Generalized software for translating data*

by EDWARD W. BIRSS and JAMES P. FRY

*This work was supported by the Department of Defense, Defense Communications Agency, WWMCCS ADP Directorate, Reston, VA Contract No. DCA 100-75-C-0064.

ABSTRACT

Many data processing installations are confronted with the problem of data conversion. Some of the conversion problems are conversion of files foreign to the installation, conversion of files into a data base management system format, and conversion of all data to upgrade hardware or software. Simple file organizations pose few conversion problems, while logically and physically complex data bases emphasize many conversion problems. The current approach of writing specific translation programs is time consuming and frequently inaccurate; a new approach is desirable.

To address these conversion problems, The University of Michigan Data Translation Project has developed a generalized translation methodology. This methodology has been applied in the development of several prototype data translators. These translators have progressively advanced the physical transformation capabilities (reformatting) and the logical transformation capabilities (restructuring). The reformatting capabilities of the translators include the ability to access and modify the physical storage structures which support sequential, indexed sequential, and network organizations. The restructuring capabilities allow complex restructuring of lists, trees, and networks.

Future extensions to the translation methodology include the decomposition of the translation process into small, but specific steps. Languages would be developed to address each of these small translations, and could lead to a generalized accessing mechanism and a data interchange form.

INTRODUCTION

The computer field is a rapidly expanding area with advancements in computer hardware and software technology that are paced by the increasing sophistication and awareness of the users. Expanding utilization of computer facilities and the ever increasing application demands continue to usurp valuable resources. Caught in the middle of this situation is the data processing manager, who must satisfy the demands of the user community and yet maintain the economic attractiveness of the computer system.

One of the many problems facing a data processing installation is the conversion and transformation of data bases. Typically this problem ranges from the conversion of computerized files from other installations (foreign files), the restructuring and reformatting of extant data bases, to the translating of data into various forms required by different applications.

Concomitant with the increased use of data base management systems, the data base administrator is faced with the necessity of creating and/or integrating existing files into data bases or of restructuring existing data bases. The latter capacity, while necessary to make effective use of the data base, is not typically found in data base management systems.

To take advantage of the economic benefits of new hardware/software capabilities and data base management systems, the data processing manager and the data base administrator need a variety of data base conversion tools. The current manual approach of writing specific programs for each conversion is both time consuming and costly. A fresh approach to the problem is therefore needed.

In order to address the data conversion problems of the Data Processing installation, a new software technology has been developing over the past five years called data translation. One group developing a data translation methodology is The University of Michigan Data Translation Project, where several prototype data translators have been developed. These developments and related activities are aimed at providing both the data processing manager and the data base administrator with facilities for foreign file conversion, data base integration, data base restructuring, and data conversion resulting from upgrading hardware or software and changing user requirements.

The purpose of this paper is to describe the progress at The University of Michigan on data conversion based on the development of a data translation methodology. The paper identifies those areas of data base...
conversion in which the translation methodology has been successfully developed and is ready to be applied to current data processing problems. It describes current areas under development and those which are in need of further research.

The paper begins by describing the current research on data base conversion and develops the data translation approach. Next, the evolution of data translators at The University of Michigan is traced in terms of their logical and physical capabilities and in terms of generality achieved. The paper concludes with some observations on a generalized data translation methodology and enumerates those areas which need to be researched.

BACKGROUND AND APPROACH

Within the last five years, a means of attacking the data base conversion problem in a general manner has been proposed. It is interesting to observe that all of these efforts employ some degree of generality and are based on a descriptive approach, which describes the source and target data bases and the necessary transformations required to derive the target data base instances from the source. These descriptions, couched in a high-level declarative language, provide the basis for two implementation approaches for a generalized translator. The interpretive approach develops a generalized processing program, and the generative approach creates a specific translation program for each conversion.

The University of Michigan Data Translation Project's approach, emblematic of others, consists of two steps:

1. The user specification of the necessary descriptions, and
2. The execution of data translator based on these descriptions (Figure 1).

The user supplies descriptions of the logical, physical, and relational aspects of the source and target data bases, along with the specifications of the restructuring transformations required to map source data into the target data. Two languages were developed to provide these descriptions; the Stored-Data Definition Language (SDDL) which is used to describe the source and target data bases, and the Translation Definition Language (TDL) used to describe the restructuring transformations.

The SDDL is based on the language proposed by Taylor. This high-level language uses a modified CODASYL model of data. At first glance, the SDDL is similar to the data description language of a data base management system, but a closer look reveals several important differences. The stored-data definition language, based on common data definition practices, is actually an extension of the logical DDL to the more physical implementation aspects. It describes the mapping of the user logical structure to physical storage structure, the mapping of the physical storage structure to a storage device, and the access paths to the data.

The Translation Definition Language, on the other hand, deals primarily with logical transformations of data and describes the translation of source instances to target instances. The language was developed at The University of Michigan and began as a simple association list of source item names and target item names, but it has since developed into a powerful restructuring language. Detailed discussions of these languages can be found in References 5 and 7.

The SDDL and TDL descriptions are processed by an Analyzer which produces an Encoded Stored-Data Description (ESDD) and an Encoded Restructuring Description (ERD) respectively, which in turn are used to drive the translator.

An auxiliary module which need not be generalized, the DDL Writer, uses the Encoded Stored Data Definition of the target to construct a data definition of the target data base in the language of the target DBMS. Major benefits of this module include not only the immediate use of the data base by application programs, but more importantly, the user verification of the target description. The DDL Writer allows the user to verify in a language familiar to the user that the target data description is consistent with his view of the target data base.

The second step in the translation process is the physical transformation of the source data into the target data. Driven by the data descriptions prepared in the first step, the second step employs three components; a Reader, a Restructurer, and a Writer (Figure 2). The Reader accesses the source data base, the Restructurer transforms the source data into a form suitable for the target, and the Writer creates the target data base.

The Reader module, driven in part by the source ESDD, performs many functions; sequentially accessing physical records, physically deblinking these records, logically identifying their components, and automatically creating an internal data base processable by the Restructurer. In dealing with complex data base structures, the Reader must keep track of
access paths between these unblocked records so that the Restructurer is provided with an accurate representation of the source data base.

The Restructurer accesses the internal representation of the source data base and converts it to a representation of the target data base. The conversion from source to target is a transformation of the logical structure and is directed by the ERD which contains the restructuring specifications.

The Writer, driven in part by the target ESDD, creates the target data base by constructing target data derived from the Restructurer's internal data base.

The approach by The University of Michigan, described above, is an interpretive translation approach in which stored-data descriptions provide parameters to a generalized translation algorithm. An alternative approach, a generative one, creates specific translators for each translation. There is, of course, a hybrid approach in which some of the components in the model of the interpretive approach produce executable code, while other components remain interpretive. The interpretive approach was chosen because it provides a good research tool and facilitates the building of series of data translators.

EVOlUTION OF DATA TRANSLATORS

The translator development at The University of Michigan has designed four translators: a Prototype, Version I, Version II, and Version IIA. Only three of these designs, however, have been implemented; Prototype, Version I, and Version IIA. The various designs were developed to address different problem areas, and each translator design has produced a major contribution to the generalized translation process.

The translator's capabilities are divided into two categories; physical and logical. The Translator's physical capabilities are characterized by the Input/Output processing ability of the Reader and Writer. The translator's logical capabilities are measured in terms of the restructuring proficiency of the translator's Restructurer component.

In the remaining portion of the paper, the purpose, architecture, major contribution, physical capabilities, logical capabilities, generality, and implications of each design are discussed.

Prototype translator

The prototype translator was developed to provide a framework for further research and to verify that the proposed technical approach was sound. The architecture of the prototype translator was very similar to the translator model described in the previous section (Figure 4). The Restructurer, however, only performed an identity transformation, merely associating source item names with target item names.

Prototype’s physical capabilities

There was not a great deal of generality in the prototype; the Reader module only accessed one source data base; and the Writer module only created a single output. The source data base was a NIPS data base unloaded in a variable blocked sequential format (Figure 5A). NIPS ran on the IBM 360/370 series of com-
puters and allowed two-level hierarchical structures with a maximum 256 different groups.

The Writer was also straightforward. It constructed a structure similar to the NIPS data base structure, in the Honeywell 6000 System Standard Format. The H6000 System Standard format was similar to the IBM variable blocked format (Figure 5B).

The Reader and Writer operated with sequential media, and physical concatenation (i.e., a simple mapping algorithm) was used to maintain the hierarchical relationships in both systems. Both systems used similar mapping, so reordering of record types was not necessary.

Prototype's logical capabilities

The Prototype translator does not perform restructuring, and when no records were eliminated, a one-to-one correspondence existed between group instances in the source data base and those in the target data base (Figure 6). Consequently, the Restructurer only processed one group instance at a time. It performed item conversions as necessary and allowed the user to modify the order or storage representation of items within a group.

Generality

The only general module (not coded specifically for the NIPS to H6000 System Standard Format) was the Restructurer. Its generality was possible because of the level of insulation provided by the encoded forms of the SDDL and TDL languages which provided all the necessary information for the table driven architecture of the Restructurer. In contrast to the Restructurer, the Reader and Writer were very specific and used the operating system facilities to perform input/output.

Design implications

Although the prototype performed a very simple translation, its major contribution was the framework for a series of generalized translator implementations. The Prototype design provided a consistent set of interfaces: between the languages and translator modules, through the encoded tables, and among the translator modules by a set of table access routines and common data formats. Throughout the development of generalized translators, this basic architecture has proven to be technically sound.

Version I translator

The Version I translator (Figure 7), although very similar to its predecessor, made some progress toward generality. Version I was developed to perform both the forward and reverse translations (NIPS→System Standard format and System Standard format→NIPS). Furthermore, the COBOL System Standard format files were constrained to a format acceptable to WWDM, a data management system on the H-6000 computer system.

The purpose of this translator was to further verify that the translation methodology was sound and to demonstrate that the translation was reversible without loss of information. The latter proved to be an interesting technique to verify that a translation was correctly performed.

Version I's physical capabilities

The Reader and Writer were required to access and create both NIPS and H-6000 formats (Figure 5B). The Reader and Writer modules were parameterized to enable them to read and write both organizations. Additionally, the Reader was generalized by taking greater advantage of the Encoded SDD. Thus, the Reader's capabilities were generalized to handle most sequential tape formats.
Version I's logical capabilities

The major difference between the Version I translator and the Prototype translator is the addition of a basic restructuring capability. The major objective of Version I was to demonstrate that the inverse translation (source-target-source) was feasible and indeed produced the "original" data base. Restructuring was often required, because the source, COBOL-WWDMS data bases, had eight levels of hierarchy, but the target, NIPS, could only handle a two-level hierarchy. To permit such translation, the first restructuring capability developed was compression of hierarchical levels. For example, a two-level target schema was constructed (Figure 8) from the multiple level source schema. In order to preserve the information of the source data base, the Restructurer created a new group PROD-PARTS and replicated the PARTS information in this new group of target data base.

Generality

The increase in generality of the Version I translator occurred primarily in the Reader and Restructuring components. A new section of the SDDL was developed to provide a better description of physically sequential media. This additional section permitted the debllocking, spanning, and identification operations to be generalized. The Restructurer increased in complexity through addition of the parsing and restructuring functions. These additions to the Restructurer were driven by additions to the SDDL and TDL. Thus, these language additions maintained the generality of the Restructurer component.

The Restructurer became complex, but the software architecture insulated it from the source and target data bases. Its complexity was isolated and could be addressed by revising single modules, as opposed to developing a new translator. The compressing restructuring capacity proved to be quite general. Not only could adjacent levels of the hierarchy be combined, but multiple levels of the hierarchy could be combined into a single level.

Design implications

The Version I translator design required the capability to compress schemas. The compression operation required access to an entry instance. An entry instance can be very large, and a mechanism was added to the translator to give the Restructurer direct access to the entry instance. This mechanism was termed a Virtual Address Space (VAS). The Reader constructed the VAS from group instances contained within one entry instance.

Version II translator design

The objective of the Version II design was to increase the input and output data base classes and to expand the restructuring capabilities of the translator. The architecture of the Version II design was different from the previous translators (Figure 9).

The main difference between the Version II translator and previous designs resulted from the realization that the more complex restructuring operations became, the greater the volume of data required. Such complex restructuring also required broader accessibility to the data base. Because the Restructurer requires many access paths to data, some of which are not available in the source data bases, an internal form of the source data was designed, the Restructurer Internal Form (RIF). A data management function was also incorporated to manage the RIF which allowed the Restructurer to access the RIF directly.

Version II's physical capabilities

The capabilities of the Reader and Writer were to be increased to handle sequential, indexed sequential (ISP), and network (IDS) structures (Figure 10 A,B,C).

These capabilities were to be implemented using a general control structure in the Reader for invoking the specific accessors for the different organizations. The lowest level components were parameterized.
The general problem in restructuring data bases was a very difficult and complex one with no established solutions. The initial research\(^\text{a}\) indicated that restructuring for hierarchical structures was feasible, but was yet to be implemented. Consequently, Version II was designed to perform restructuring within hierarchical structures, which could possibly be subsets of network structures.

The Restructurer's design facilitated the creation of any target logical structure derivable through repeated applications of Expansion, Compression, Partitioning, and Merging restructuring operations on the source logical structure. Examples of these restructuring operations are shown in Figure 11.

**Generality**

Extending the capabilities of the Reader and Writer from Version I's sequential representations to the sequential, indexed sequential, and network representations of Version II had an impact on generality. The sequential Reader of Version I was reasonably general, but the implementation of the SDDL's extended capabilities to handle the representation of complex logical structures on disk media proved extremely difficult. Consequently, accessors, low level routines coded specifically for each organization, were designed for the more complex input/output operations and interfaced with the general software in the Reader. The Restructurer provided an increased level of generality by providing a comprehensive set of restructuring capabilities for hierarchical structures.

**Design implications**

The complex restructuring operations desired for the Version II frequently required access to the entire data base. This requirement led to the development of a Restructurer Internal Form (RIF) in the Translator and the result that the read function had to be performed completely before the Restructurer was invoked. In order to help manage the complex and voluminous RIF data base, a data base management system\(^\text{b}\) was added as a major component of the Translator's design. This DBMS was used not only to provide direct access to information stored in the RIF, but also was used to manage the internal tables.

**Version IIA translator**

After evaluating our design of the Version II translator and some actual data base reorganizations, a decision was made to emphasize restructuring at the expense of the physical capabilities—reading and writing. An alternative design, the Version IIA translator, was developed which focused on the restructuring within the I-D-S data base organizations.\(^\text{11}\) The specific nature of the translator allowed the design effort to focus on user orientation as well as restructuring. It was decided that an expanded version of the I-D-S DDL could be used for the description of the source and target data bases. The I-D-S DDL was augmented by using Level 66 descriptions. Another DDL analyzer, specific to I-D-S, had to be developed, which not only processed the extended DDL but also produced the Encoded SDD (Figure 12).

**Version IIA's physical capabilities**

Since the Version IIA had to restructure I-D-S data bases, the input/output class of the Reader and Writer were limited to these structures (Figure 10C). However, the Reader was generalized to populate the RIF from multiple source I-D-S data bases which allowed the integration of I-D-S data bases, the addition of new data and structure, and the addition of new relationships among existing data instances.
The Writer, since it was coded for I-D-S data bases and used the I-D-S access methods, could perform limited storage optimization through reformatting of the target data base. The various storage parameters (place near, . . .) available in the I-D-S DDL could be used to create more efficient target data bases.

**Version IIA’s logical capabilities**

The Restructurer made a major research and development step by extending its capabilities to network structures. The Restructurer implemented was extremely powerful and could not only change the implementation structure of existing I-D-S relationships, but also had the capability to create structures more powerful than I-D-S I could process. Some of these restructuring capabilities developed in addition to the Version I capabilities are shown in Figure 13A-13C.

**Generality**

Overall the Version IIA translator is less general than the Version II design chiefly because the Reader and Writer were tailored to process I-D-S data base thereby simplifying the coding effort. The specific approach to the Reader and Writer stemmed from the Version IIA emphasis on the Restructurer, and the difficulty of implementing the extended SDDL to describe complex structures residing on disk media. The Restructurer, however, maintained its generality by the table driven architecture and greatly expanded its capabilities from hierarchical to network structures.

**Summary of capabilities**

During the evolution of the Michigan data translators, both the Physical and Logical Capabilities of the translators were increased in incremental steps. The capabilities of the Reader and Writer modules have been extended from sequential to network data base representations with some loss of generality. Restructuring capabilities have increased from simple renaming of structures, through hierarchical restructuring to network restructuring. Although it was necessary to expand the Restructurer’s accessibility of the data base from a single record to the entire data base in order to achieve these sophisticated restructuring capabilities, the generality of the Restructurer was preserved through the RIF and the DBMS. In addition to the translation capabilities, it became clear that additional capabilities of adding data and new relationships to the data base were necessary and implemented in the Version IIA. The evolution of the basic capabilities of the various Michigan translators are summarized in Figure 14.
The interdependency of the logical and physical aspects of data, coupled with the dependence of the physical transformation on the hardware/software environment, makes generalized data base conversion a complex but challenging problem. The data translation methodology developed separates this problem into a logical and physical transformation processes which allow specific modules to address these transformations. Significant results have been achieved in generalizing the logical transformation process—the research, development and implementation of a generalized Restructurer. Although a generalized Restructurer has been shown to be technically feasible, its economic justification has not been demonstrated. The economic feasibility of the approach is in doubt because of the slow execution speed of the Restructurer (Version IIA), however, an optimization effort has been initiated which should greatly improve the efficiency of this process.

The achievement of complete generality in the physical transformation process has yet to be achieved. It appears much more difficult since the deeper one goes into the actual representation, the more complex the description process and the implementation of the physical transformation modules. For example, inverted structures that are implemented in SYSTEM 2000 and ADABAS are extremely difficult to translate because the SDDL for such organizations must not only describe the data, but also describe the format and semantics of the indices. Extending this part of the SDDL, the Reader, and the Writer modules to handle these organizations would require a substantial effort. At this point in the research, the development of a completely general physical transformation modules does not appear to be economically justifiable.

Building on this result, a vehicle for further decomposing this complex process into its physical and logical components needs to be developed. One such candidate is a common Data Interchange Form (DIF) for the transferability of structured data bases. Although the development of the Data Interchange Form is not easy and requires an additional transformation in the translation process, it would nevertheless be a simplifying ingredient. The design and specification of a common Data Interchange Form would result in:

(i) Separation of the hardware/software environmental considerations;
(ii) Development of specific accessors and constructors to address the physical transformation process;
(iii) Distribution of the effort between the source and target machines;
(iv) Development of a more efficient internal form of data for the Restructurer.

Further, the development of a Data Interchange
Form would facilitate the general transportability of structured data bases. Such transportability would not only occur using interchange media (e.g., tapes), but also occur over communication networks (e.g., ARPANET).

Another argument for a Data Interchange Form is that it allows the translation process to be more explicitly decomposed. The stored-data definition language could be divided according to individual transformation, which would allow different forms and types of the language to address the differing levels of detail in the translation process. The translation modules would be more dependent on their respective environments and, consequently, easier to build. Further, such a decomposition would allow optimization to take place at different stages in the translation process.

The design and specification of a common Data Interchange Form still requires research and development. It must be self-describing in that it would access interpretively, independent of the environment in which it was created. The Data Interchange Form must be logically capable of expressing the most sophisticated data base structure known, but be physically simplistic without being overly inefficient. Consequently herein lies a direction for future research.

REFERENCES
