM SIGMOD International Conf. on Management of Data,* W. F. King (ed.), ACM, N.Y., pp. 99–110.

DT12 Navathe, S. B., and Merten, A. B., "Investigations into the Application of the Relation Model of Data to Data Translation," *Proc. 1975 ACM SIGMOD International Conf. on Management of Data,* W. F. King (ed.), ACM, N.Y., pp. 123–138.

DT13 Birss, E. W., and Fry, J. P., "Generalized Software for Translating Data," *Proc. of the 1976 National Computer Conference,* Vol. 45, AFIPS Press, Montvale, N.J., pp. 889–899.

DT14 Lum, V. Y., Shu, N. C., and Housel, B. C., "A General Methodology for Data Conversion and Restructuring," *IBM R & D Journal,* Vol. 20, No. 5, 1976, pp. 483–497.

DT15 Housel, B. C., et al., "Express: A Data Extraction, Processing, and Restructuring System," *Transactions on Data Base Systems,* 2,2, ACM, N.Y., 1977, pp. 134–174.

DT16 Swartwout, D. E., Deppe, M. E., and Fry, J. P., "Operational Software for Restructuring Network Data Bases," *Proc. of the 1977 National Computer Conference,* Vol. 46, AFIPS Press, Montvale, N.J., pp. 499–508.

DT17 Goguen, N. H., and Kaplen, M. M., "An Approach to Generalized Data Translation: The ADAPT System," Bell Telephone Laboratories Internal Report, October 5, 1977.

(GG) GENERAL

GG1 Ingals, D., "The Execution Time Profile as a Pro-

gramming Tool," in *Design and Optimization of Compilers,* ed. by R. Rustin, Prentice Hall, 1972, pp. 108–128.

GG2 ERDA Interlaboratory Working Group for Data Exchange (IWGDE) Annual Report for Fiscal Year 1976, NTIS LBL-5329.

GG3 Date, C. J., An Introduction Data Base System, Addison-Wesley, 1975.

GG4 Codd, E. F., "Relational Completeness of Data Base Sublanguage," In Data Base Systems, Caurant Computer Science Symposia Series, Vol. 6, Prentice Hall, 1972.

(M) MODELS-THEORY

M11 Chen, P. P. S., "The Entity-Relationship Model—Towards a Unified View of Data," *Transactions on Data Base Systems* 1, 1(1976):9–36.

(PT) PROGRAM TRANSLATION

PT1 Share Ad-Hoc Committee on Universal Languages, "The Problem of Programming Communication with Changing Machines: A Proposed Solution," *Comm. ACM*, Aug., 1958, pp. 12–18.

PT2 Share Ad-Hoc Committee on Universal Languages, "The Problem of Programming Communication with Changing Machines: A Proposed Solution, Part 2," *Comm. ACM,* Sept. 1958, pp. 9–16.

PT3 Sibley, E. H., and Merten, A. G., "Transferability and Translation of Programs and Data," *Information Systems, COINS IV,* Plenum Press, N.Y., 1972, pp. 291–301.

PT4 Yamaguchi, K., and Merten, A. G., "Methodology for Transferring Programs and Data," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 141–156.

PT5 Housel, B. C., Lum, V. Y., and Shu, N., "Architecture to an Interactive Migration System (AIMS)," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 157–170.

PT6 Mehl, J. W., and Wang, C. P., "A Study of Order Transformation of Hierarchical Structures in IMS Data Bases," *Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ACM, N.Y., pp. 125–140.

PT7 Housel, B. C., and Halstead, M. H., "A Methodology for Machine Language Decompilation," *Proc. of the 1974 ACM Annual Conference,* ACM, N.Y., pp. 254–260.

PT8 Honeywell Information Systems, "Functional Specification Task 609 Data Base Interface Package," Defense Communications Agency Contract DCA 100-73-C-0055.

PT9 Kintzer, E., "Translating Data Base Procedures," *Proc. 1975 ACM National Conference,* ACM, N.Y., pp. 359–62.

PT10 Schindler, S., "An Approach to Data Base Application Restructuring," Working Paper 76 ST 2.3, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Mich. 1976.

PT11 Dale, A. G., and Dale, N. B., "Schema and Occurrence Structure Transformation in Hierarchical Systems," *Proc. 1976 International Conference on Management of Data,* pp. 157–168.

PT12 Su, Stanley Y. W., "Application Program Conversion Due to Data Base Changes," *Proc. of the 2nd International Conference on VLDB,* Brussels, Sept. 8–10, 1976, pp. 143–157.

PT13 Su, S. Y. W., and Liu, B. J., "A Methodology of Application Program Analysis and Conversion Based on Data Base Semantics," *Proceedings of the International Conference on Management of Data,* 1977, pp. 75–87.

PT14 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.

PT15 Su, S. Y. W., and Reynolds, M. J., "Conversion of High-Level Sublanguage Queries to Account for Data Base Changes," *Proc. of NCC,* 1978, pp. 857–875.

(R) RESTRUCTURING

R1 Fry, J. P., and Jeris, D., "Towards a Formulation of Data Reorganization," *Proc. 1974 ACM/SIGMOD Workshop on Data Description, Access and Control,* ed. by R. Rustin, ACM, N.Y., pp. 83–100.

R2 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.

R3 Shoshani, A., "A Logical-Level Approach to Data Base Conversion," *Proc. 1975 ACM/SIGMOD International Conf. on Management of Data,* ACM, N.Y., pp. 112–122.

R4 Shoshani, A., and Brandon, K., "On the Implementation of a Logical Data Base Converter," *Proc. International Conference on Very Large Data Bases,* ACM, N.Y., 1975, pp. 529–531.

R5 Housel, B. C., and Shu, N. C., "A High-Level Data Manipulation Language for Hierarchical Data Structures,"*Proc. of the 1976 Conference on Data Abstraction, Definition and Structure,* Salt Lake City, Utah, pp. 155–169.

R6 Navathe, S. B., and Fry, J. P., "Restructuring for Large Data Bases: Three Levels of Abstraction," *ACM Transactions on Data Base Systems,* 1,2, ACM, N.Y., 1976, pp. 138–158.

ACM, N.Y., 1976, pp. 138–158.

R7 Navathe, S. B., "A Methodology for Generalized Data Base Restructuring," Ph.D. dissertation, The University of Michigan, 1976.

R8 Gerritsen, Rob, and Morgan, Howard, "Dynamic Restructuring of Data Bases With Generation Data Structures," *Proc. of the 1976 ACM Conference,* ACM, N.Y., pp. 281–286.

R9 Swartwout, D., "An Access Path Specification Language for Restructuring Network Data Bases," *Proc. of the 1977 SIGMOD Conference,* ACM, N.Y., pp. 88–101.

R10 Navathe, S. B., "Schema Analysis for Data Base Restructuring," *Proc. 3rd International Conference on Very Large Data Bases,* ACM, N.Y., 1977. To appear in TODS.

R11 Edelman, J. A., Jones, E. E., Liaw, Y. S., Nazif, Z. A., and Scheidt, D. L., "REORG—A Data Base Reorganizer," Bell Laboratories Internal Technical Report, April, 1976.

(SL) STORED-DATA DEFINITION

SL1 Storage Structure Definition Language Task Group (SSDLTG) of Codasyl Systems Committee, "Introduction to Storage Structure Definition" (by J. P. Fry); "Informal Definitions for the Development of a Storage Structure Definition Language" (by W. C. McGee); "A Procedural Approach to File Translation" (by J. W. Young, Jr.); "Preliminary Discussion of a General Data to Storage Structure Mapping Language" (by E. H. Sibley and R. W. Taylor), *Proc. 1970 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ed. by E. F. Codd, Houston, Tex., Nov. 1970, pp. 368–80.

SL2 Smith, D. C. P., "An Approach to Data Description and Conversion," Ph.D. dissertation, Moore School Report 72-20, University of Pennsylvania, Philadelphia, Pa., 1972.

SL3 Taylor, R. W., "Generalized Data Base Management System Data Structures and Their Mapping to Physical Storage," Ph.D. dissertation, The University of Michigan, Ann Arbor, Mich., 1971.

SL4 Fry, J. P., Smith, D. C. P., and TAYLOR, R. W., "An Approach to Stored-Data Definition and Translation," *Proc. 1972 ACM-SIGFIDET Workshop on Data Description, Access and Control,* ed. by A. L. Dean, Denver, Colo., Nov. 1972, pp. 13–55.

SL5 Bachman, C. W., "The Evolution of Storage Structures," *Comm. ACM 15,7* (July 1972), pp. 628–34.

SL6 Sibley, E. H. and Taylor, R. W., "A Data Definition and Mapping Language," *Comm. ACM 16,12 (Dec. 1973), pp. 750–59.*

SL Housel, B., Smith, D., Shu, N., and Lum, V., "Define: A Non-Procedural Data Description Language for Defining Information Easily," *Proc. of 1975 ACM Pacific Conference,* San Francisco, CA, April 1975, pp. 62–70.

SL8 The Stored-Data Definition and Translation Task Group, "Stored-Data Description and Data Translation: A Model and Language," *Information Systems* 2,3 (1977):95–148.

(SM) DATA SEMANTICS

SM1 Schmid, H. A., and Swenson, J. R., "On the Semantics of the Relational Model," *Proc., ACM-SIGMOD 1975 Conference,* May 1975, pp. 211–233.

SM2 Roussopoulos, N., and Mylopoulos, J., "Using Semantic Networks for Data Base Management," *Proc. Very Large Data Base Conference,* Framingham, Mass., Sept. 1975, pp. 144–172.

SM3 Su, Stanley Y. W., and Lo, D. H., "A Multi-level Semantic Data Model," CAASM Project, Technical Report No. 9, Electrical Engineering Dept., University of Florida, June 1976, pp. 1–29.

(TA) TRANSLATION APPLICATIONS

TA1 Winters, E. W., and Dickey, A. F., "A Business Application of Data Translation," *Proceedings of the 1976 SIGMOD International Conference on Management of Data,* Ed. by J. B. Rothnie, Washington, D.C., June 1977, pp. 189–196.

(UR) UM RESTRUCTURING

UR1 Lewis, K., Driver, B., and Deppe, M., "A Translation Definition Language for the Version II Translator," Working Paper 809, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.

UR2 Lewis, K., and Fry, J., "A Comparison of Three Translation Definition Languages," Working Paper DT 5.1, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.

UR3 Deppe, M. E., "A Relational Interface Model for Data Base Restructuring," Technical Report 76 DT 3, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.

UR4 Deppe, M.E., Lewis, K. H., and Swartwout, D. E., "Restructuring Network Data Bases: An Overview," Data Translation Project, Technical Report 76 DT 5, The University of Michigan, Ann Arbor, Michigan, 1976.

UR5 Deppe, M. E., and Lewis, K. H., "Data Translation Definition Language Reference Manual for Version IIA Release 1," Data Translation Project, Working

Paper 76 DT 5.2, The University of Michigan, Ann Arbor, Michigan, 1976.

UR6 Swartwout, D. E., Marine, A. M., and Bakkom, D. E., "Partial Restructuring Approach to Data Translation," Data Translation Project, Working Paper 76 DT 8.1, The University of Michigan, Ann Arbor, Michigan, 1976.

UR7 Swartwout, D. E., Wolfe, G. J., and Burpee, C. E., "Translation Definition Language Reference Manual for Version IIA Translator, Release 3," Data Translation Project, Working Paper 77 DT 5.3, The University of Michigan, Ann Arbor, Michigan, 1977.

(US) UM STORED-DATA DEFINITION

US1 Data Translation Project, "Stored-Data Definition Language Reference Manual," The University of Michigan, Ann Arbor, Michigan, 1972.

US2 Data Translation Project, "Revised Stored-Data Definition Language Reference Manual," The University of Michigan, Ann Arbor, Michigan, 1974.

US3 Data Translation Project, "University of Michigan Stored-Data Definition Language Reference Manual for Version II Translator," The University of Michigan, Ann Arbor, Michigan, 1975.

US4 Birss, E. W., and Fry, J. P., "A Comparison of Two Languages for Describing Stored Data," Data Translation Project, Technical Report 76 DT1, The University of Michigan, Ann Arbor, Michigan, 1976.

(UT) UM TRANSLATION

UT1 "Functional Design Requirements for a Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1972.

UT2 "Design Specifications of a Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1972.

UT3 "Program Logic Manual for the University of Michigan Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.

UT4 "Users Manuals for the University of Michigan Prototype Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.

UT5 "Functional Design Requirements of the Version I Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1973.

UT6 "Program Logic Manual for the University of Michigan Version I Data Translator," Working Paper 306, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1974.

UT7 "Design Specifications: Version II Data Translator," Working Paper 307, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.

UT8 Birss, E., Deppe, M., and Fry, J., "Research and Data Reorganization Capabilities for the Version IIA Data Translator," Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1975.

UT9 Birss, E., et al., "Program Logic Manual for the Version IIA Data Translator," Working Paper 76 DT 3.1, Data Translation Projects, The University of Michigan, Ann Arbor, Michigan, 1976.

UT10 Bodwin, J., et al., "Data Translator Version IIA Release 1 User Manual," Working Paper 76 DT 3.2, Data Translation Project, The University of Michigan, Ann Arbor, Michigan, 1976.

UT11 Bodwin, J., et al., "Data Translator Version IIA Release 2 User Manual," Working Paper 76 DT 3.4, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1976.

UT12 Kintzer, E., et al., "Michigan Data Translator Version IIB Release 1 User Manual," Technical Paper 77 DT 8, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.

UT13 Burpee, C. E., et al., "Michigan Translator Program Logic Manual Version IIB Release 1," Working Paper 77 DT 3.7, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.

UT14 Bakkom, D., et al., "Specifications for a Generalized Reader and Interchange Form," Working Paper 77 DT 6.2, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.

UT15 DeSmith, D., and Hutchins, L., "Michigan Data Translator Design Specifications Version IIB," Working Paper 77 DT 3.8, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.

UT16 Kintzer, E., et al., "Michigan Data Translator Version IIB Release 1.1 User Manual," Technical Paper 77 DT 8.1, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Michigan, 1977.

UT17 Bakkom, D. E., and Schindler, S. J., "Operational Capabilities for Data Base Conversion and Restructuring," Technical Report 77 DT 6, Data Base Systems Research Group, The University of Michigan, Ann Arbor, Mich., 1977.

(Z) RELATIONAL SYSTEM

Z5 Chamberlain, D. C., and Boyce, R. F., "SEQUEL: A Structured English Query Language," *Proceedings of the ACM-SIGMOD Workshop on Data Description, Access and Control,* ACM, N.Y., 1974.